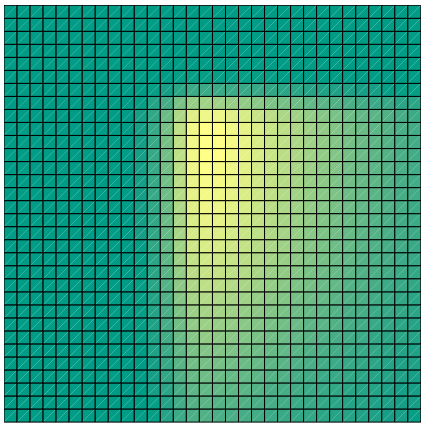
N o r b e r t J a n k o w s k i

Ontogeniczne sieci neuronowe

O sieciach zmieniających swoją strukturę



Warszawa 2003

Opracowanie książki było wspierane stypendium
Uniwersytetu Mikołaja Kopernika

Spis treści

Spis rysunków

Spis tabel

Wprowadzenie

Dziś chyba już z pewnością możemy powiedzieć, że właśnie *w stronę uśmiechniętych maszyn*[1] kierują się liczne badania przełomu wieku XX i XXI. Do tego stanu rzeczy niewątpliwie przyczynił się dynamiczny rozwój technologiczny komputerów, który niewątpliwie mobilizował rozwój informatyki.

Początek istnienia komputerów to czas, w którym można było je znaleźć jedynie na uniwersytetach lub w instytucjach naukowo-badawczych. W ostatnich latach ogromnemu zwiększeniu uległa moc obliczeniowa komputerów, jak i możliwości ich integracji ze środowiskiem (możliwości sieci komputerowych, możliwości jakie daje łączenie komputerów z wieloma typami urządzeniami zewnętrznymi). Dodatkowo niezwykła chłonność rynku na sprzęt komputerowy jaką mogliśmy obserwować w ostatnich latach sprawiła, że ceny komputerów, o których jeszcze nie dawno nie można było marzyć, stały się przystępne dla kieszeni obywateli krajów *w miarę* rozwiniętych, otwierając w ten sposób możliwości ich szerokiego zastosowania.

Obecna moc obliczeniowa komputerów pozwoliła znacznie zwiększyć rozmiar problemów jakie można rozwiązywać. Z drugiej strony dziś można efektywnie rozwiązywać nie tylko problemy, których złożoność jest wielomianowa, ale również dość skutecznie rozwiązywać sporą część problemów NP-zupełnych, których do niedawna w ogóle nie można było rozwiązywać. Oczywiście *racjonalne* rozwiązywanie problemów NP-zupełnych sprowadza się do coraz lepszych rozwiązań przybliżonych, ale na tyle dobrych, by były wręcz nieodróżnialne od rozwiązań idealnych, bądź stanowiły rozwiązania satysfakcjonujące, które umożliwią ich użycie w praktyce.

W realnych zastosowaniach na brak trudnych (NP-zupełnych) problemów nie można narzekać. Jest ich wręcz za dużo. Już choćby takie sztandarowe problemy jak gra w szachy, czy problem komiwojażera, są na to dowodem. O więcej przykładów naprawdę nietrudno, wystarczy spojrzeć na typowe problemy w przemyśle, na przykład problemy optymalizacyjne, czy niezwykle szeroki wachlarz problemów współczesnej medycyny, których rozwiązanie najczęściej polega na *inteligentnym* przetwarzaniu informacji.

Trzeba pamiętać jednak, iż moc obliczeniowa komputerów to jedynie czynnik niezbędny do rozwiązywania takich problemów. Rozwiązywanie trudnych

[1] Tytuł książki prof. R. Tadeusiewicza [242].

problemów staje się możliwe przede wszystkim dzięki rozwojowi nowych algorytmów obliczeniowych, które najczęściej stanowią połączenie pewnej wiedzy o problemie z metodami przetwarzania i wykorzystywania tej wiedzy. Taka metodologia postępowania jest dziś spotykana w rozmaitych aplikacjach.

Niewątpliwie w obecnych czasach coraz częściej będą poszukiwane systemy, które będą w stanie możliwie *inteligentnie* wynajdywać i przetwarzać informacje. Będzie (czasami już jest) to spowodowane coraz bogatszymi źródłami informacji, lecz informacji, która nieprzetworzona nie będzie miała żadnej wartości. Na myśli mam wszelakie źródła informacji, których jest wciąż coraz więcej, poczynając od Internetu, rozlicznych baz wiedzy/informacji do przeróżnych systemów pomiarowych w technice (zaawansowany przemysł, biotechnologia, nowoczesna aparatura medyczna, technika wojskowa, etc.).

Ogromną część problemów stanowią różnego typu analizy uprzednio zebranych danych, analizy obrazów, klasyfikacja i rozpoznawanie wzorców, prognozowanie itp. Gałęzie nauki, które zajmują się rozwiązywaniem tego typu problemów, można objąć wspólną nazwą *metod inteligencji obliczeniowej*. Do metod inteligencji obliczeniowej z pewnością zaliczyć można sztuczne sieci neuronowe, uczenie maszynowe, metody regresji i estymacji, statystykę, teorie filtrów adaptacyjnych, modelowanie Bayesowskie, logikę rozmytą, teorię zbiorów przybliżonych, algorytmy ewolucyjne, metody drążenia danych, modelowanie koneksjonistyczne, neuroinformatykę. Większość modeli wyrosłych z powyższych dziedzin mają bardzo ważną wspólną cechę, mianowicie są to *metody uczenia się z danych*[2]. Na polskim rynku wydawniczym do tej pory ukazały się już książeki, które także dotykają tych problematyk [18, 239, 220, 159, 144, 221, 246, 198, 199, 218, 219, 240, 73, 46, 262] [3]. Mam jednak nadzieję, że książka, którą mają państwo w rękach będzie miłym dopełnieniem stanu wiedzy z zagadnień dotyczących uczenia się modeli adaptacyjnych ze szczególnym uwzględnieniem sztucznych sieci neuronowych.

Materiał poniższej monografii trudno sklasyfikować tylko do jednej z powyżej wspomnianych gałęzi metod inteligencji obliczeniowej. Choć niewątpliwie większość materiału jest bezpośrednio związana ze sztucznymi sieciami neuronowymi, to nietrudno dopatrzyć się metod uczenia maszynowego, statystyki, teorii filtrów adaptacyjnych, czy metod wizualizacji.

Pierwszy rozdział stanowi obszerne omówienie funkcji transferu sztucznych sieci neuronowych, czyli funkcji realizowanych przez poszczególne sztuczne neurony. Funkcje transferu mają ogromny wpływ na własności sieci i tym samym na możliwości sztucznych sieci neuronowych. Dlatego też w tym rozdziale zebrano informacje o wielu funkcjach transferu. Zaprezentowano również ich nowe, bardziej efektywne wersje, które można zastosować do wielu znanych modeli.

Dokonano systematycznego omówienia funkcji aktywacji, podzielonych na funkcje bazujące na iloczynie skalarnym, mierze odległości (lub podobieństwa)

[2] *Uczenie się z danych (ang. Learning from data)* — tytuł książki V. Cherkasskiego i F. Muliera

[3] Cytowania w kolejności chronologicznej.

i ich kombinacji. Po funkcjach aktywacji przedstawiono funkcje wyjścia: sigmoidalne, zlokalizowane i semi-centralne. Zaproponowane taksonomie są pierwszą tego typu próbą systematyzacji wiedzy o funkcjach, które mogą być realizowane przez sztuczne neurony. Następnie zostały przedstawione funkcje transferu, jako złożenia funkcji aktywacji z funkcjami wyjścia. Najpierw przedstawiono funkcje nielokalne, następnie lokalne, semi-lokalne i uniwersalne.

Kolejna część rozdziału obejmuje nowe funkcje transferu, wśród których dużą grupę stanowią funkcje bicentralne. Zostały opisane formy podstawowe funkcji bicentralnych, jak i ich ciekawe rozszerzenia, które umożliwiają osiągnięcie jeszcze większej elastyczności na przykład poprzez wykorzystanie obrotu w wielowymiarowej przestrzeni, czy delokalizację. W końcowej części rozdziału dokonano tabelarycznego porównania ważnych własności funkcji transferu omówionych w tym rozdziale. Zaproponowano także hierarchiczne uporządkowanie funkcji transferu pod względem ich elastyczności.

Drugi rozdział omawia różne aspekty sieci neuronowych z radialnymi funkcjami bazowymi (RBF). Początek rozdziału to omówienie podstaw sieci RBF. Następnie przedstawione zostały metody inicjalizacji sieci typu RBF. Potem omówiono standardowe, jak i mniej znane metody uczenia sieci RBF. Zaprezentowane zostały człony regularyzacyjne stosowane w sieciach RBF. Dokonano także porównania wielowarstwowych sieci perceptronowych (MLP) z sieciami RBF.

Końcowa część rozdziału poświęcona jest sieciom probabilistycznym, które mają także spore więzi z modelami RBF.

Kolejny rozdział (trzeci) poświęcono bardzo związanemu z siecią RBF modelowi Support Vector machines (SVM). Przedstawiono kilka typów modeli SVM związanych z klasyfikacją i regresją. Omówiono także algorytm uczenia modelu SVM i własności zbieżności. Przedstawiona została także rozbudowana wersja modelu SVM nazwana Meta-SVM.

Rozdział czwarty obejmuje omówienie sieci ontogenicznych ze szczególnym uwzględnieniem ontogenicznej sieci IncNet jak i sieci optymalnych funkcji transferu. Pierwsza część omawia modele, które umożliwiają usuwanie wag lub neuronów ze struktury sieci neuronowej. Druga część rozdziału omawia modele, których struktura rozrasta się podczas procesu adaptacji. Wskazano liczne wady, zalety i ograniczenia przedstawionych modeli ontogenicznych. Omówiona została również sieć z przydziałem zasobów (RAN).

Pozostała część rozdziału to wstęp i omówienie sieci *Incremental Network* (IncNet). Opisano, jak można zastosować rozszerzony filtr Kalmana (EKF) do uczenia sieci typu RBF. Zaproponowano także nową odmianę rozszerzonego filtra EKF o mniejszej złożoności obliczeniowej, dzięki której można prowadzić adaptację bardziej złożonych problemów. Zaproponowano nowe, statystyczne metody kontroli złożoności sieci neuronowych. Do zastosowań klasyfikacyjnych została zaprezentowana sieć, która składa się z klastra podsieci IncNet i modułu decyzyjnego.

Następnie opisano możliwości diagnostyczne współczynników, które są wyznaczane przez wspomniany klaster sieci IncNet i moduł decyzyjny, w tym

także prawdopodobieństwa przynależności klasyfikowanych wektorów do poszczególnych cech. Opisano także własności różnych innych możliwości kontroli sieci IncNet.

W dalszej części rozdziału zaproponowano używanie *przedziałów ufności*, które stanowią bardzo silną alternatywę dla reguł logicznych. Zaproponowano także bardzo ciekawe metody wizualizacji w oparciu o przedziały ufności, jak i ich rozwinięcia, których celem jest wspomaganie procesu diagnozy, szczególnie w medycynie.

Następnie zaprezentowano sieć optymalnych funkcji transferu, która jest wyposażona w mechanizm kontroli złożoności i używa heterogenicznych funkcji transferu.

Rozdział piąty omawia techniki budowania i uczenia komitetów modeli, czyli modeli złożonych z innych modeli. Omówiono różne modele komitetów i ich zastosowań, jak również ich wpływ na złożoność modelu ostatecznego jak i wpływ na generalizację i wariancję błędu.

Przedstawiono komitety wykorzystywane w problemach wieloklasowych, dwuklasowych jak i takie które mogą być używane w klasyfikacji i aproksymacji. Omówiono komitety z głosowaniem, ważeniem i ich pochodne. Opisane są także bardziej zaawansowane komitety jak AdaBoosting czy Stacking, jak i komitety złożone z modeli heterogenicznych.

Następny, szósty rozdział zawiera opis metod przetwarzania wstępnego danych. Transformacje te mogą odegrać kluczowe znaczenie i ich wpływ może okazać się (czy też po prostu jest zawsze) ogromny. Omówione zostały metody transformacji danych, postępowania z wartościami brakującymi, opisano metody selekcji i ważenia cech. Opisano także zupełnie nową metodę, której zadaniem jest regularyzacja danych, która może być wykorzystana w analizach danych lub modelach adaptacyjnych.

Rozdział siódmy prezentuje zastosowania sieci IncNet dla realnych i sztucznych danych. W rozdziale omówione zostały metody porównania modeli, transformacje danych (standardowe jak i nowe), problemy wartości nietypowych i wartości brakujących, oraz ważniejsze aspekty metod selekcji cech.

Pierwszy przykład zastosowania sieci IncNet, to analiza danych psychometrycznych. Celem jest klasyfikacja pacjentów do odpowiednich typów nozologicznych w oparciu o wykonywane testy psychometryczne i w rezultacie poprawienie jakości klasyfikacji dokonywanej obecnie przez psychologów. Dokonano szczegółowej analizy otrzymanych rezultatów dla różnych końcowych sieci IncNet. Kolejne zastosowania sieci IncNet, to problemy klasyfikacji raka piersi, zapalenia wątroby, cukrzycy, zapalenia wyrostka i chorób tarczycy. Wszystkie zastosowania zostały omówione i porównane z innymi, najlepszymi obecnie klasyfikatorami dla danych baz.

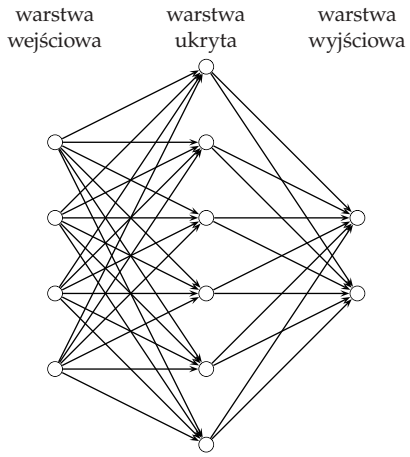
Jako uzupełnienie powyżej wspomnianych zastosowań zostały dołączone zastosowania sieci IncNet w problemach aproksymacyjnych. Zastosowano sieć IncNet do aproksymacji czterech przykładowych funkcji i porównano rezultaty z kilkoma modelami.

Niniejsza książka stanowi rozwinięcie mojej pracy doktorskiej. Prace nad rozwojem monografii, jak i samo wydanie wspierane było przez Uniwersytet Mikołaja Kopernika za co chciałbym serdecznie podziękować.

Ze względów na koszt druku nie można było umieścić większości ilustracji w kolorze. Jednakże na stronie **http://www.phys.uni.torun.pl/norbert/ontogen** znajduje się dokument w formie elektronicznej ze wszystkimi kolorowymi ilustracjami znajdującymi się w książce.

Bardzo krótkie wprowadzenie do sieci neuronowych

Zanim przejdziemy do omawiania szczegółów dotyczących funkcji transferu, przyjrzymy się budowie sztucznych sieci neuronowych i ich działaniu. Poniższy fragment rozdziału jest tylko pobieżnym wprowadzeniem do tematyki sztucznych sieci neuronowych. Dlatego też osoby, które napotkają trudności w rozumieniu dalszych części materiału zachęcam do zapoznanie się choćby z jedną z następujących pozycji książkowych [239, 159, 221, 246, 198, 219, 240]. Sztuczne sieci neuronowe z informatycznego punktu widzenia to nic innego jak grafy z odpowiednio określoną rolą węzłów i krawędzi. Sieć neuronowa to graf skierowany. Oznacza to, że krawędzie łączące węzły grafu (czyli neurony sieci), są jednokierunkowe. Choć bywa wśród sieci neuronowych z rekurencją, że pary neuronów są połączone w obu kierunkach, tworząc cykl.



Rysunek 1: Przykład sieci neuronowej z jedną warstwą ukrytą.

Przykład prostej sieci neuronowej można zobaczyć na rysunku 1. Jest to sieć, która składa się z trzech *warstw* neuronów, co jest dość typowe (np. dla sieci typu RBF, por. rozdział 2). Neurony należące do tej samej warstwy najczęściej mają takie same własności i rolę w sieci neuronowej. Na przykład na wspomnianym już rysunku pierwsza od lewej to warstwa neuronów wejściowych. Takie

neurony warstwy wejściowej tworzą źródło informacji dla całej sieci neuronowej. Właśnie od tych neuronów informacja jest propagowana dalej zgodnie z kierunkiem połączeń (krawędzi grafu skierowanego) pomiędzy neuronami warstwy wejściowej, a pozostałymi neuronami. Sposób przepływu informacji pomiędzy neuronami regulowany jest przez odpowiednie funkcje, które są przypisane do właściwych typów neuronów — temu właśnie będzie poświęcony ten rozdziału. Warstwa po prawej części rysunku to warstwa neuronów wyjściowych. Właśnie wartości neuronów wyjściowych stanowią wynik, który jest związany pewną relacją, jaka zachodzi pomiędzy wejściem i wyjściem. Spowodowanie, aby sieć realizowała określoną relację pomiędzy wejściem i wyjściem jest głównym celem procesu *uczenia sieci*, jak i *doboru* jej *struktury*. Bywa, że te dwa etapy przebiegają równocześnie. Wtedy mamy do czynienia z *sieciami ontogenicznymi*, czyli takimi, które same korygują swoją strukturę. Korekcje struktury mogą polegać na zmianie liczby neuronów lub połączeń. Relacja pomiędzy wejściem i wyjściem sieci neuronowej może odpowiadać rozpoznawaniu pisma ręcznego czy syntezie głosu ludzkiego. W przypadku rozpoznawania pisma wejściem sieci jest odpowiednio przekształcony obraz pisma, a wyjściem może być znak. Jeśli proces ten przebiega dobrze (z powodzeniem) mówimy, że sieć neuronowa dobrze nauczyła się rozpoznawać pismo ręczne, bądź dokonywać syntezy dźwięku. Oczywiście sieci neuronowe mogą i są wykorzystywane na wielu polach nauki, technologii, medycyny i nie tylko. Uczenie sieci neuronowej polega na adaptacji wolnych parametrów sieci, czyli na zmianie wartości wag związanych z krawędziami grafu (czasem także innych wolnych parametrów, które jeśli nie będą związane z pewnymi krawędziami grafu, to będą związane z pewnymi neuronami). Za adaptację parametrów sieci odpowiada *algorytm uczenia sieci*. Bardzo często sieć neuronowa oprócz warstwy wejściowej i wyjściowej ma jedną bądź więcej warstw ukrytych. Ich zadaniem jest z jednej strony zwiększenie pojemności sieci (możliwości adaptacyjnych), a z drugiej umożliwienie odzwierciedlania przez sieć znacznie bardziej skomplikowanych relacji, najczęściej umożliwiając tworzenie odwzorowań nieliniowych. Czasem kolejne warstwy ukryte mogą odpowiadać różnym typom funkcjonalności (różnym filtrom, transformacjom). Liczba warstw, jak i liczby neuronów w poszczególnych warstwach ukrytych powinny zależeć od złożoności problemu jaki ma być rozwiązywany przez daną sieć neuronową. Rozmiar struktury zależy także od algorytmu uczenia, który sam w sobie może narzucać pewne ograniczenia na strukturę sieci neuronowej (zazwyczaj na liczbę warstw, rzadziej na liczbę neuronów).

Rozdział 1

Funkcje transferu

Wybór funkcji transferu ma niezwykle duży wpływ na możliwości działania sieci neuronowych. Chociaż funkcje sigmoidalne jako funkcje transferu są powszechnie stosowane nie ma powodu, aby to one były optymalne we wszystkich przypadkach. Przedstawione zostaną tu zalety i wady wielu różnych funkcji transferu jak i szeregu nowych funkcji transferu posiadających większe możliwości. Przedstawiona zostanie również propozycja taksonomii funkcji aktywacji i funkcji wyjścia. Będą opisane również uniwersalne funkcje, które poprzez zmianę parametrów stają się lokalne lub nielokalne, albo nielokalne w pewnych podprzestrzeniach, a w innych podprzestrzeniach lokalne. Również i inne funkcje zostaną zaprezentowane, włączając w to funkcje bazujące na nieeuklidesowej mierze odległości. Następnie wprowadzone zostaną funkcje bicentralne, które powstają jako liniowy produkt par funkcji sigmoidalnych. Taki produkt składający się z N funkcji bicentralnych w N wymiarowej przestrzeni jest w stanie reprezentować o wiele większą klasę gęstości prawdopodobieństw wejściowej przestrzeni wektorów, niż np. typowa wielowymiarowa funkcja gaussowska. Przedstawione są też różne możliwości rozszerzeń funkcji bicentralnych, które mogłyby stanowić pewien złoty środek pomiędzy złożonością samej sieci, a jej możliwością do uczenia się. Funkcje bicentralne i ich rozszerzenia mogą być z powodzeniem stosowane do różnych sieci neuronowych w szczególności do jakich jak RBFN, RAN, IncNet i FSM. Z kolei, używając takich funkcji i wymuszając ostre granice (duże skosy), podążamy do logicznej interpretacji sieci neuronowej.

Powstanie sztucznych sieci neuronowych jako systemów adaptacyjnych było początkowo motywowane możliwościami przetwarzania informacji mózgu ludzkiego [118, 13, 216]. Pojedyncze sztuczne neurony, jak i architektury sztucznych sieci neuronowych mają niewiele wspólnego z prawdziwą biologiczno–logiczną budową mózgu. Sztuczne sieci neuronowe są sieciami złożonymi z prostych elementów, nazywanych neuronami, które posiadają parametry adap-

tacyjne $\mathbf{w}$. Modyfikacje tych parametrów prowadzą do uczenia się przez sieć odwzorowania wektora $\mathbf{x}$ z przestrzeni wejściowej do przestrzeni wyjściowej $y = A_{\mathbf{w}}(\mathbf{x})$ (w ogólności y może być także wektorem). Ze statystycznego punktu widzenia systemy adaptacyjne powinny charakteryzować się zbieżnością funkcji decyzyjnej (czyli funkcji określającej granice decyzji) do optymalnej funkcji decyzyjnej dla rozkładu prawdopodobieństwa łącznego $p(\mathbf{x}, y)$ lub chociaż prawdopodobieństwa warunkowego $p(y|\mathbf{x})$. Do estymacji granic decyzji rozkładu prawdopodobieństwa konieczna jest *adaptowalność* kształtu powierzchni funkcji transferu i właśnie to stanowi o *sile* adaptacyjnej sieci neuronowej.

Sztuczne sieci neuronowe są systemami, które posiadają moc obliczeniową komputera uniwersalnego, tj. mogą realizować dowolne odwzorowanie z jednej przestrzeni (wejściowej) do drugiej (wyjściowej). Różnią się pod wieloma względami, lecz wspólną cechą jest obliczanie wartości funkcji transferu przez każdy neuron. Pierwszymi modelami sztucznych sieci były sieci logiczne [180] lub urządzenia progowe, obliczające funkcje schodkową. Funkcje schodkowe zostały następnie uogólniane do funkcji o kształcie sigmoidalnym. Pokazano też, że sieć neuronowa z jedną warstwą ukrytą z funkcjami sigmoidalnymi jest uniwersalnym aproksymatorem [52, 122], tj. może aproksymować dowolną ciągłą funkcję z dowolną dokładnością przy wystarczającej liczbie neuronów. Taką samą własność mają sieci z funkcjami gaussowskimi, użytymi w miejsce funkcji sigmoidalnych [114, 203].

Nowy typ funkcji transferu zwanych wstęgowymi (*gaussian bars*) został zaproponowany przez Hartmana i Keelera[113]. Pao zaprezentował nowy typ sieci (*functional link networks*) [201], w którym wykorzystano kombinacje różnych funkcji, takich jak wielomiany, funkcje periodyczne, funkcje sigmoidalne i gaussowskie. Haykin i Leung proponują użycie *rational transfer functions* i prezentują bardzo dobre wyniki przy użyciu tych funkcji transferu [165]. W pracy Dorffnera [56] prezentowane są funkcje stożkowe, które gładko zmieniają się od funkcji o kształcie sigmoidalnym do funkcji zbliżonej do funkcji gaussowskiej. Można też użyć funkcji Lorentzowskiej, jako uproszczenia funkcji gaussowskiej zaproponowanej przez Girauda i in. [104]. Te prace, jak i sporo innych, pokazują, iż wybór funkcji transferu jest istotny i tak samo ważny jak i dobór architektury sieci czy algorytmu uczenia.

Sieci neuronowe są używane do aproksymacji rozkładu prawdopodobieństwa dla klasyfikacji lub do aproksymacji gęstości prawdopodobieństwa zbioru danych treningowych [13, 216]. Żadne z powyżej wspomnianych funkcji nie są wystarczające do reprezentacji rozkładu prawdopodobieństwa wielowymiarowej przestrzeni wejściowej przy użyciu małej liczby parametrów. Problem uczenia, z geometrycznego punktu widzenia, można przestawić jako cel, którym jest wybór takiej przestrzeni funkcji i ich parametrów, które dają jak największą adaptowalność kształtu aproksymowanej funkcji przy użyciu jak najmniejszej liczby parametrów adaptacyjnych.

Żadne z powyżej wspomnianych funkcji transferu nie są wystarczająco elastyczne do opisu powierzchni decyzji złożonych danych z wielowymiarowej przestrzeni wejściowej, przy użyciu małej liczby parametrów adaptacyjnych. Do

testowania metod adaptacyjnych statystycy preferują sztuczne dane [117, 92]. Jest oczywiste, iż pewne rozkłady danych są łatwo aproksymowane przy użyciu funkcji zlokalizowanych (np. funkcji gaussowskich), a inne rozkłady są prostsze w aproksymacji wykorzystując funkcje nielokalne (np. funkcje sigmoidalna z aktywacją w postaci liniowej kombinacji wejść). W [117] rozważany był problem o N wymiarowej przestrzeni wejściowej, w którym wektory znajdujące się wewnątrz pewnej sfery należą do jednej klasy, a na zewnątrz do drugiej. Łatwo zauważyć, iż do rozwiązania takiego problemu wystarczy jedna wielowymiarowa funkcja gaussowska z $2N$ parametrami adaptacyjnymi (na centrum i rozmycia). Jednakże rozwiązanie tego samego problemu wymaga wielu hiperpłaszczyzn tworzonych przez funkcje sigmoidalne. Najprostsza możliwa sieci MLP, która rozwiązała by powyższy problem musi skonstruować sympleks przy użyciu N funkcji sigmoidalnych i jednego dodatkowego neuronu na wygładzenie powierzchni, co stanowi $N^2 + N$ parametrów adaptacyjnych i znacznie komplikuje proces uczenia. Z kolei, w innym problemie, gdy do pierwszej klasy zakwalifikować punkty z rogu układu współrzędnych, ograniczając obszar płaszczyzną $(1, 1, ..., 1)$, to wystarczy jedna płaszczyzna ($N + 1$ parametrów), aby rozdzielić dwie klasy. Natomiast znacznie trudniej jest rozwiązać problem przy użyciu funkcji gaussowskich. Umieszczając jedną funkcję w centrum obszaru i $N + 1$ po rogach wymaga $2N(N + 2)$ parametrów nie rozwiązuje się idealnie problemu, a i znacznie utrudnia się proces adaptacji. Usprawnianie algorytmów uczenia lub struktur sieci nie będą wystarczające, gdy obszary decyzyjne będą złożeniem funkcji *sferycznych* lub hiperpłaszczyzn.

Poniżej rozważane są różne funkcje transferu dla sztucznych sieci neuronowych. Jednak nie jest celem tego rozdziału przedstawienie wszelkich prac, jakie były prowadzone na ten temat. Anderson [6] uzasadnia użycie funkcji sigmoidalnych dla motoneuronów, lecz przejście od neuronów impulsowych (*ang. spiking neurons*) kory mózgowej (jej asocjacyjnej funkcji) do modelu, w którym używa się ciągłych funkcji transferu, nie jest trywialne (teoretyczne wprowadzenie w modele oparte o neurony impulsowe można znaleźć w [176]). Bardzo ciekawym aspektem jest też budowanie neuronów analogowych lub modeli sprzętowych [186, 266, 126], lecz ten temat również wykracza już po za główny temat pracy. Nie będą też rozważane funkcje używane w modelach asocjacyjnych, takie jak funkcje monotoniczne [156, 189, 271, 267, 268], funkcje periodyczne [269, 269, 152, 192] i neurony chaotyczne [99, 270]. Te ostatnie mogą być bardziej przydatne w neurobiologi i mogą unikać złudnych lokalnych minimów funkcji błędu. Także w rozmytych sieciach neuronowych używa się specjalnych funkcji transferu, te również zostaną pominięte. Pominięty zostanie też model *neuronu złożonego* (por. [238]).

Ciekawą rzeczą okazało się sporządzenie systematycznego przeglądu przeróżnych funkcji transferu dla sieci neuronowych, jak i taksonomii funkcji aktywacji i wyjścia, ponieważ, jak dotąd, informacje te w literaturze były zupełnie rozproszone poza nielicznymi wyjątkami, które prezentują funkcje alternatywne do funkcji sigmoidalnej. Część z funkcji, które zostały zaprezentowane poniżej,

nigdy nie były jeszcze użyte.

W poniższym podrozdziale przedstawiono ogólne pojęcia związane z opisywaniem funkcji transferu. W następnym podrozdziale przedstawiono szeroki opis funkcji aktywacji neuronu. Opis obejmuje szeroki wachlarz różnych miar odległości. Kolejny podrozdział przedstawia przeróżne funkcje wyjścia, po czym następuje podrozdział, w którym przedstawiono różne funkcje transferu, podzielone na kilka grup. Porównywanie rezultatów uzyskanych za pomocą różnych funkcji transferu jest przedsięwzięciem bardzo trudnym. Różne funkcje mogą być użyte w bardzo różnych sieciach. Również i sposób inicjalizacji sieci może prowadzić do bardzo zróżnicowanych wyników. Tym samym, nie jest możliwe w pełni obiektywne i jednoznaczne porównanie takich wyników.

1.1. Funkcje realizowane przez neuron

Za przetwarzanie sygnału przez każdy neuron odpowiedzialne są dwie funkcje — funkcja aktywacji i funkcja wyjścia. *Funkcja aktywacji* oblicza wartość całkowitego sygnału wejściowego neuronu. W tym podrozdziale będzie to liniowa kombinacja sygnałów wejściowych, choć w podrozdziale 1.2.1 zostaną przedstawione bardzo różne funkcje odległości, które będą mogły zastąpić ową liniową kombinację.

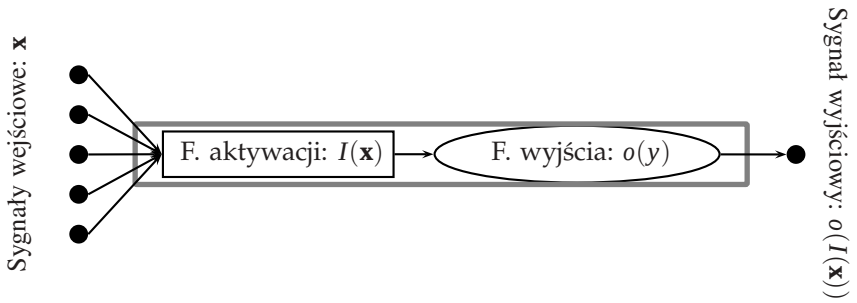
Jeśli neuron i jest połączony z neuronem j (gdzie $j = 1, \ldots, N$) i wysyła sygnał o wartości x_j z *siłą* połączenia równą W_{ij}, to całkowita aktywacja I_i będzie równa:

$$I_i(\mathbf{x}; \mathbf{W}) = \sum_{j=1}^{N} W_{ij} x_j. \tag{1.1}$$

Powyższa liniowa kombinacja wejść jest najczęściej stosowaną funkcją aktywacji używaną w sieciach MLP.

Drugą funkcją przetwarzaną przez neuron jest *funkcja wyjścia* $o(I)$. Te dwie funkcje razem decydują o wartości sygnału na wyjściu neuronu. Całość przetwarzania informacji przez neuron odbywa się w N wymiarowej przestrzeni wejściowej, która jest także nazywana przestrzenią parametrów. Złożenie funkcji aktywacji z funkcją wyjścia nazywa się *funkcją transferu* $o(I(\mathbf{x}))$. Porównaj rysunek 1.1.

Funkcje aktywacji i wyjścia dla warstwy wejściowej i wyjściowej mogą być inne niż dla warstw ukrytych. Zazwyczaj stosowane są funkcje liniowe w warstwie wejściowej i wyjściowej, a dla warstw ukrytych wybiera się nieliniowe funkcje transferu. Pewne funkcje transferu nie mogą być w naturalny sposób podzielone na funkcję aktywacji i funkcje wyjścia. Za lokalną funkcję transferu będzie się przyjmować funkcję, której wartości będą istotnie różne od zera (tj. $|o(I(\mathbf{x}))| > \epsilon$ dla pewnego ϵ) dla wartości $\mathbf{x}$ leżących na skończonym obszarze przestrzeni wejściowej. To oznacza, że lokalny charakter funkcji transferu będzie zależał nie tylko od funkcji wyjścia, ale również od funkcji aktywacji.



Rysunek 1.1: Model neuronu. Sygnał wejściowy i wyjściowy. Funkcja aktywacji i wyjścia. Funkcja transferu jako złożenie funkcji aktywacji i wyjścia.

Pierwsze modele sieci neuronowych zaproponowane w pracy McCulloch'a i Pitts'a [180] wykorzystywały w przetwarzaniu funkcje logiczne. Funkcja wyjścia w takim modelu była funkcją schodkową (progową) $\Theta(I;\theta)$, która przyjmowała wartość 0 poniżej progu θ i 1 powyżej progu:

$$\Theta(I;\theta) = \begin{cases} 1 & I > \theta, \\ 0 & I \leq \theta. \end{cases} \tag{1.2}$$

Używanie funkcji progowych było motywowane analizą logicznego działania podukładów komputerów, jak i wyobrażaniem sposobu pracy mózgu, jako podobnego do sposobu przetwarzania informacji w strukturach składających się z elementów przełącznikowych (logicznych).

W zasadzie można dokonywać dowolnych obliczeń przy użyciu neuronów logicznych (tj. używających funkcji logicznych). Trzeba wtedy rzeczywiste wartości dyskretyzować i użyć neuronów logicznych do uczenia ich reprezentacji bitowej. Ogromną zaletą korzystania z logicznych elementów jest możliwość szybkiego przetwarzania takiej informacji, jak również możliwość efektywnej realizacji takich funkcji sprzętowo. Granice decyzji, otrzymane w wyniku użycia neuronów logicznych są hiperpłaszczyznami zdefiniowanymi przez parametry W_{ij}. Wtedy sieć oparta o takie elementy dzieli przestrzeń wejściową na hiperwielościany lub pewne nieskończone fragmenty przestrzeni.

Funkcje wieloschodkowe stanowią etap pośredni pomiędzy funkcjami schodkowymi, a funkcjami semi-liniowymi. Liczba progów funkcji wieloschodkowej jest określona, a samą funkcję można zdefiniować poprzez:

$$\varsigma_m(I) = y_i \quad \text{dla} \quad \theta_i \leq I < \theta_{i+1}. \tag{1.3}$$

Aby uniknąć konstrukcji warunkowych dla stałych różnic $\theta = \theta_i - \theta_{i+1}$ wieloschodkowe funkcje można implementować efektywnie przy użyciu wektorów schodków $\mathbf{v}$ i arytmetyki stałopozycyjnej do konwersji przeskalowanych wartości wejściowych do danej przestrzeni wyjściowej: $\mathbf{v}\,[\Theta\,(1 + Int[(I - \theta_1)/\theta])]$,

gdzie θ_1 jest pierwszym progiem. Zamiast funkcji wieloschodkowej stosuje się funkcje semi-liniowa:

$$s_l(I;\theta_1,\theta_2) = \begin{cases} 0 & I \le \theta_1, \\ (I-\theta_1)/(\theta_2-\theta_1) & \theta_1 < I \le \theta_2, \\ 1 & I > \theta_2. \end{cases} \tag{1.4}$$

Te funkcje zostały później uogólnione do funkcji *logistycznej*, powszechnie spotykanej w literaturze (patrz rys. 1.2):

$$\sigma(I) = \frac{1}{1+e^{-sI}}. \tag{1.5}$$

Stała s określa skos funkcji logistycznej wokół jej liniowej części. Skos funkcji logistycznej wokół jej liniowej części zależny jest także od normy wektora wag $\mathbf{w}$. Istnieje cała grupa różnych funkcji o kształcie podobnym do funkcji logistycznej nazwana *funkcjami sigmoidalnymi*. W granicy, gdy $||\mathbf{w}||$ dąży do nieskończoności wszystkie funkcje sigmoidalne przechodzą w funkcję schodkową.

Złożenie liniowej aktywacji (1.1) z funkcją logistyczną, daje najbardziej popularną spośród funkcji transferu sieci neuronowych. Złożenia funkcji sigmoidalnych z liniową aktywacją dają w rezultacie funkcję nielokalną, choć nic nie stoi na przeszkodzie aby sigmoidalnych funkcji wyjściowych użyć w złożeniu z innymi lokalnymi funkcjami aktywacji (por. równania (1.66–1.69)), tworząc w ten sposób lokalną funkcję transferu.

Ciągle panuje powszechne przekonanie, że aktywność neuronów biologicznych ma wiele wspólnego z funkcjami sigmoidalnymi, choć nie jest to powód, dla którego funkcje sigmoidalne są tak popularne. Z wyjątkiem paru neurobiologicznych inspiracji, funkcje sigmoidalne mogą mieć uzasadnienie statystyczne [13, 143].

Rozważmy problem klasyfikacji w N wymiarowej przestrzeni z dwiema klasami o normalnym rozkładzie z równymi macierzami kowariancji

$$p(\mathbf{x}|C_k) = \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\bar{\mathbf{x}}_k)^T\Sigma^{-1}(\mathbf{x}-\bar{\mathbf{x}}_k)\right\}. \tag{1.6}$$

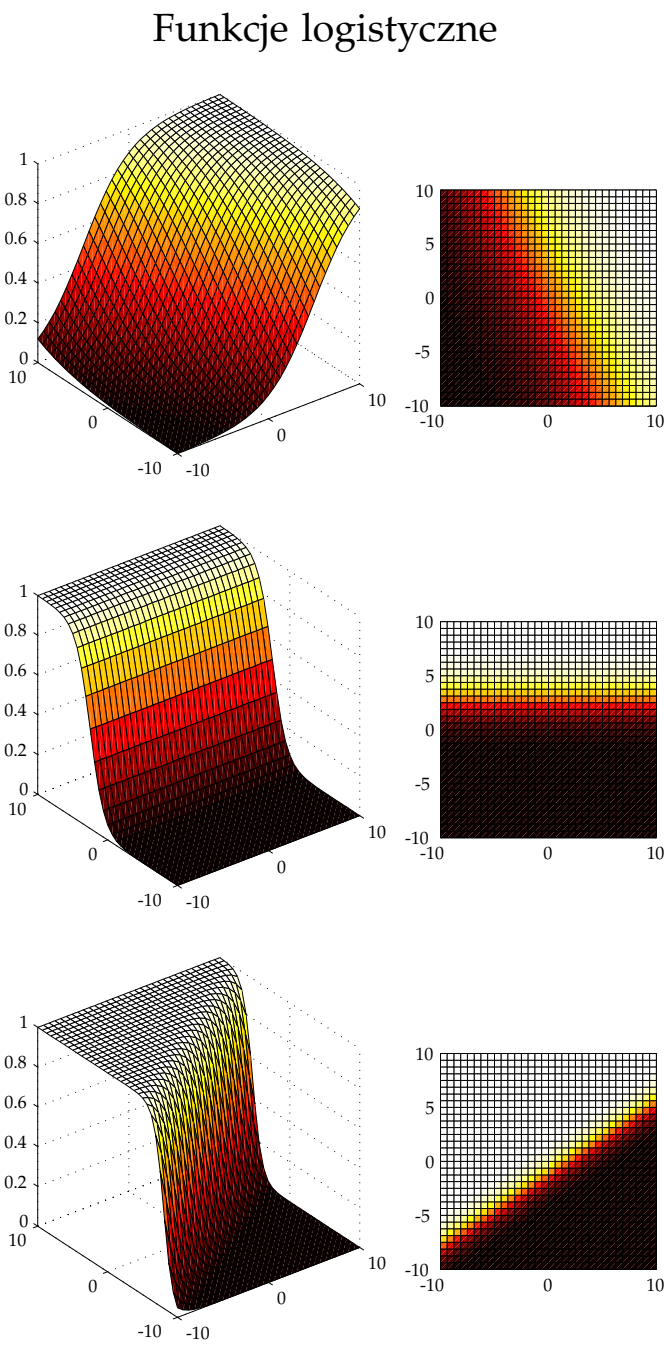
Korzystając z twierdzenia Bayesa prawdopodobieństwo *a posteriori* dla pierwszej klasy jest określone przez:

$$p(C_1|\mathbf{x}) = \frac{p(\mathbf{x}|C_1)p(C_1)}{p(\mathbf{x}|C_1)p(C_1)+p(\mathbf{x}|C_2)p(C_2)} = \frac{1}{1+\exp(-y(\mathbf{x}))}, \tag{1.7}$$

gdzie $p(C_k)$ jest prawdopodobieństwem klas *a priori*, a funkcja $y(\mathbf{x})$ jest zdefiniowana przez:

$$y(\mathbf{x}) = \ln\frac{p(\mathbf{x}|C_1)p(C_1)}{p(\mathbf{x}|C_2)p(C_2)}. \tag{1.8}$$

Mamy równość: $p(C_2|\mathbf{x}) = 1 - p(C_1|\mathbf{x})$. Prowadzi to do logistycznej funkcji wyjścia z dość skomplikowaną funkcją aktywacji. Takie funkcje są używane w



Rysunek 1.2: Funkcje logistyczne w dwóch wymiarach.

logistycznej analizie dyskryminacyjnej [5]. Dla problemów więcej niż dwuklasowych można użyć znormalizowanej funkcji eksponencjalnej (czasem zwanej *softmax*):

$$p(C_k|\mathbf{x}) = \frac{\exp(y_k(\mathbf{x}))}{\sum_i \exp(y_i(\mathbf{x}))}. \tag{1.9}$$

Po takiej normalizacji wartości $p(C_k|\mathbf{x})$ mogą być interpretowane jako prawdopodobieństwa.

Innym uzasadnieniem racjonalności funkcji sigmoidanych [61] może być fakt, iż wartości wejściowe pochodzą zazwyczaj z obserwacji, które nie są całkiem dokładne, dlatego można zamiast wartości $\bar{y}$ użyć wartość rozkładu Gaussa $G_y = G(y; \bar{y}, s_y)$ wokół $\bar{y}$ z odchyleniem standardowym s_y. Rozkład ten można też traktować jako funkcje przynależności rozmytej liczby G_y [161]. Dystrybuanta wygląda natomiast tak:

$$p(x - \bar{y}) = \int_{-\infty}^{x} G(y; \bar{y}, s_y)dy = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x - \bar{y}}{s_y\sqrt{2}}\right)\right] \approx \sigma\left(\frac{x - \bar{y}}{T}\right), \tag{1.10}$$

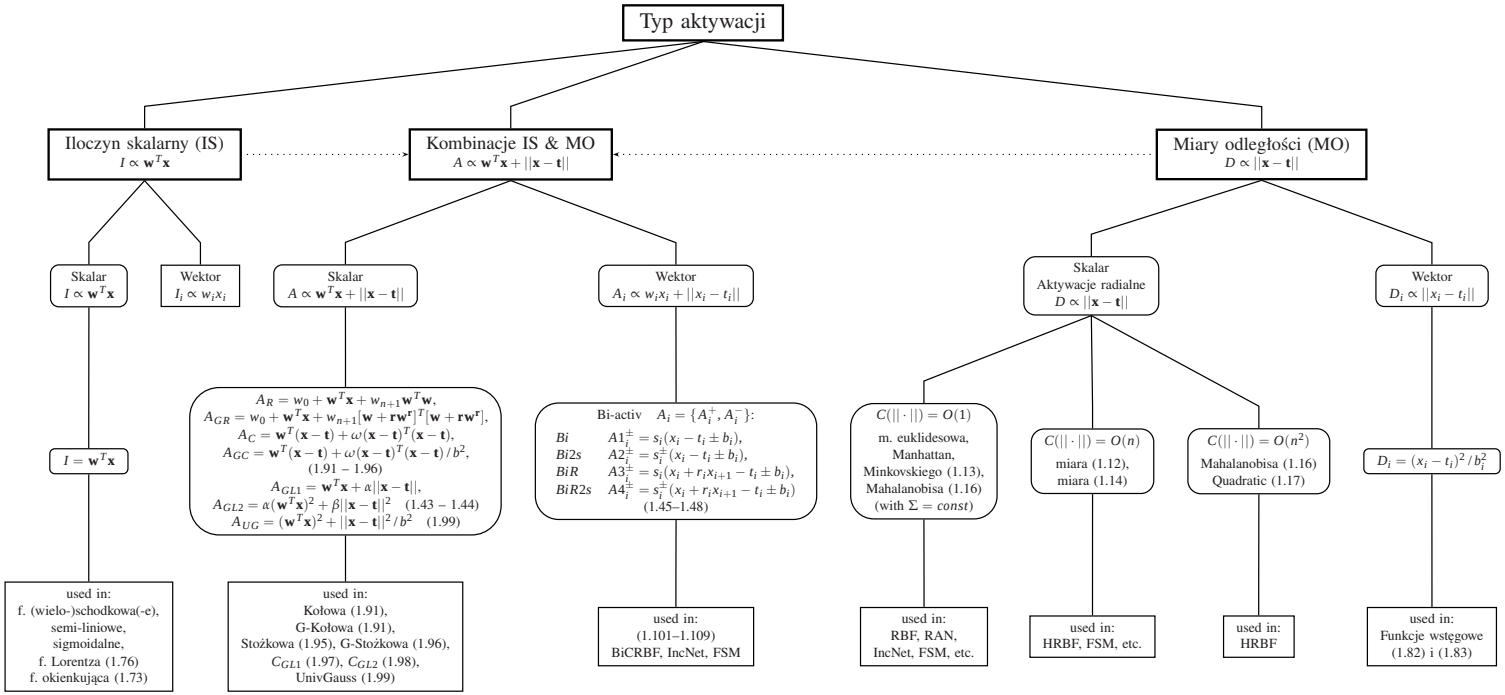
gdzie erf jest funkcją błędu, a $T = \sqrt{2}s_y / 2.4$. Dokładność tej aproksymacji jest nie gorsza niż 0.02 w każdym punkcie $\mathbf{x}$. Skumulowany rozkład $p(x - \bar{y})$ może być interpretowany jako prawdopodobieństwo zajścia reguły $R_x(z)$ wtedy i tylko wtedy gdy $z \leq x$ jest prawdą, tj. $p(R_x|G_y) = p(x - \bar{y})$.

W następnym podrozdziale przedstawione zostaną różne typy funkcji aktywacji.

1.2. Funkcje aktywacji

Liniowa kombinacja wejść, w literaturze angielskiej zwana *fan-in activation* (1.1), jako aktywacja jest stosowana nie z powodów inspiracji biologicznych, lecz dlatego, że kontury o stałej wartości $I(\mathbf{x}) = const$ formują hiperpłaszczyznę. Metody statystyczne klasyfikacji mogą być podzielone na dwie grupy. Pierwszą grupę stanowią metody bazujące na analizie dyskryminacyjnej, które używają hiperpłaszczyzn lub innych powierzchni do podziału przestrzeni wejściowej. Druga grupa obejmuje metody klasteryzacji i metody oparte na podobieństwie, które korzystają z pewnych miar odległości lub funkcji podobieństwa. Stąd też mamy do czynienia z dwoma różnymi typami funkcji aktywacji i ich kombinacją:

- **Kombinacja liniowa** (iloczyn skalarny) $I(\mathbf{x}; \mathbf{w}) \propto \mathbf{w}^T \cdot \mathbf{x}$ (używana na przykład w sieciach perceptronowych).
- **Miary odległości** jako aktywacje, lub ogólniej miary podobieństwa, $D(\mathbf{x}; \mathbf{t}) \propto ||\mathbf{x} - \mathbf{t}||$, wyznaczają podobieństwo wektora $\mathbf{x}$ do wektora $\mathbf{t}$.
- **Kombinacje** dwóch powyższych aktywacji, $A(\mathbf{x}; \mathbf{w}, \mathbf{t}) \propto \alpha\ \mathbf{w}^T \cdot \mathbf{x} + \beta\ ||\mathbf{x} - \mathbf{t}||$,



Rysunek 1.3: Taksonomia funkcji aktywacji. $C(||\cdot||)$ jest liczbą parametrów wolnych normy $||\cdot||$.

Taksonomia przeróżnych funkcji aktywacji została zaprezentowana na rysunku 1.3.

W każdym przypadku końcowa aktywacja jest wielkością skalarną lub wektorową. Na przykład typowa funkcja odległości $D(\mathbf{x}, \mathbf{t})$ daje jako wynik skalar, choć jej składowe być używane jako wektor $D_i(x_i, t_i)$, gdzie $D_i(x_i, t_i)$ może być zdefiniowane jako:

$$D_i^2(x_i, t_i, b_i) = (x_i - t_i)^2 / b_i^2. \tag{1.11}$$

Kwadrat powyższej funkcji aktywacji jest formą kwadratową. Uznając wszystkie parametry takiej formy za niezależne i przekształcając do formy kanonicznej, mamy:

$$I^2(\mathbf{x}; \mathbf{w}) \sim D^2(\mathbf{x}; \mathbf{t}, \mathbf{a}) = \sum_i^N a_i (x_i' - t_i)^2, \tag{1.12}$$

gdzie zmienne x_i' są liniowymi kombinacjami oryginalnych zmiennych x_i i odpowiadają pseudo-euklidesowej mierze odległości. Jeśli parametry a_i są dodatnie i przyjmiemy $a_i = 1/b_i^2$, to otrzymuje się miarę euklidesową z hiperelipsoidalnymi konturami dla stałych wartości miary. Kwadrat liniowej kombinacji wejść był użyty do *Lorentzowskiej* funkcji transferu (1.76, rys. 1.17). Bardzo podobny efekt uzyskuje się używając aktywacji iloczynu skalarnego z gaussowską funkcją wyjścia (1.73, rys. 1.14), tworząc w ten sposób funkcję okienkującą. Lorentzowska funkcja nie ma elipsoidalnych konturów, powierzchnie są nielokalne, natomiast lokalne ze względu na przekroje prostopadłe do hiperpłaszczyzny zdefiniowanej przez I. Kontury tworzą *okienka* aktywacji (tj. wycinają obszar *okienkowy*, w którym wartości funkcji Lorentza są większe od pewnego α).

1.2.1. Miary odległości i podobieństwa jako funkcje aktywacji.

Drugą grupę funkcji aktywacji stanowią aktywacje oparte o podobieństwo wejściowych wektorów do pewnych wektorów prototypowych lub ich uogólnień.

Niektóre miary mogą być wręcz równoważne pewnym przekształceniom samych danych wejściowych jeszcze przed procesem uczenia, tym samym miary takie pełnią raczej dość statyczną rolę w procesie uczenia.

Warto tu zaznaczyć, że głównym celem przekształcenia danych wejściowych powinno być dokonanie takiej transformacji danych, aby model adaptacyjny mógł z nich wyekstrahować jak najwięcej informacji i uzyskać możliwie maksymalną generalizację. Z kolei inne miary nie mogą być zastąpione poprzez transformacje danych przed uczeniem, wtedy też ich charakter podczas procesu uczenia może być dynamiczny poprzez możliwość adaptacji parametrów takiej miary.

Mary mogą być jednorodne i niejednorodne. Miara odległości jest jednorodna, gdy wszystkie cechy przestrzeni wejściowej traktuje tak samo. Miary niejednorodne mogą stosować zupełnie inne sposoby oceny wartości w poszczególnych cechach.

1.2.1.1. Jednorodne miary odległości.

Jako miary podobieństwa może być używana nie tylko miara euklidesowa, często wykorzystywana w sieciach z radialnymi funkcjami bazowymi (*ang. radial basis function*), ale również jej naturalne uogólnienie do poniższej miary Minkowskiego (jak i inne miary przedstawione w dalszej części):

$$D_M(\mathbf{x}, \mathbf{y}; \alpha) = \left(\sum_{i=1}^{N} |x_i - y_i|^{\alpha} \right)^{1/\alpha}. \tag{1.13}$$

Miara euklidesowa i Manhattan są oczywiście specjalnymi przypadkami miary Minkowskiego dla $\alpha = 2$ i $\alpha = 1$. Można jeszcze bardziej rozbudować miarę Minkowskiego, wprowadzając czynniki skalujące:

$$D_{Mb}(\mathbf{x}, \mathbf{y}; \mathbf{b})^{\alpha} = \sum_{i}^{N} d(x_i, y_i)^{\alpha} / b_i. \tag{1.14}$$

Funkcja $d(\cdot)$ jest używana do estymacji podobieństwa dla danego wymiaru, najczęściej stosuje się po prostu: $d(x_i, y_i) = |x_i - y_i|$. Dla $\alpha = 2$ wektor $||\mathbf{x}|| = 1$ znajduje się na sferze jednostkowej, dla większych wartości α sfera przechodzi w gładki hipersześcian, a dla $\alpha < 1$ przyjmuje kształt hipocykloidy (patrz rys. 1.4).

Podobna funkcja była użyta jako jądro (*ang. kernel function*) w modelu Generalized Memory-Based Learning [45]:

$$C_K(\mathbf{x}, \mathbf{y}, \mathbf{b}) = \left(\sum_{k=1}^{d} (x_k - y_k)^2 / b_k^2 \right)^{-q}, \tag{1.15}$$

gdzie $q > 0$.

Inną, miarą jest miara Mahalanobisa:

$$D_M^2(\mathbf{x}; \mathbf{y}) = \sum_{ij} (x_i - y_i) \Sigma^{-1} (x_j - y_j) \tag{1.16}$$

gdy Σ jest macierzą kowariancji, to miara wyznacza odległości we współrzędnych głównych.

W mierze można użyć bardziej ogólnej formy kwadratowej z dodatnio określoną macierzą Q ustaloną dla danego problemu:

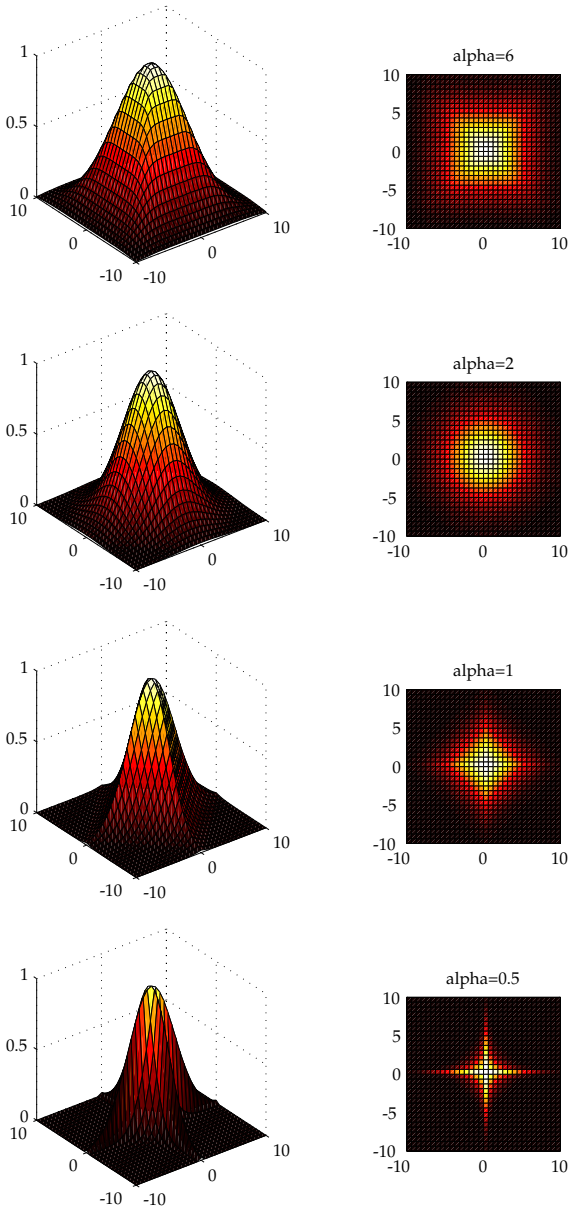
$$D_Q(\mathbf{x}, \mathbf{y}; Q) = (\mathbf{x} - \mathbf{y})^{\mathrm{T}} Q (\mathbf{x} - \mathbf{y}). \tag{1.17}$$

Z kolei miara Czebyszewa określona jest przez:

$$D_{Ch}(\mathbf{x}, \mathbf{y}) = \max_{i=1,\ldots,N} |x_i - y_i|, \tag{1.18}$$

Różnego rodzaju czynniki korelacyjne są również pożądane. Na przykład funkcja Canberra:

$$D_{Ca}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} \frac{|x_i - y_i|}{|x_i + y_i|} \tag{1.19}$$



Rysunek 1.4: Funkcja Gaussa (1.65) z miarą Minkowskiego o różnych współczynnikach równania 1.13.

czy też odległość χ^2:

$$D_\chi(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{N} \frac{1}{\mathbf{sum}_i}\left(\frac{x_i}{\mathbf{size}_x} - \frac{y_i}{\mathbf{size}_y}\right)^2, \tag{1.20}$$

gdzie $\mathbf{sum}_i$ jest sumą wszystkich wartości cechy i ze zbioru trenującego, a $\mathbf{size}_x$ i $\mathbf{size}_y$ są sumami wszystkich wartości wektorów x i y. Należy zapewnić aby wartości $\mathbf{sum}_i$, $\mathbf{size}_x$ i $\mathbf{size}_y$ były różne od zera.

Korelacyjna funkcja podobieństwa (nie będąca miarą) jest zdefiniowana poprzez:

$$D_{Cd}(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{N}(x_i-\bar{x}_i)(y_i-\bar{y}_i)}{\sqrt{\sum_{i=1}^{N}(x_i-\bar{x}_i)^2\sum_{i=1}^{N}(y_i-\bar{y}_i)^2}}, \tag{1.21}$$

gdzie $\bar{x}_i$ i $\bar{y}_i$ są wartościami średnimi cechy i ze zbioru treningowego.

Z kolei funkcję korelacyjną rangową Kendalla definiuje poniższe wyrażenie:

$$D_{KRC}(\mathbf{x},\mathbf{y}) = 1 - \frac{2}{n(n-1)}\sum_{i=1}^{N}\sum_{j=1}^{i-1}\mathrm{sign}(x_i-x_j)\,\mathrm{sign}(y_i-y_j). \tag{1.22}$$

$\mathrm{sign}(x)$ jest równy 1 dla $x>0$, 0 dla $x=0$ i -1 dla $x<0$.

Wszystkie z powyższych funkcji mogą zastąpić odległość Euklidesową w radialnych funkcjach transferu (1.3.2).

1.2.1.2. Niejednorodne miary odległości.

Mary niejednorodne umożliwiają różne traktowanie poszczególnych cech bądź grup cech. To jest szczególnie ważne, gdy mamy do czynienia z wartościami, które nie mają porządku (wartości nominalne (symboliczne)). Należy zwrócić uwagę, że wartości dyskretne mogą, ale nie muszą posiadać porządek.

Dlatego na przykład dla cech numerycznych można użyć miary Minkowskiego, a dla cech symbolicznych miar statystycznych. W metodach rozumowania opartych na precedensach (*ang. memory-based reasoning*) popularność zyskała miara VDM (*ang. Value Difference Metric*) [260, 261, 258]. Odległość VDM pomiędzy dwoma N wymiarowymi wektorami $\mathbf{x}$, $\mathbf{y}$ z cechami o wartościach dyskretnych (w tym cechami symbolicznymi) w C-klasowym problemie jest definiowana poprzez prawdopodobieństwa warunkowe jako:

$$D^q_{VDM}(\mathbf{x},\mathbf{y}) = \sum_{j=1}^{N}\sum_{i=1}^{C}\left|p(C_i|x_j) - p(C_i|y_j)\right|^q, \tag{1.23}$$

gdzie $p(C_i|x_j)$ jest estymowane przez liczbę $N_i(x_j)$ wystąpień wartości x_j cechy j w wektorach należących do klasy C_i podzielonej przez liczbę $N(x_j)$ wystąpień wartości x_j cechy j w wektorach należących do dowolnej klasy:

$$D^q_{VDM}(x,y) = \sum_{j=1}^{N}\sum_{i=1}^{C}\left|\frac{N_i(x_j)}{N(x_j)} - \frac{N_i(y_j)}{N(y_j)}\right|^q. \tag{1.24}$$

Różnica wartości dla j-tej cechy jest zdefiniowana jako:

$$d_{VDM}^{q}(x_j, y_j) = \sum_{i=1}^{C} |(p(C_i|x_j) - p(C_i|y_j))|^q \tag{1.25}$$

pozwala policzyć $D_{VDM}(\mathbf{x}, \mathbf{y})$ przez sumowanie różnic wartości po wszystkich wymiarach. Tak zdefiniowana miara odległości jest zależna od danych (poprzez macierz z liczbą wierszy równą liczbie klas, liczbie kolumn równej liczbie cech). Uogólnienie tej miary na wartości ciągłe, wymaga zbioru funkcji gęstości $p_{ij}(x)$ z $i = 1, \ldots, C$ i $j = 1, \ldots, N$.

Niejednorodna miara HEOM (*ang. Heterogeneous Euclidean-Overlap Metric*) jest pewnym uproszczeniem miary VDM:

$$D_{HEOM}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{j=1}^{N} d_j(x_j, y_j)^2}, \tag{1.26}$$

gdzie odległość d_j wyznaczana jest przez:

$$d_j(x_j, y_j) = \begin{cases} 1 & \text{gdy } x_j \text{ lub } y_j \text{ jest nieznany, nieustalony,} \\ overlap(x_j, y_j) & \text{gdy atrybut } x_j \text{ jest nominalny,} \\ \frac{|x_j - y_j|}{x_j^{max} - x_j^{min}} & \text{wp.p.,} \end{cases} \tag{1.27}$$

x_j^{max} i x_j^{min} jest odpowiednio maksymalną i minimalną wartością j-tego atrybutu:

$$x_j^{max} = \max_i x_j^i \qquad x_j^{min} = \min_i x_j^i. \tag{1.28}$$

Różnica pomiędzy x_j^{max} i x_j^{min} określa zakres j-tego atrybutu. Funkcja *overlap* jest zdefiniowana poprzez:

$$overlap(x, y) \quad = \quad \begin{cases} 0 & x = y, \\ 1 & x \neq y. \end{cases} \tag{1.29}$$

Niejednorodną miarę HVDM (*ang. Heterogeneous Value Difference Metric*) można zdefiniować poprzez:

$$D_{HVDM}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{j=1}^{N} \left(dh_j(x_j, y_j)\right)^2}, \tag{1.30}$$

gdzie $dh_j(x_j, y_j)$ jest określone przez

$$dh_j(x_j, y_j) = \begin{cases} 1 & x \text{ lub } y \text{ jest nieznany,} \\ N\_vdm_j(x_j, y_j) & \text{cecha } j \text{ jest nominalna,} \\ N\_dif_j(x_j, y_j) & \text{cecha } j \text{ jest liniowa,} \end{cases} \tag{1.31}$$

a

$$N\_dif_j(x_j, y_j) = \frac{|x_j - y_j|}{4\sigma_j}, \tag{1.32}$$

gdzie σ_j oznacza odchylenie standardowe wartości cechy j. Znormalizowaną odległość VDM można wyznaczyć na kilka sposobów:

$$N1\_vdm(x,y) = \sum_{i=1}^{C}\left|\frac{N_i(x)}{N(x)} - \frac{N_i(y)}{N(y)}\right|, \tag{1.33}$$

$$N2\_vdm(x,y) = \sqrt{\sum_{i=1}^{C}\left(\frac{N_i(x)}{N(x)} - \frac{N_i(y)}{N(y)}\right)^2}, \tag{1.34}$$

$$N3\_vdm(x,y) = \sqrt{C}\ \ N2\_vdm(x,y). \tag{1.35}$$

Dyskretna odmiana miary VDM (*ang. Discrete Value Difference Metric*) może być używana dla ciągłych wartości atrybutów:

$$d_{DVDM}(\mathbf{x},\mathbf{y}) = \sum_{j=1}^{N} vdm_j\big(disc_j(x_i), disc_j(y_j)\big)^2, \tag{1.36}$$

gdzie *disc* jest funkcją dyskretyzacji:

$$disc_j(x_j) = \begin{cases} \left\lfloor \frac{x - min_j}{w_j} \right\rfloor + 1 & \text{cecha } j \text{ jest ciągła} \\ x & \text{cecha } j \text{ jest dyskretna} \end{cases}, \tag{1.37}$$

w_j są parametrami. Dyskretyzacja umożliwia użycie miary VDM zarówno do nominalnych wartości, jak i do ciągłych. Jeszcze innym sposobem obliczania miary VDM dla cech o ciągłych wartościach jest użycie interpolowanej miary VDM (*ang. Interpolated Value Difference Metric*):

$$d_{IVDM}(\mathbf{x},\mathbf{y}) = \sum_{j=1}^{N} ivdm_j\big(x_j, y_j\big)^2, \tag{1.38}$$

gdzie

$$ivdm_j(x_j, y_j) = \begin{cases} vdm_j(x_j, y_j) & \text{cecha } j \text{ jest dyskretna} \\ \sum_{i=1}^{C}\big(p(C_i|x_j) - p(C_i|y_j)\big)^2 & \text{cecha } j \text{ jest ciągła} \end{cases}. \tag{1.39}$$

Wyżej użyte prawdopodobieństwa są wyznaczane poprzez interpolację:

$$p(C_i|x_j) = p(C_i|x_j, u) + \frac{x_j - x_{j,u}^{mid}}{x_{j,u+1}^{mid} - x_{j,u}^{mid}}\big(p(C_i|x_j, u+1) - p(C_i|x_j, u)\big), \tag{1.40}$$

gdzie $x_{j,u}^{mid}$ i $x_{j,u+1}^{mid}$ są środkami dwóch następujących zdyskretyzowanych podziałów, spełniających nierówność $x_{j,u}^{mid} \leq x_j \leq x_{j,u+1}^{mid}$. $p(C_i|x_j, u)$ jest prawdopodobieństwem zdyskretyzowanego podziału u, zdefiniowanego w jego środku. Wartości podziałów u są wyznaczane przez funkcje $disc_j$: $u = disc_j(x_j)$.

Miary typu VDM mogą być stosowane w problemach, w których korzysta się z metod gradientowych. W pełni numeryczne wektory wejściowe uzyskuje się, używając ciągłych wymiarów, które zastępują wartości symboliczne i dyskretne poprzez prawdopodobieństwa $p(C_i|x_j)$.

Jak widać możliwości doboru funkcji odległości są całkiem bogate, choć w praktyce rzadko się spotyka użycie odległości innej niż Euklidesowa. Również i sposób, w jaki oddziaływują funkcje odległości z daną metodą uczenia może być dalece inny.

1.2.2. Funkcje aktywacji powstające jako złożenie iloczynu skalarnego i miar podobieństwa

Aby zwiększyć możliwości typowej funkcji sigmoidalnej można skorzystać z bardziej wyrafinowanych funkcji aktywacji. Dobrym przykładem takiej funkcji aktywacji może być funkcja zaproponowana przez Ridellę *i. in.* [215]:

$$A_R(\mathbf{x}; \mathbf{w}) = w_0 + \sum_{i=1}^{N} w_i x_i + w_{N+1} \sum_{i=1}^{N} x_i^2. \tag{1.41}$$

Inną bardzo ciekawą kombinację zaproponował Dorffner [56], do stworzenia stożkowych funkcji transferu:

$$A_C(\mathbf{x}; \mathbf{w}, \mathbf{t}, \omega) = I(\mathbf{x} - \mathbf{t}; \mathbf{w}) + \omega D(\mathbf{x} - \mathbf{t}). \tag{1.42}$$

Wektor $\mathbf{t}$ określa centrum aktywacji dla drugiej części prawej strony równania, ω jest wolnym parametrem.

Funkcje transferu C_{GL1} i C_{GL2} opisane wzorami (1.97 i 1.98) używają jeszcze innych kombinacji komponując równie ciekawe funkcje aktywacji:

$$A_{GL1} = \mathbf{w}^T\mathbf{x} + \alpha||\mathbf{x} - \mathbf{t}||, \tag{1.43}$$
$$A_{GL2} = \alpha(\mathbf{w}^T\mathbf{x})^2 + \beta||\mathbf{x} - \mathbf{t}||^2, \tag{1.44}$$

α i β są parametrami wolnymi.

Wynikiem powyższych funkcji aktywacji jest wartość skalarna. Bicentralne funkcje transferu (dokładnie opisane w podrozdziale 1.4.6) korzystają z wektora aktywacji. Co więcej, bicentralne funkcje transferu korzystają dokładnie z dwóch wektorów aktywacji, lewej i prawej: $A_i = \{A_i^+, A_i^-\}$ i $\mathbf{A} = [A_1, \ldots, A_n]$. Poniżej

prezentowane są różne funkcje aktywacji dla różnych funkcji bicentralnych:

$$\text{Bi} \qquad A1_i^{\pm} = s_i(x_i - t_i \pm b_i), \tag{1.45}$$

$$\text{Bi2s} \qquad A2_i^{\pm} = s_i^{\pm}(x_i - t_i \pm b_i), \tag{1.46}$$

$$\text{BiR} \qquad A3_i^{\pm} = s_i(x_i + r_i x_{i+1} - t_i \pm b_i), \tag{1.47}$$

$$\text{BiR2s} \qquad A4_i^{\pm} = s_i^{\pm}(x_i + r_i x_{i+1} - t_i \pm b_i), \tag{1.48}$$

wektor $\mathbf{t} - \mathbf{b}$ i $\mathbf{t} + \mathbf{b}$ określają centra aktywacji miary, a wektor $\mathbf{r}$ określa obrót.

Wtedy ogólnie funkcje bicentralną można zdefiniować poprzez:

$$BiTF(\mathbf{x}) = \prod_{i=1}^{N}[\sigma(Ak_i^+) \circ \sigma(Ak_i^-)]. \tag{1.49}$$

Przydatność takich funkcji aktywacji okaże się oczywista przy analizie podrozdziału 1.4.6).

1.3. Funkcje wyjścia

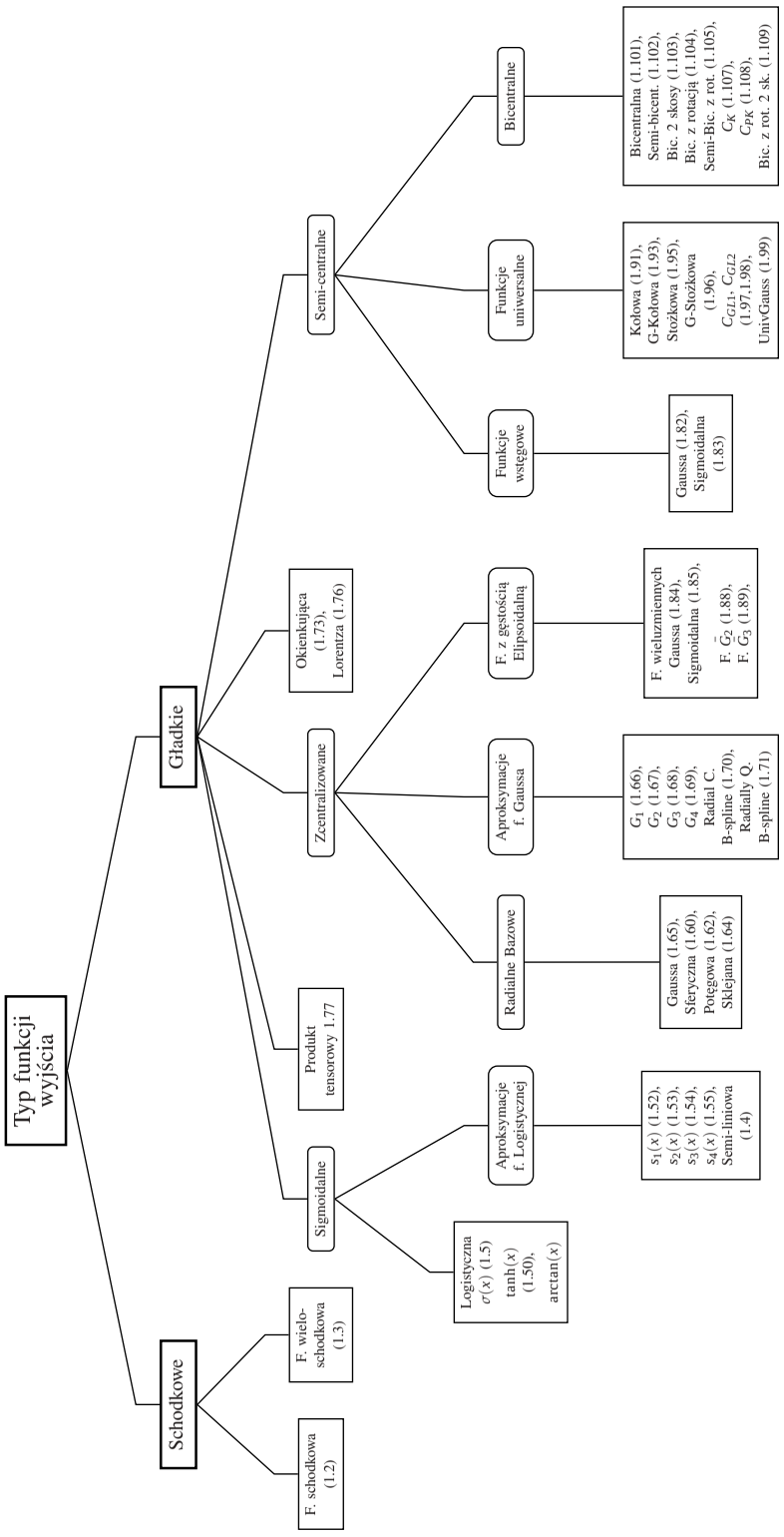
Najprostszym przykładem funkcji wyjścia jest oczywiście funkcja tożsamościowa. Pomimo swej prostoty (a może raczej dzięki swej prostocie!) często jest używana w warstwie wejściowej jak i wyjściowej różnych sieci neuronowych. Poza tym używana jest również w sieciach liniowych i warstwie ukrytej sieci RBF, gdzie w połączeniu z miarą odległości tworzy sferyczną funkcję transferu $||\mathbf{x} - \mathbf{t}||$ (warstwa wejściowa i nierzadko wyjścia sieci RBF wykorzystują również funkcję tożsamościową).

Ponieważ zazwyczaj funkcje aktywacji nie są ograniczone, funkcje wyjścia używane są do ograniczania ostatecznych wartości sieci neuronowej. Na trzy główne typy funkcji wyjścia składają się:

- Funkcje sigmoidalne.
- Funkcje zlokalizowane wokół pewnego centrum.
- Semi-centralne funkcje, które mogą być oparte na kilku centrach, niekoniecznie lokalne.

Należy także zwrócić uwagę, iż typ lokalności (lub nielokalności) zależy również od tego jaka aktywa zostanie użyta do danej funkcji wyjścia. Jako przykład warto porównać dalej opisaną funkcję gaussowska (1.65) i funkcję okienkującą (1.73).

Na rysunku 1.5 przedstawiona została taksonomia funkcji wyjścia.



Rysunek 1.5: Taksonomia funkcji wyjścia.

1.3.1. Funkcje sigmoidalne

Sigmoidalne funkcje wyjścia, czyli funkcje o kształcie S są nie tylko naturalne ze statystycznego punktu widzenia, lecz również w bardzo naturalny sposób umożliwiają ograniczenie nieograniczonych wartości pochodzących z funkcji aktywacji. Funkcje sigmoidalne są funkcjami nielokalnymi — są niezerowe na nieskończonym obszarze.

Ważną własnością jest także gładkość funkcji sigmoidalnych (i łatwość jej regulowania), co jest ważne dla gradientowych metod uczenia. Dla funkcji logistycznej (1.5) pochodna jest łatwa do wyznaczenia i jest ciągła $\sigma(I)' = \sigma(I)(1-\sigma(I))$. Funkcja logistyczna może być zastąpiona funkcją błędu (erf), funkcją arcus tangens lub też funkcją tangensa hiperbolicznego:

$$\tanh(I/b) = \frac{1-e^{-I/b}}{1+e^{-I/b}}, \tag{1.50}$$

$$\tanh'(I/b) = \operatorname{sech}^2(I/b)/b = \frac{4}{b(e^{-I/b}+e^{+I/b})^2}. \tag{1.51}$$

Ponieważ obliczanie funkcji eksponencjalnej jest istotnie wolniejsze od obliczania podstawowych operacji arytmetycznych, możliwe jest wykorzystanie innych funkcji o kształcie sigmoidalnym tak, aby przyśpieszyć obliczenia:

$$s_1(I;b) = \Theta(I)\frac{I}{I+b} - \Theta(-I)\frac{I}{I-b} = I\frac{\operatorname{sign}(I)I-b}{I^2-b^2}, \tag{1.52}$$

$$s_2(I;b) = \frac{bI}{1+\sqrt{1+b^2I^2}} = \frac{bI}{1+q}, \tag{1.53}$$

$$s_3(I;b) = \frac{bI}{1+|bI|}, \tag{1.54}$$

$$s_4(I;b) = \frac{bI}{\sqrt{1+b^2I^2}} = \frac{bI}{q}, \tag{1.55}$$

gdzie $\Theta(I)$ jest funkcją schodkową, a $q = \sqrt{1+b^2I^2}$. Pochodne powyższych funkcji daje się łatwo wyznaczyć i szybko obliczyć:

$$s_1'(I;b) = \frac{b}{(I+b)^2}\Theta(I) + \frac{b}{(I-b)^2}\Theta(-I) = \frac{b}{(I+\operatorname{sign}(I)b)^2}, \tag{1.56}$$

$$s_2'(I;b) = \frac{b}{q(1+q)}, \tag{1.57}$$

$$s_3'(I;b) = -\operatorname{sign}(I)\frac{b^2I}{(1+|bI|)^2} + \frac{b}{1+|bI|}, \tag{1.58}$$

$$s_4'(I;b) = -\frac{b^3I^2}{(1+b^2I^2)^{3/2}} + \frac{b}{\sqrt{1+b^2I^2}}. \tag{1.59}$$

Kształty tych funkcji[1] można porównać na rysunku 1.6. Funkcja sigmoidalna i tangens hiperboliczny są trudne do rozróżnienia, natomiast arcus tangens

[1]Wszystkie te funkcje zostały przeskalowane liniowo tak aby ich wartości znajdowały się w

i funkcje s_1, s_2 osiągają nasycenie wolniej dla większych wartości aktywacji. Wszystkie funkcje wygładzają jednak podobnie i dlatego można zaproponować korzystanie z funkcji s_1 lub s_2 aby zaoszczędzić na czasie obliczeń – w praktyce oznacza to oszczędności 2-3 krotne. Innym sposobem zaoszczędzenia czasu na liczeniu funkcji eksponencjalnej jest skorzystanie z pomysłu zamieszczonego w pracy [230], w której funkcja eksponencjalna jest wyznaczana jeszcze szybciej (wykorzystuje się reprezentacje bitową), ale z ok. 2% błędem, co czasem nie jest istotne.

Sieci neuronowe oparte na funkcjach sigmoidalnych mają całkiem dobre właściwości matematyczne. Jedną z nich jest fakt, iż sieć korzystająca z takich funkcji, składająca się z jednej warstwy neuronów, jest uniwersalnym aproksymatorem [52, 122]. Nie jest to wielką niespodzianką, ponieważ wiele funkcji może być użytych jako funkcje transferu dla uniwersalnego aproksymatora.

Interesujące jest to, że udało się ocenić zbieżność estymacji przy użyciu funkcji sigmoidalnych. Dla sieci o jednej warstwie z n neuronami i przy pewnych (nie bardzo istotnych) założeniach o aproksymowanej funkcji, współczynnik zbieżności zmienia się z $O(n^{-\frac{1}{2}})$, tj. nie zależy to od rozmiaru przestrzeni wejściowej [7, 163, 164]. Dla funkcji wielomianowych współczynnik zależy od rozmiaru przestrzeni wejściowej d i wynosi $O(n^{-\frac{1}{2d}})$, co dla wielowymiarowych przestrzeni prowadzi do katastrofalnie wolnej zbieżności. Z tego też powodu należy być sceptycznym w wyborze wielomianowych funkcji ortogonalnych na funkcje transferu [212, 42]. Stosowanie niewielomianowych funkcji, w tym funkcji periodycznych czy funkcji lokalnych sprawia, że współczynnik zbieżności nie zależy od rozmiaru przestrzeni wejściowej problemu [123].

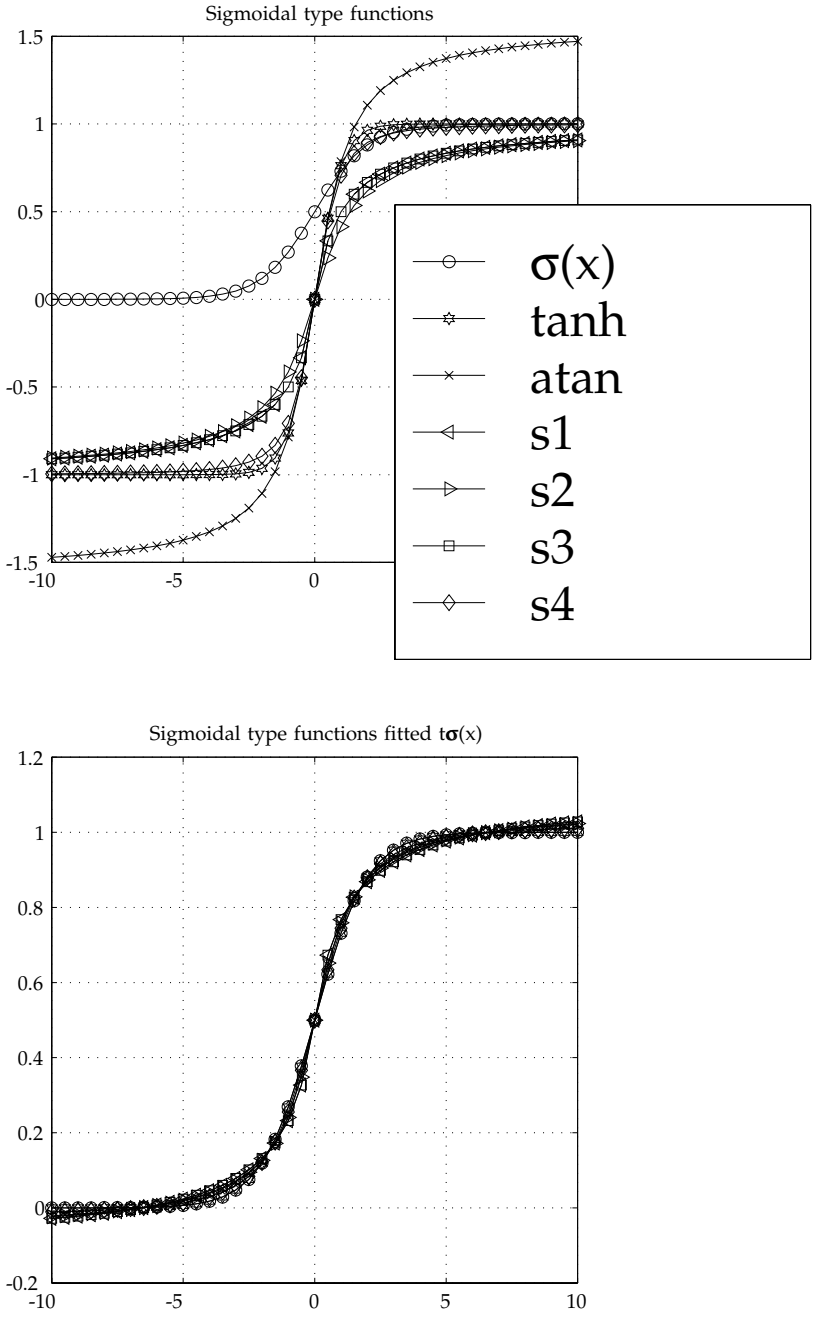
1.3.2. Funkcje zlokalizowane wokół jednego centrum

Inną silną klasę funkcji stanowią **radialne funkcje bazowe** (*ang. radial basis function*), wywodzące się głównie z teorii aproksymacji [211, 76, 84], a później także z metod rozpoznawania wzorców, choć występowały tam pod inna nazwą, jako funkcje potencjałowe [98]. Bardzo dobrym wprowadzeniem w świat radialnych funkcji bazowych, jak i sieci pod kątem teorii regularyzacji, jest praca napisana przez Poggio i Girosiego [209]. Dużo ciekawych informacji na temat funkcji RBF można znaleźć w [118, 27, 56, 172, 173, 174, 11, 203, 13].

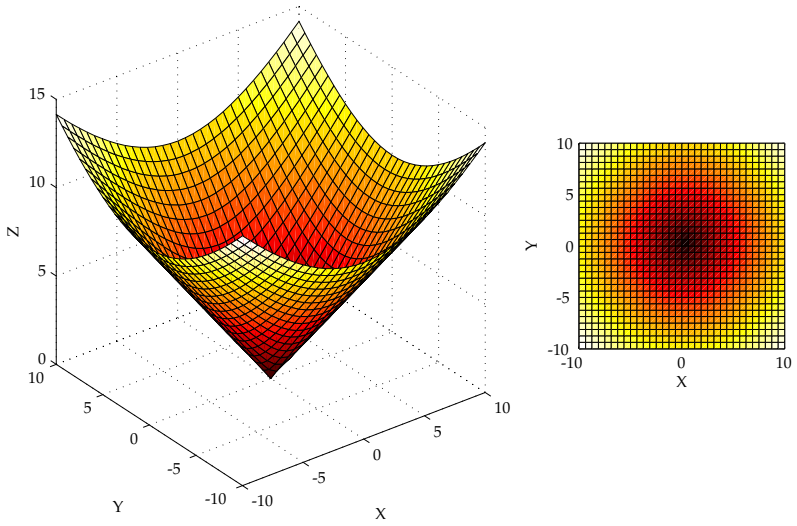
Radialne funkcje bazowe zawsze wykorzystują jako funkcje aktywacji odległość od centrum $\mathbf{t}$ funkcji bazowej $r = ||\mathbf{x} - \mathbf{t}|| / b^2$. W tym rozdziale opisane zostaną radialne funkcje wyjścia oparte o radialną aktywację: $o(r)$. Niektóre z radialnych funkcji wyjścia są nielokalne, a inne są lokalne. Najprostszą funkcję radialną uzyskuje się poprzez złożenie radialnej aktywacji z funkcją tożsamościową, jako funkcją wyjścia, tworząc w ten sposób funkcję sferyczną (patrz rys. 1.7). Funkcja ta nie jest lokalna i określona jest poprzez:

$$h(r) = r = ||\mathbf{x} - \mathbf{t}|| / b^2. \tag{1.60}$$

przedziale od -1 do 1, skosy poszczególnych funkcji zostały tak dobrane, aby uzyskać jak największe dopasowanie tych funkcji do funkcji sigmoidalnej.



Rysunek 1.6: Porównanie sigmoidalnych funkcji transferu.



Rysunek 1.7: Funkcja sferyczna (1.60).

Allison [2] proponuje użycie prostej funkcji potęgowej (*ang. simple multiquadratic*):

$$s_m(I;\Delta) = \sqrt{I^2+\Delta^2}; \qquad s'_m(I;\Delta) = \frac{I}{s_m(I;\Delta)}, \tag{1.61}$$

gdzie Δ określa gładkość funkcji.

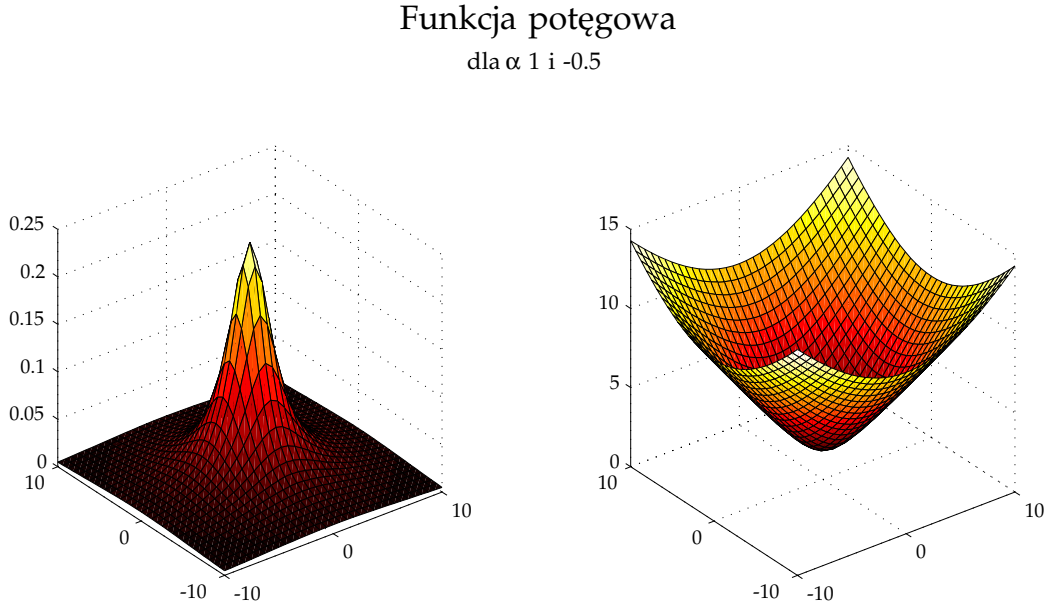
Innymi przykładami funkcji radialnych jest ogólniejsza wersja funkcji potęgowej (*ang. general multiquadratics*) i funkcja sklejana (rys. 1.10, 1.8, 1.9, 1.7):

$$h_1(\mathbf{x};\mathbf{t},b,\alpha) = (b^2+||\mathbf{x}-\mathbf{t}||^2)^{-\alpha}, \quad \alpha>0, \tag{1.62}$$

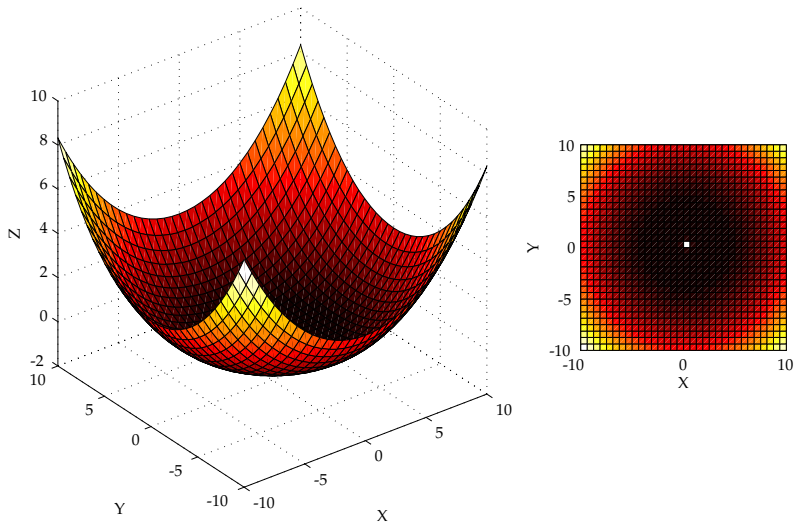
$$h_2(\mathbf{x};\mathbf{t},b,\beta) = (b^2+||\mathbf{x}-\mathbf{t}||^2)^{\beta}, \quad 0<\beta<1, \tag{1.63}$$

$$h_3(\mathbf{x};\mathbf{t},b) = (b||\mathbf{x}-\mathbf{t}||)^2\ln(b||\mathbf{x}-\mathbf{t}||). \tag{1.64}$$

Istnieją różne typy lokalnych radialnych funkcji bazowych. Niewątpliwie najczęściej wykorzystywaną spośród nich jest funkcja gaussowska (patrz rys. 1.10). Funkcja ta najczęściej występuje w kombinacji z odległością euklidesową, choć może być używaną równie dobrze z każdą inna miarą, która może być zapisana jako suma niezależnych komponentów, dzięki czemu funkcja gaussowska jest



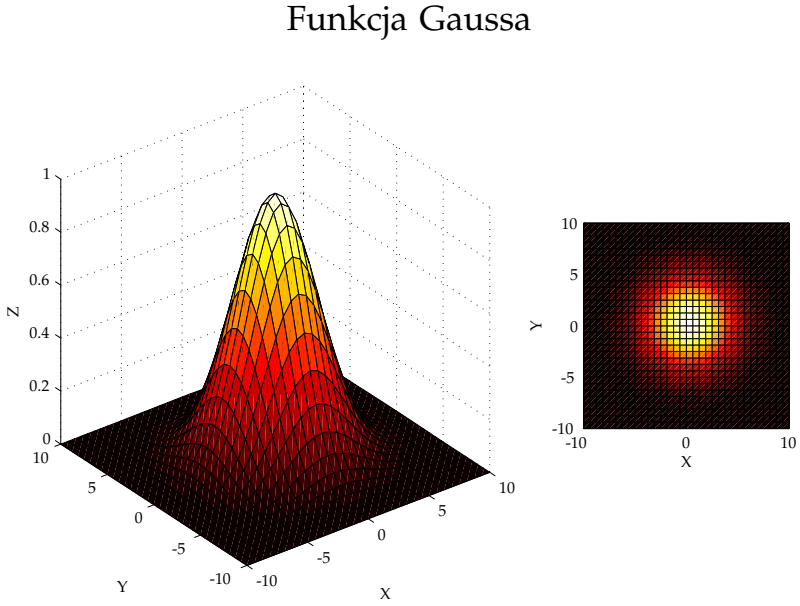
Rysunek 1.8: Funkcja potęgowa h_1 i h_2 (1.62).



Rysunek 1.9: Funkcja sklejana h_3 (1.64).

separowalna[2].Funkcja gaussowska jest zdefiniowana poniżej:

$$G(r, b) = e^{-r^2/b^2}. \tag{1.65}$$



Rysunek 1.10: Funkcja gaussowska (1.65).

Większość funkcji bicentralnych również posiada własność separowalności (por. podrozdział 1.4.6).

Choć moc przetwarzania sieci z nielokalnymi neuronami nie jest bardzo uzależniona od wyboru niewielomianowej funkcji, to w przypadku korzystania z funkcji lokalnych jest inaczej. Funkcja gaussowska $e^{-D(x)^2}$ jest prawdopodobnie najprostszą postacią funkcji lokalnej, chociaż nie najtańszą w sensie obliczeniowym. Funkcja logistyczna, tanh lub funkcje drugiego lub czwartego stopnia ze

[2]Dzięki separowalności można w dowolny sposób wybierać dowolną podprzestrzeń przestrzeni wejściowej i poddawać ją przeróżnym analizom. Jest to niemal koniecznym warunkiem w niektórych zastosowaniach, dla przykładu w wyciąganiu reguł logicznych z sieci typu RBF czy FSM, jak i w zastosowaniu do baz danych, w których napotyka się na wartości brakujące.

zlokalizowaną funkcją aktywacji aproksymują kształt funkcji Gaussa:

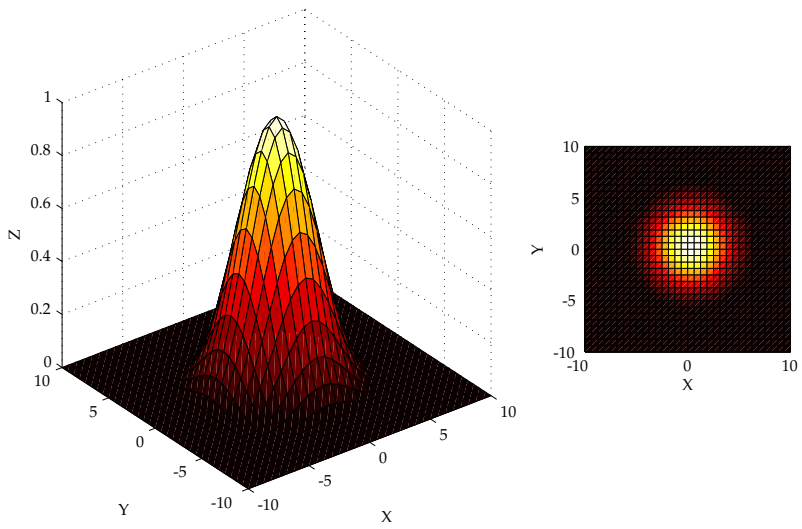
$$G_1(r) = 2 - 2\sigma(r^2), \tag{1.66}$$
$$G_2(r) = 1 - \tanh(r^2), \tag{1.67}$$
$$G_3(r) = \frac{1}{1+r^2}; \quad G_3'(r) = -2rG_3^2(r), \tag{1.68}$$
$$G_4(r) = \frac{1}{1+r^4}; \quad G_4'(r) = -4r^3G_4^2(r). \tag{1.69}$$

Niestety nie są to funkcje separowalne.

W [222] została zaproponowana kołowa funkcja sklejana trzeciego stopnia (*ang. radial cubic B-spline function*). Jest ona zdefiniowana poprzez:

$$RCBSpline(r) = \frac{1}{4h^2}\begin{cases} h^3 + 3h^2(h-r) + 3h(h-r)^2 + 3(h-r)^3 & r \le h, \\ (2h-r)^3 & h < r \le 2h, \\ 0 & 2h < r, \end{cases} \tag{1.70}$$

gdzie $r = ||\mathbf{x} - \mathbf{t}_i||^2$, a $\mathbf{t}_i$ to centrum. Rysunek 1.11 pokazuje przykład takiej funkcji.



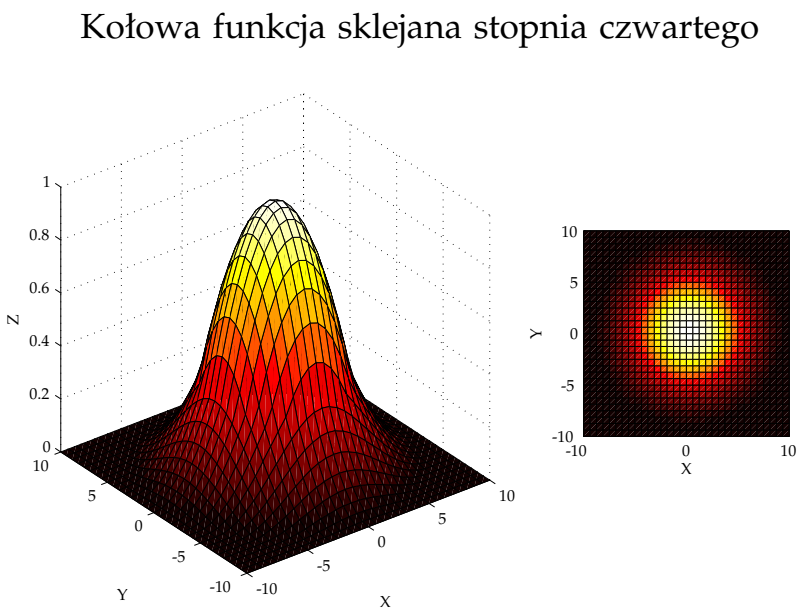
Rysunek 1.11: Kołowa funkcja sklejana trzeciego stopnia (1.70).

Saranli i Baykal [222] opisują też kołową funkcję sklejaną stopnia czwartego

(*ang. radially quadratic B-spline function*) zdefiniowaną przez poniższe równanie:

$$RQBSpline(r) = \frac{1}{2h^2} \begin{cases} -2r^2 + 3h^2 & r \leq h, \\ (2h - r)^2 & h < r \leq 2h, \\ 0 & 2h < r, \end{cases} \tag{1.71}$$

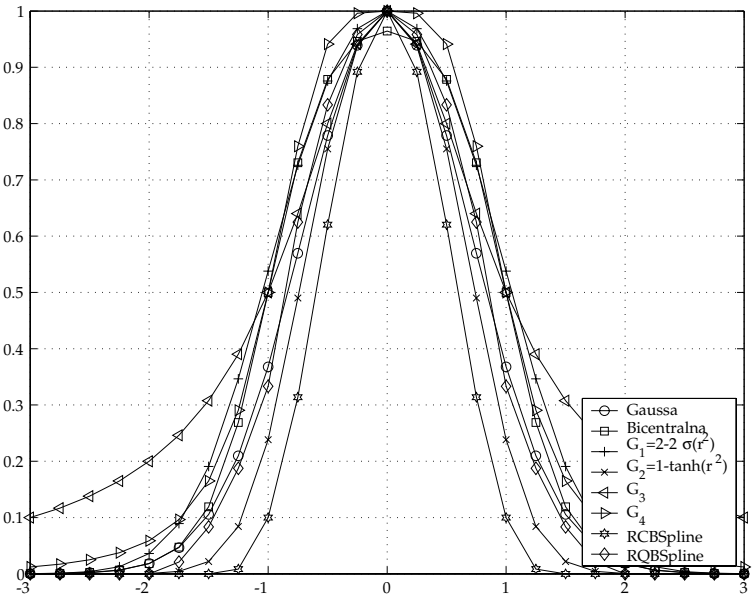
gdzie $r = ||\mathbf{x} - \mathbf{t}_i||$ i $\mathbf{t}_i$ jest i-tym centrum (patrz rys. 1.12, w artykule Saranli i Baykala [222] były błędy w definicjach powyższych dwóch funkcji, tutaj są one zaprezentowane już w poprawnej formie).



Rysunek 1.12: Kołowa funkcja sklejana czwartego stopnia (1.71).

Porównanie różnych funkcji zbliżonych kształtem do funkcji gaussowskiej przedstawiono na rys. 1.13.

Szybkość zbieżności radialnych funkcji bazowych ze stałym rozmyciem została opisana przez Niyogiego i Girosiego [195]. Ponieważ prawdziwa funkcja jest nieznana, błąd może być mierzony jedynie względem najlepszej możliwej estymacji tej funkcji, zwanej funkcją regresji $f_0(X)$. Różnice pomiędzy funkcją regresji i funkcją realizowaną poprzez sieć złożoną z radialnych funkcji bazowych z n neuronami, o d wymiarowej przestrzeni wejściowej i k wzorcach uczących,



Rysunek 1.13: Porównanie lokalnych funkcji wyjścia (patrz równania (1.65, 1.101, 1.66–1.70)).

estymuje z poziomem zaufania $1-\delta$ poniższe wyrażenie:

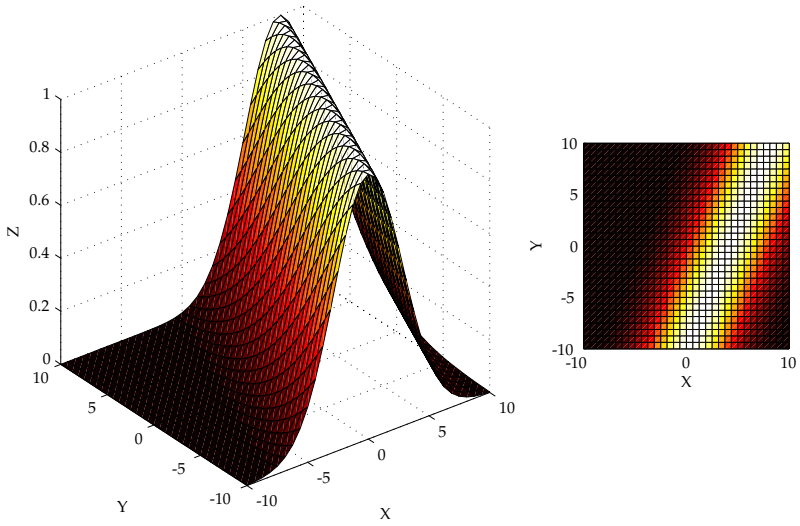
$$E\left[\left(f_0(\mathbf{x})-F_{n,k}(\mathbf{x})\right)^2\right] = \int_{\mathbf{x}} d\mathbf{x}\, P(\mathbf{x})\left(f_0(\mathbf{x})-F_{n,k}(\mathbf{x})\right)^2 \leq O\left(\frac{1}{n}\right)+O\left(\sqrt{\frac{nd\ln(nk)-\ln\delta}{k}}\right). \tag{1.72}$$

Pierwsza część wynika z teorii aproksymacji, $O(1/n)$, natomiast druga, $O\left(((nd\ln(nk)-\ln\delta)/k)^{1/2}\right)$, wynika ze statystyki. Błąd zanika tylko gdy złożoność sieci (sieć opisana jest poprzez n neuronów) jest nieporównanie mała względem liczby wzorców uczących k. Dla każdego wybranego zbioru wzorców istnieje pewna optymalna liczba neuronów, która minimalizuje błąd generalizacji.

Bardzo prostą i ciekawą funkcje uzyskuje się poprzez kombinację funkcji gaussowskiej wyjścia z aktywacją w postaci iloczynu skalarnego I:

$$W(\mathbf{x};\mathbf{w}) = e^{-[I(\mathbf{x};\mathbf{w})]^2}. \tag{1.73}$$

Tak zdefiniowana funkcja *okienkuje* (lokalizuje) maksimum aktywacji wzdłuż hiperpłaszczyzny opisanej przez aktywację $I(\mathbf{x};\mathbf{w})$. Szerokość takiego okna (od-

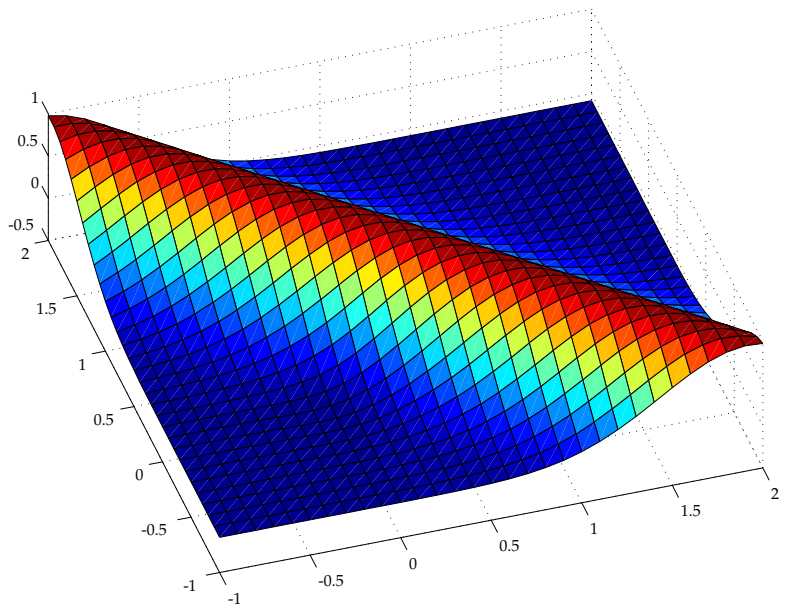


Rysunek 1.14: Funkcja okienkująca (1.73).

wrotnie proporcjonalna do $||\mathbf{w}||$), jego kierunek i przesunięcie jest kontrolowane przez wagi $\mathbf{w}$. Dzięki temu, że szerokość może być dowolna, można uznać iż taka funkcja może zastąpić funkcję sigmoidalną w skończonym wypukłym obszarze przestrzeni wejściowej. Bardzo podobną funkcją jest funkcja Lorentza (1.76). Z drugiej strony okienkowanie otwiera tak naprawdę zupełnie nowe możliwości. Rozpatrzmy problem nieparzystości (XOR) bardzo często używany do rozlicznych przykładów w zagadnieniach sieci neuronowych. Łatwo zauważyć, że mając do dyspozycji funkcję okienkującą wystarczy tak naprawę jeden neuron, aby rozwiązać ten problem. Co więcej (na-)uczenie takiego neuronu jest trywialne (problem w takiej przestrzeni funkcyjnej jest łatwo separowalny). Zobaczmy na rysunku 1.15 wyniki takiego uczenia. Z kolei z rozdziale 4.4 zobaczyć będzie można jak pewna sieć neuronowa spośród kilku różnych neuronów, które realizują różne funkcje transferu sieć najczęściej będzie wybierać właśnie funkcje okienkującą jako rozwiązania problemu XOR.

Oczywiście nie tylko w problemie XOR zachodzi potrzeba *wykrojenia* części przestrzeni i z pewnością nie warto zapominać o tej prostej, ale użytecznej i taniej pod względem liczby parametrów adaptacyjnych funkcji transferu.

W rozdziale o funkcjach uniwersalnych (1.4.5) zobaczyć będzie można jak działa aktywacja w postaci kombinacji iloczynu skalarnego z miarą odległości wraz z gaussowską funkcja wyjścia tworząc uniwersalną funkcje gaussowską (1.99).



Rysunek 1.15: Problem parzystości rozwiązany przy użyciu funkcji okienkującej (1.73).

1.3.3. Funkcje semi-centralne

Semi-centralne funkcje wyjścia wykorzystują funkcję aktywacji w postaci wektorowej. Dla przykładu wstęgowa funkcja Gaussa przyjmuje postać:

$$\bar{G}(\mathbf{r}, \mathbf{b}, \mathbf{v}) = \sum_{i=1}^{N} v_i e^{-r_i^2 / b_i^2}; \quad r_i = (x_i - t_i). \tag{1.74}$$

Natomiast rodzinę funkcji bicentralnych można określić wzorem:

$$Bi(\mathbf{r}; \mathbf{b}, \mathbf{s}) = \prod_{i=1}^{N} \sigma(I^+)(1 - \sigma(I^-)), \tag{1.75}$$

gdzie I^+ i I^- są zdefiniowane przez równania (1.45–1.48), tj. wykorzystują dwa wektory aktywacji. Więcej informacji można znaleźć w podrozdziale 1.4.6.

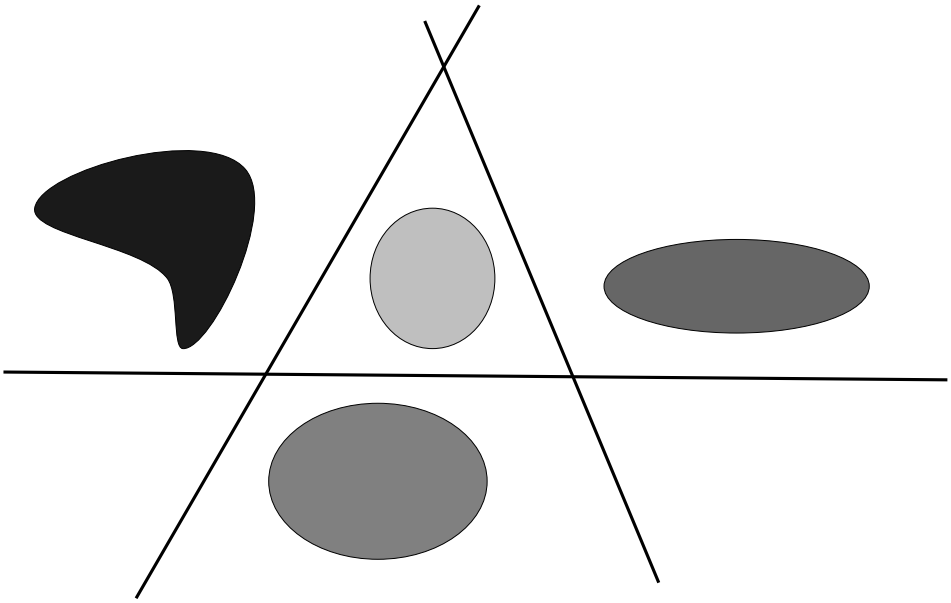
1.4. Funkcje transferu

W tym podrozdziale omawiane są różne kombinacje funkcji aktywacji z funkcjami wyjścia, tworzące przeróżne funkcje transferu. Funkcje transferu zostały

podzielone na lokalne i nielokalne. Wyodrębnione zostały też grupy funkcji o hiperelipsoidalnych konturach gęstości funkcji jak i te, które mogą być lokalne i nielokalne, w zależności od doboru parametrów wolnych funkcji, funkcje te będą nazywane *funkcjami uniwersalnymi*. Ostatnią grupę funkcji stanowi rodzina uniwersalnych funkcji bicentralnych.

1.4.1. Nielokalne funkcje transferu

Nielokalne funkcje transferu używane w sieciach neuronowych dzielą wielowymiarową przestrzeń wejściową, dokonując jej podziału na regiony odpowiadające różnym klasom lub różnym wartościom wektorów wyjściowych. Należy zwrócić uwagę, iż zmiana pojedynczego parametru może powodować istotne zmiany wartości wyjściowej całej sieci dla wszystkich punktów przestrzeni wejściowej. Dlatego też algorytm uczenia musi dokonywać zmian wszystkich parametrów adaptacyjnych w odpowiedniej kolejności. Typowe sigmoidalne funkcje transferu (1.5, 1.50–1.52) oparte o liniową kombinację wejść (1.1), jako aktywację, używane w sieciach MLP do klasyfikacji, jak i aproksymacji, zostały już opisane w podrozdziale 1.3.1.



Rysunek 1.16: Podział na regiony decyzji uformowane przy użyciu funkcji sigmoidalnych z aktywacją zdefiniowaną przez (1.1).

Obszary decyzji jakie powstają przy użyciu takich funkcji w klasyfikacji danych są tworzone przez przecięcia wielowymiarowej przestrzeni wejściowej hiperpłaszczyznami (patrz rysunek 1.16). Taki model *udaje*, że wie wszystko, a to

może być niepożądane, głównie w tych miejscach przestrzeni wejściowej, w których nie było żadnych danych uczących. W takich właśnie obszarach dochodzi do arbitralnej klasyfikacji. Funkcje sigmoidalne wygładzają całkowitą funkcję wyjściową sieci. Dla problemów klasyfikacyjnych może to być nawet przydatne, lecz ogólnie, w uczeniu odwzorowań (w mapowaniu) może to jednak ograniczać precyzję modelu adaptacyjnego.

Radialne funkcje bazowe można znaleźć w wielu różnych pakietach programów sieci neuronowych, lecz w większości tych pakietów dostępna jest tylko funkcja gaussowska, która jest lokalna. Jak już zostało wspomniane, do nielokalnych funkcji bazowych należą funkcje potęgowe (1.62), funkcje sklejane (1.64). Korzystają one z radialnej aktywacji opartej o normę Minkowskiego lub inną miarę odległości (por. rysunki 1.8 i 1.9). Funkcje te dają gęstości elipsoidalne. Giraud (i in.) [104] użyli liniowej kombinacji wejść do stworzenia funkcji Lorentzowskiej (patrz rys. 1.17):

$$L(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + I^2(\mathbf{x}; \mathbf{w})} = \frac{1}{1 + \left(\sum_{i=0}^{N} w_i x_i\right)^2}, \tag{1.76}$$

gdzie $x_0 = 1$.

Funkcja Lorentzowska nie ma gęstości elipsoidalnych czy kolistych tak jak radialne funkcje bazowe. Powierzchnia stałej gęstości jest w tym przypadku typu okna funkcji nielokalnej, którego połowa szerokości jest równa $1 / \sqrt{\sum_i w_i^2}$. Nielokalne funkcje typu *okna* mogą być uzyskane z wielu lokalnych i semi-lokalnych funkcji opisanych poniżej, gdy tylko produkt poszczególnych składowych tych funkcji nie obejmuje wszystkich wymiarów.

Potęgowy iloczyn tensorowy został wprowadzany do algorytmu MARS przez Friedmana [91] (patrz rys. 1.18). Zdefiniowany jest poprzez

$$T(\mathbf{x}; \mathbf{t}, \mathbf{s}, \mathcal{I}) = \prod_{i \in \mathcal{I}} [s_i(x_i - t_i)]_+^q, \tag{1.77}$$

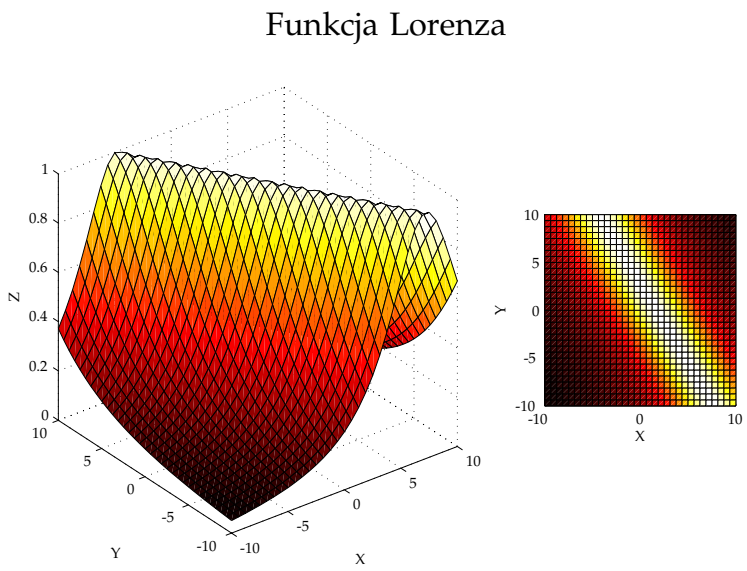
gdzie $\mathcal{I}$ jest pewnym podzbiorem cech wejściowych, q jest stałą ustaloną przez użytkownika, wektor $\mathbf{t}$ definiuje położenie centrum, a s_i definiuje kierunek i może być równe 1 lub -1, z kolei $[\,\cdot\,]_+$ oznacza, że wartości mniejsze od zera są zastępowane zerami.

Iloczyn tensorowy w algorytmie MARS został użyty jako komponent do budowy aproksymatora jako jego liniowej kombinacji:

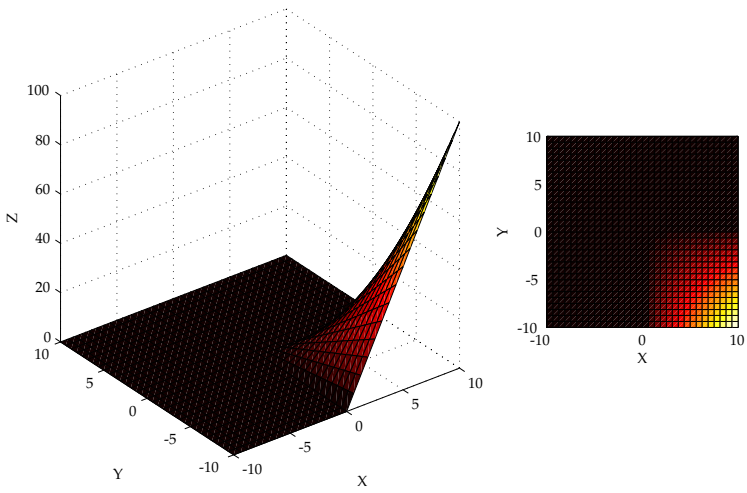
$$MARS(\mathbf{x}; \mathbf{w}, \mathbf{T}, \mathbf{S}, \mathcal{I}) = \sum_{i=1}^{N} w_i \; T(\mathbf{x}; \mathbf{t}_i, \mathbf{s}_i, \mathcal{I}_i) + w_0. \tag{1.78}$$

1.4.2. Lokalne i semi-lokalne funkcje transferu

Zlokalizowane funkcje transferu używają parametrów, które mają lokalny wpływ na zmiany wartości sieci, czyli zmiany wartości parametrów funkcji. Powodują zmiany wartości całkowitego wyjścia sieci tylko dla wektorów położonych



Rysunek 1.17: Funkcja Lorentzowska (1.76).



Rysunek 1.18: Funkcja bazowa z potęgowego iloczynu tensorowego.

w lokalnym rejonie oddziaływania przestrzeni wejściowej. Były czynione różne próby używania funkcji zlokalizowanych wśród wczesnych modeli sieci neuronowych. Niektóre z nich dotyczyły prac w rozpoznawaniu wzorców [98]. Moody i Darken [188] używali zlokalizowanych neuronów do uczenia odwzorowań rzeczywistych danych i do klasyfikacji, wykorzystując metody samoorganizacji i uczenia z nadzorem. Wybór zlokalizowanych neuronów miał na celu przyśpieszenie procesu uczenia w sieciach wstecznej propagacji. Bottou i Vapnik [19] pokazali siłę metod opartych o zlokalizowane uczenie w ogólności. Z kolei Kadirkamanathan i Niranjan [149] pokazali, że dla konstruktywistycznych metod warunek gładkości dla dodawania nowych neuronów jest spełniony tylko dla neuronów z odpowiednio małym rozmyciem funkcji Guassowskiej.

Najczęściej wykorzystywanym typem aktywacji funkcji RBF jest euklidesowa miara odległości, choć prosto można ją uogólnić, czy zmienić, do dowolnej innej miary $D(\mathbf{x};\mathbf{t})$, gdzie $\mathbf{t}$ jest centrum funkcji bazowej. Funkcja aktywacji minimum osiąga dla centrum: $\mathbf{x} = \mathbf{t}$ i monotonicznie rośnie wraz z oddalaniem się od centrum wektora $\mathbf{x}$. Odległość Hamminga jest często wykorzystywana dla wejść binarnych. Można też wzbogacić miarę aktywacji o czynniki skalujące (por. 1.14), co prowadzi do dodania nowych N parametrów adaptacyjnych.

Najprostszym sposobem na zbudowanie sieci neuronowej z radialnymi funkcjami bazowymi jest zgrupowanie pewnej liczby n funkcji radialnych $G_i(\mathbf{x})$ z uprzednio wyznaczonymi parametrami b i centrami $\mathbf{t}$ i wyznaczenie współczynników w_i. Pozycje centrów mogą być wyznaczone przy użyciu klasteryzacji k-średnich (*ang. k-means clastering*) i ustaleniu rozmyć b na dwukrotną wartość najmniejszej odległości pomiędzy funkcjami bazowymi [162]. Wtedy sieć RBF można przedstawić w poniższej postaci:

$$f(\mathbf{x};\mathbf{w},\mathbf{T},\mathbf{b}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x},\mathbf{t}_i,b_i) = \sum_{i=1}^{M} w_i e^{-||\mathbf{x}-\mathbf{t}_i||^2/b_i^2}. \tag{1.79}$$

W uogólnionej sieci RBF z regularyzacją również centra funkcji bazowych $\mathbf{t}_i$ podlegają adaptacji [209], pozwalając w ten sposób zredukować liczbę funkcji bazowych adekwatnie do szumu, jaki może się znajdować w danych (to właśnie składa się na regularyzację aproksymowanej funkcji). Szczegółowe informacje o różnych algorytmach uczenia sieci RBF można znaleźć w rozdziale 2.

W N wymiarowej przestrzeni centrum i-tej funkcji bazowej jest opisane przez N współczynników wektora $\mathbf{t}_i$ i jeden parametr b_i, który wyznacza rozmycie funkcji. W prosty sposób można dokonać uogólnienia funkcji gaussowskiej z odległością euklidesową umożliwiając adaptacje rozmycia dla każdego z wymiarów przestrzeni wejściowej niezależnie, co daje w rezultacie $2N$ parametrów adaptacyjnych na jeden neuron.

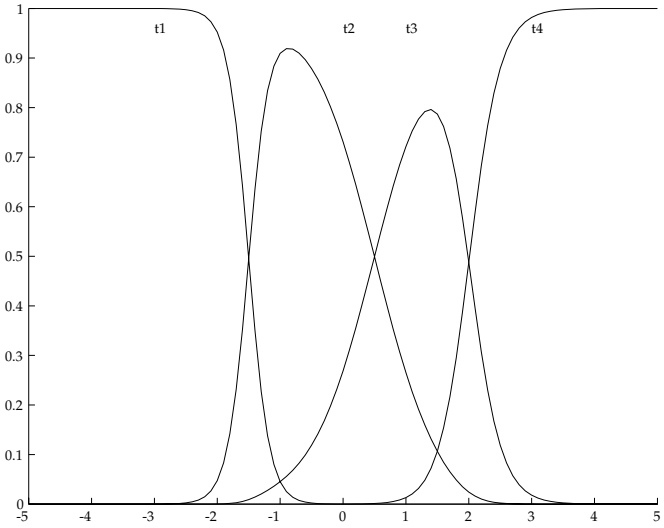
Moody i Darken [188] zaproponowali używanie znormalizowanej wersji funkcji Gaussa:

$$GN(\mathbf{x};\mathbf{t},b) = \frac{e^{-||\mathbf{x}-\mathbf{t}||^2/b^2}}{\sum_{j=1}^{M} e^{-||\mathbf{x}-\mathbf{t}_j||^2/b_j^2}}. \tag{1.80}$$

Interesującą właściwością tej funkcji jest, że dla każdego wektora $\mathbf{x}$ z przestrzeni wejściowej, suma wartości funkcji w tym punkcie jest równa 1:

$$\sum_{i=1}^{M} GN(\mathbf{x}; \mathbf{t}_i, b_i) = 1. \tag{1.81}$$

Dzięki temu, wyjście każdej z tak określonych funkcji można interpretować jako prawdopodobieństwo wyznaczane przez *i*-ty neuron. Na rysunku 1.19 są pokazane wyjścia (kompletu) czterech znormalizowanych funkcji Gaussa. Bridle [26] zaproponował nazywanie tych funkcji funkcjami *soft-max*, co było motywowane wyżej przedstawionymi właściwościami tych funkcji.



Rysunek 1.19: Znormalizowana funkcja Gaussa — softmax.

Bazując na znormalizowanej funkcji Gaussa można zbudować prosty i zarazem bardzo ciekawy model klasyfikacyjny (NRBFN, por. rozdział 6.4). Taki model będzie określał prawdopodobieństwa przynależności do poszczególnych klas. W rozdziale 6.4 proponuje się też wykorzystanie znormalizowanych funkcji Gaussa w systemie regularyzacji danych, który może być używany do usuwania nietypowych wartości, bądź ważenia ich wpływu na proces uczenia wybranego algorytmu adaptacyjnego. Taka kontrola wpływu poszczególnych wektorów może być postrzegana jako mechanizm stabilizacji procesu uczenia.

1.4.3. Gaussowska i sigmoidalna funkcja wstęgowa

Na wejścia sieci neuronowych nie zawsze są podawane istotne informacje. Mamy wtedy do czynienia ze zbędnymi cechami, czy też cechami o zmniejszonym

udziale w procesie adaptacji. Typowe funkcje radialne nie są zdolne do automatycznego wyeliminowania takich cech. Problem ten był rozpatrywany przez Hartmana i Keelera [114, 113] także przez Parka i Sandberga [203]. W przeciwieństwie do wielowymiarowej funkcji gaussowskiej autorzy tworzą funkcje transferu przez kombinacje jednowymiarowych ważonych funkcji gaussowskich (rys. 1.20):

$$G_b(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{v}) = \sum_{i=1}^{N} v_i e^{-(x_i-t_i)^2/b_i^2}. \tag{1.82}$$

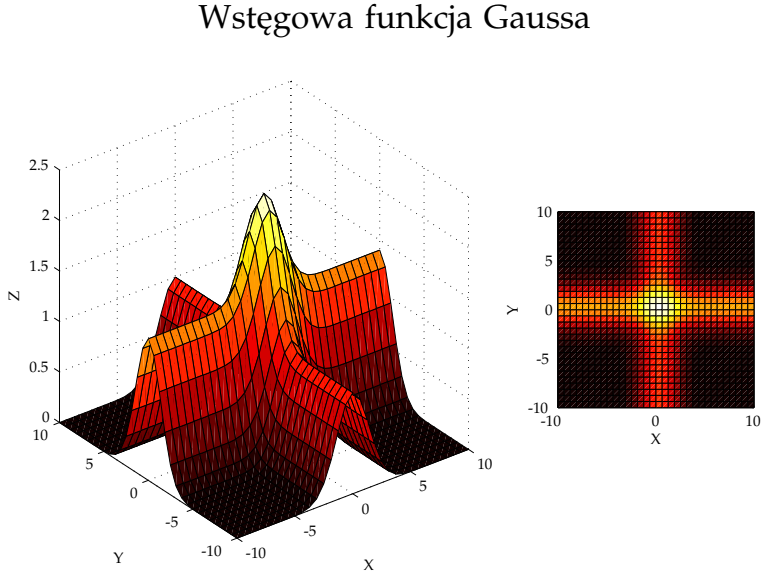
W tym przypadku funkcje aktywacji i wyjścia nie są separowalne. Funkcja wstęgowa ma $3N$ parametrów adaptacyjnych na jeden neuron. Oryginalna angielska nazwa funkcji *Gaussian bar functions* pochodzi od jej charakteru w wielowymiarowej przestrzeni. Dla przestrzeni wielowymiarowych N wartości poszczególnych wag wstęg v_i mogą być znacznie mniejsze niż suma wszystkich N wag v_i wokół $\mathbf{t}$. W warstwie wyjściowej stosuje się funkcję sigmoidalną dla wygładzenia funkcji sieci i zredukowania liczby małych maksimów.

Funkcje wstęgowe pozwalają na eliminację nieistotnych wymiarów wejściowych prowadząc do redukcji wymiarów i to w prostszy sposób niż np. w przypadku wielowymiarowej funkcji gaussowskiej. Rozmycia w poszczególnych wymiarach również prowadzą do redukcji wymiarów (por. przykład odwzorowania logistycznego kwadratowego, Moody i Darken [188]). Inna zaleta użycia funkcji wstęgowej płynie z samej istoty istnienia wstęg, które podlegają sumowaniu. Pojedyncze maksimum lub kilka odseparowanych maksimów mogą być opisane przez małą liczbę funkcji gaussowskich (z $N+1$ parametrami na funkcję) a także przez taką samą liczbę funkcji wstęgowych z niemal trzykrotnie większą liczbą parametrów. Lecz gdy w danych wejściowych znajduje się k klastrów w regularnych odstępach w każdym z N wymiarów, formują w ten sposób k^N klastrów w N wymiarowym hipersześcianie. Takie dane wejściowe wymagają tylu funkcji gaussowskich wielu zmiennych, co klastrów. Natomiast funkcji wstęgowych wystarczy Nk.

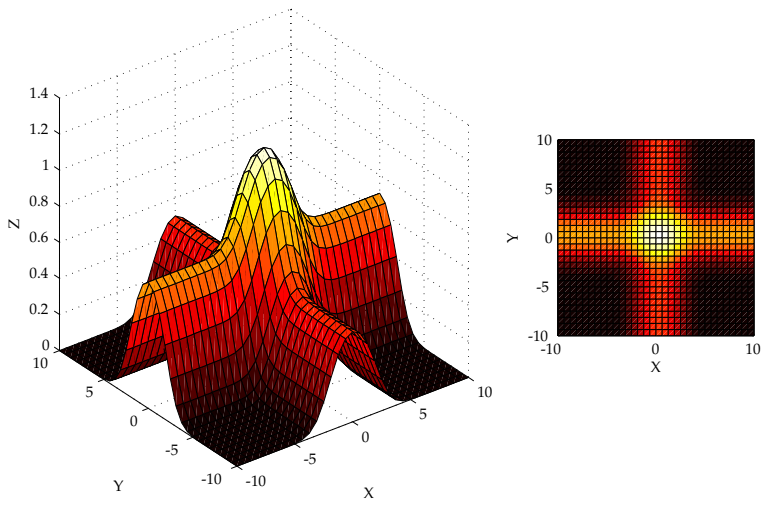
Można utworzyć podobne złożenie, funkcji wstęgowej sigmoidalnej, (*ang. sigmoidal bar function*), korzystając z funkcji sigmoidalnych (patrz rys. 1.21):

$$\sigma_b(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{v}) = \sum_{i=1}^{N} \frac{v_i}{1+e^{-(x_i-t_i)^2/b_i^2}}. \tag{1.83}$$

Ta funkcja, podobnie jak i gaussowska funkcja wstęgowa, daje kontury o stałych gęstościach, których obrócenie nie jest proste, co też jest wadą tych funkcji. Sigmoidalna funkcja wstęgowa nie powinna być używana do reprezentacji klastra danych zlokalizowanych wokół pewnego centrum, ponieważ każdy taki klaster będzie potrzebował $2N$ funkcji sigmoidalnych, podczas gdy wystarczyłaby jedna funkcja gaussowska. Jednak gdy klastry są rozłożone regularnie na kwadratowej siatce, to k^2 klastrów może być reprezentowanych przez tyle samo funkcji gaussowskich lub jedynie $2k$ sigmoidalnych funkcji wstęgowych, czy k gaussowskich funkcji wstęgowych. Z kolei, aby te same klastry przestrzeni wej-



Rysunek 1.20: Wstęgowa funkcja Gaussa (1.82).



Rysunek 1.21: Sigmoidalna funkcja wstęgowa (1.83).

ściowej opisać przy użyciu hiperpłaszczyzn lub funkcji sigmoidalnych, potrzeba by ich $2k-2$.

1.4.4. Funkcje o gęstościach elipsoidalnych

Wielowymiarowa funkcja gaussowska jest przykładem funkcji, której kontury stałych wartości tworzą hiperelipsoidy (patrz rys. 1.22):

$$G_g(\mathbf{x};\mathbf{t},\mathbf{b}) = e^{-D^2(\mathbf{x};\mathbf{t},\mathbf{b})} = \prod_{i=1}^{N} e^{-(x_i-t_i)^2/b_i^2}. \tag{1.84}$$

Rozmycia b_i mogą być rozumiane jako współczynniki skalujące w funkcji odległości D^2 (1.12). Podobny rezultat uzyskuje się przy użyciu kombinacji funkcji logistycznych (lub innych funkcji sigmoidalnych) z kwadratem funkcji odległości, na przykład (patrz rys. 1.23):

$$\begin{aligned} G_S(\mathbf{x};\mathbf{t},\mathbf{b}) &= 1-\sigma(D^2(\mathbf{x};\mathbf{t},\mathbf{b})), & (1.85)\\ &= \frac{1}{1+\prod_{i=1}^{N} e^{(x_i-t_i)^2/b_i^2}} = \frac{1}{1+e^{D^2(\mathbf{x};\mathbf{t},\mathbf{b})}}. & (1.86) \end{aligned}$$

W N wymiarowej przestrzeni wejściowej każda funkcja elipsoidalna używa $2N$ parametrów adaptacyjnych. Używając odległości Mahalanobisa $D_M(\mathbf{x};\mathbf{t})$ (por. 1.16), z symetryczną macierzą kowariancji Σ, dostajemy możliwość dowolnej rotacji w wielowymiarowej przestrzeni hiperelipsoidnych konturów stałych wartości funkcji. Gdy potraktować elementy macierzy kowariancji jako parametry adaptacyjne, będzie to równoważne metryce tensorowej z funkcją odległości:

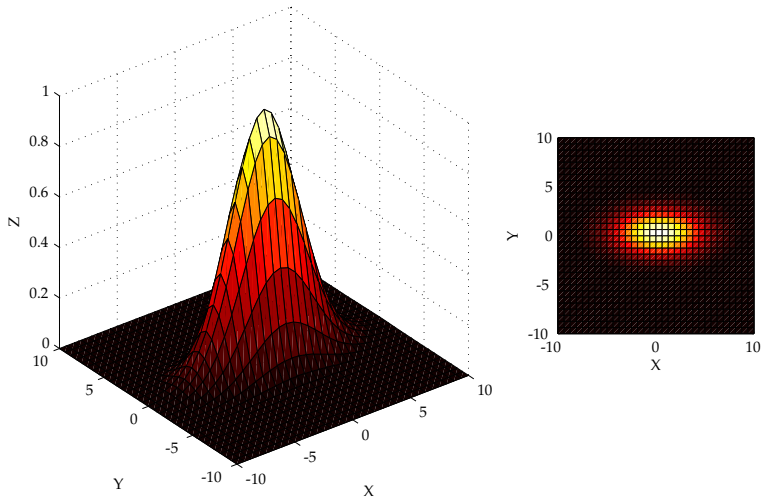
$$D_Q^2(\mathbf{x};\mathbf{Q};\mathbf{t}) = \sum_{i\geq j} Q_{ij}(x_i-t_i)(x_j-t_j). \tag{1.87}$$

Całkowita liczba parametrów adaptacyjnych na jeden neuron wynosi $N(N+3)/2$. Powierzchnie funkcji są dane w ogólnej formie kwadratowej i tym samym mogą być elipsoidalne, paraboliczne czy hiperboliczne.

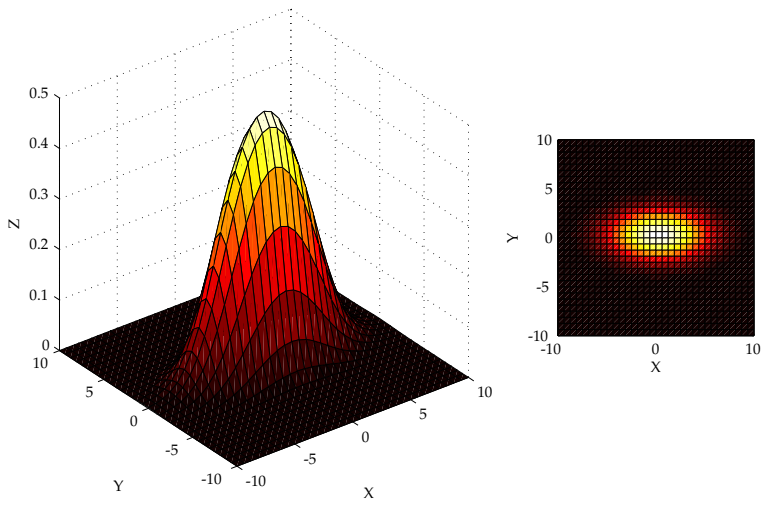
Pojedynczy neuron może być jeszcze bardziej złożony jeśli zostałaby zastosowana ogólniejsza funkcja odległości. Należy jednak unikać zbyt wielkiej liczby nieliniowych parametrów na jeden neuron. Można też podać jeszcze prostszą postać funkcji o elipsoidalnych konturach stałych wartości funkcji (patrz rys. 1.24):

$$\bar{G}_2(\mathbf{x};\mathbf{t},\mathbf{b}) = \prod_{i=1}^{N} \frac{1}{1+(x_i-t_i)^2/b_i^2}. \tag{1.88}$$

Tego wzoru nie można jednak przedstawić w podobnej formie jak powyższe funkcje, uzależniając go od pewnej funkcji odległości ponieważ nie jest to funkcja radialna.



Rysunek 1.22: Funkcja Gaussa wielu zmiennych (1.84).



Rysunek 1.23: Funkcja sigmoidalna wielu zmiennych (1.85).

Przez liniową aproksymację funkcji G_S (pomijając wielowymiarowy iloczyn) otrzymujemy kwadratową funkcję odległości w mianowniku (patrz rys. 1.25):

$$\bar{G}_3(\mathbf{x};\mathbf{t},\mathbf{b}) = \frac{1}{1+\sum_{i=1}^{N}(x_i-t_i)^2/b_i^2} = \frac{1}{1+D^{\mathbf{b}}(\mathbf{x};\mathbf{t},\mathbf{b})}. \tag{1.89}$$

Funkcje te również dają hiperelipsoidalne kontury stałych wartości.

Udało się też zaobserwować ciekawą własność funkcji gaussowskiej G_g (1.84) dzięki całkiem prostej renormalizacji[3]:

$$G_R(\mathbf{x};\mathbf{t},\mathbf{b}) = \frac{G_g(\mathbf{x};\mathbf{t},\mathbf{b})}{G_g(\mathbf{x};\mathbf{t},\mathbf{b})+G_g(\mathbf{x};-\mathbf{t},\mathbf{b})} = \frac{1}{1+e^{-4\sum_{i=1}^{N}x_it_i/b_i^2}}, \tag{1.90}$$

funkcja G_R staje się nielokalna i jest równoważna funkcji sigmoidalnej $\sigma(\mathbf{x};\mathbf{p})$ dla $p_i = b_i^2/4t_i$.

Tym samym można używać sieci RBFN zamiast MLP i vice versa, a prosta transformacja wejścia umożliwia użycie sieci MLP ze zlokalizowanymi funkcjami transferu [57, 58].

1.4.5. Uniwersalne funkcje transferu

Połączenie liniowego czynnika we wzorze aktywacji $I(\mathbf{x};\mathbf{w})$ i kwadratowego czynnika używanego w liczeniu odległości Euklidesowej, daje w rezultacie funkcję, która dla pewnych parametrów staje się lokalna, a dla innych nielokalna (dalej takie funkcje będą też nazywane funkcjami semi-lokalnymi). Ridella i in. [215] używali neuronów kołowych w ich w sieci wstecznej propagacji. Funkcją wyjścia jest standardowa funkcja sigmoidalna lecz funkcja aktywacji posiada dodatkowy człon (patrz rys. 1.26):

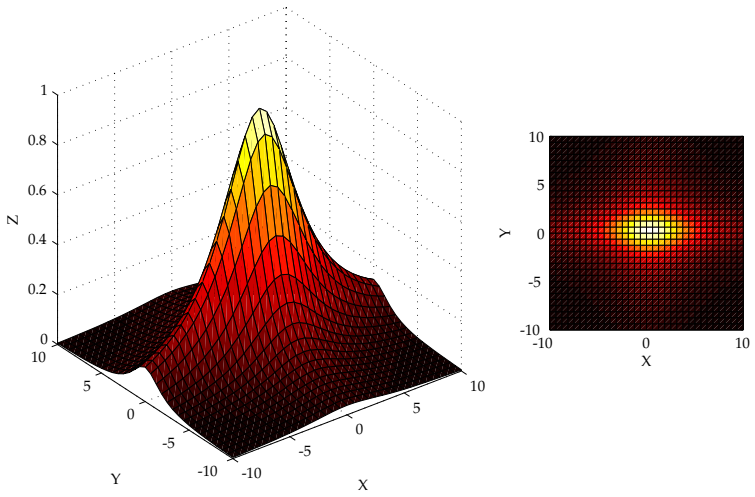
$$A_R(\mathbf{x};\mathbf{w}) = w_0 + \sum_{i=1}^{N} w_i x_i + w_{N+1}\sum_{i=1}^{N} x_i^2. \tag{1.91}$$

I_c może też być przedstawione w formie odległości:

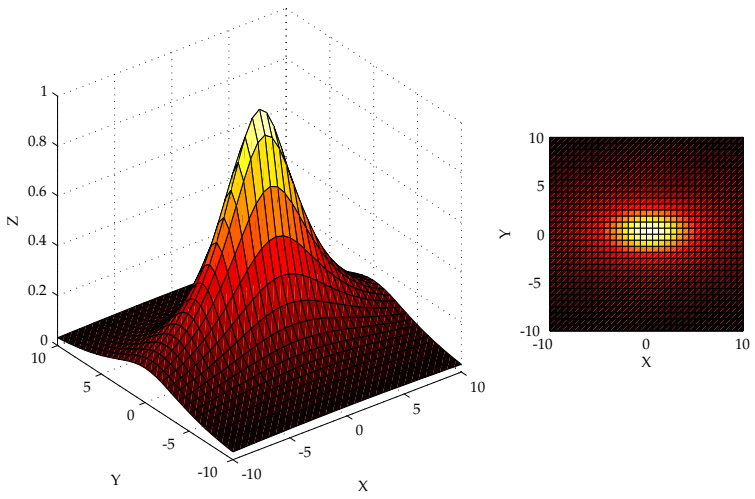
$$\begin{aligned} A_R(\mathbf{x};\mathbf{w}) &= d_c(\mathbf{x};\mathbf{c}) = (||\mathbf{x}-\mathbf{c}||^2-\theta)w_{N+1}, \\ c_i &= -w_i/2w_{N+1}; \quad \theta = \frac{1}{w_{N+1}}\left(\sum_{i=1}^{N}\frac{w_i^2}{4w_{N+1}^2} - w_0\right). \end{aligned} \tag{1.92}$$

Ridella i in. [215] uzyskali bardzo dobre rezultaty przy użyciu tych neuronów w sieci ze wsteczną propagacją. Udowodnili też, iż w wielu przypadkach neurony kołowe dają optymalne rozwiązanie dla problemów klasyfikacyjnych. Inny typ neuronu kołowego został zaproponowany przez Kirbiego i Mirande

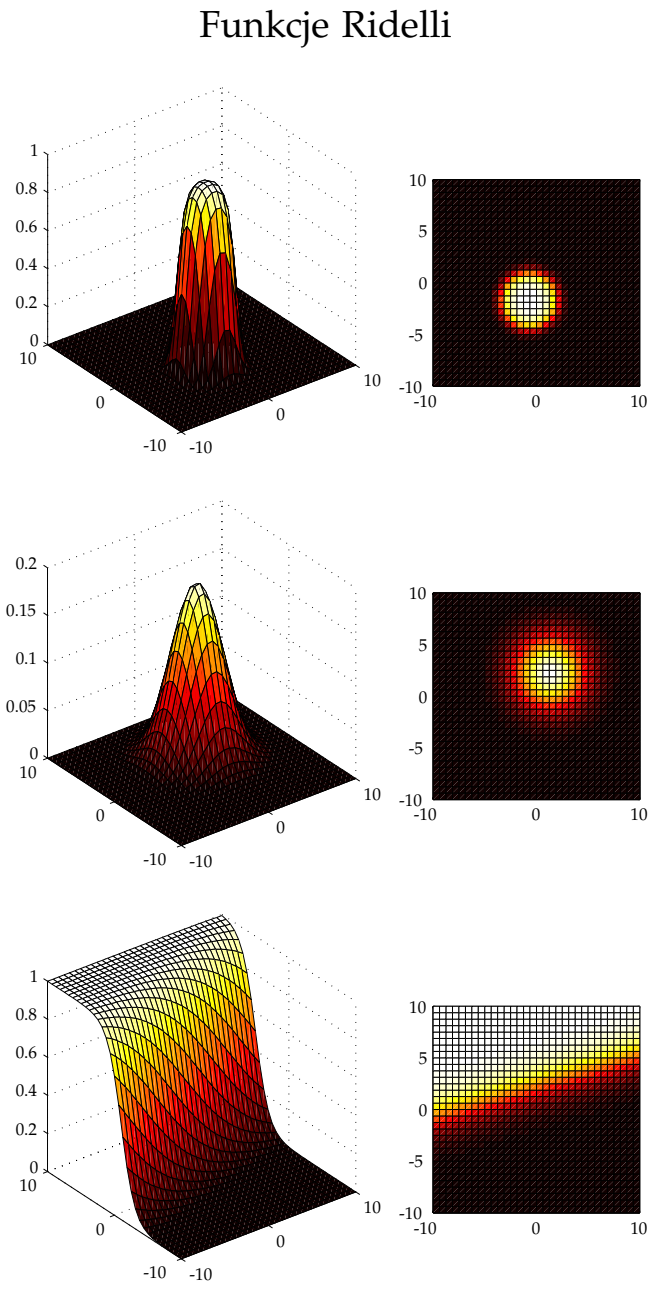
[3] Własność tę pokazał Igor Grabiec z Uniwersytetu w Lublanie Włodzisławowi Duchowi w prywatnej dyskusji.



Rysunek 1.24: Funkcja $\bar{G}_2$ (1.88).



Rysunek 1.25: Funkcja $\bar{G}_3$ (1.89).



Rysunek 1.26: Funkcja kołowa Riddelli (1.91).

[155]. W ich rozwiązaniu dwa neurony sigmoidalne współdziałają razem, a ich wspólne wyjście jest ograniczane tak, aby leżało w kole jednostkowym.

Powyżej zdefiniowany neuron kołowy Ridelli można rozbudować o możliwość obrotu w wielowymiarowej przestrzeni poprzez dodanie jedynie N kolejnych punktów swobody.

$$A_{GR}(\mathbf{x};\mathbf{w}) = w_0 + \sum_{i=1}^{N} w_i x_i + w_{N+1}[x + rx^r]^T[x + rx^r], \tag{1.93}$$

gdzie $x^r = [x_2, \ldots, x_{N-1}, x_1]$. Taka funkcja staje się już silnym narzędziem. Może obracać kontury stałych wartości funkcji, dobierać promienie hiperelipsoidy, bądź przekształcać się w funkcje nielokalną tak, jak podstawowa funkcja kołowa (patrz rys. 1.27).

Dorffner [56] zaproponował użycie stożkowej funkcji transferu jako funkcji uniwersalnej dla sieci MLP i RBFN. Linie proste i elipsy to tylko specjalne przypadki funkcji stożkowej. Z geometrycznych rozważań, Dorffner proponuje połączenie dwóch funkcji aktywacji: liniowej kombinacji wejść i funkcji odległości (patrz rys. 1.28), co razem daje aktywację:

$$\begin{aligned} A_C(\mathbf{x};\mathbf{w},\mathbf{t},\omega) &= I(\mathbf{x}-\mathbf{t};\mathbf{w}) + \omega D(\mathbf{x}-\mathbf{t}), \\ &= \sum_{i=1}^{N+1} w_i(x_i - t_i) + \omega \sqrt{\sum_{i=1}^{N+1}(x_i - t_i)^2}. \end{aligned} \tag{1.94}$$

Aktywacja ta jest później przekształcana przez funkcję sigmoidalną, by uzyskać końcową - stożkową funkcję transferu:

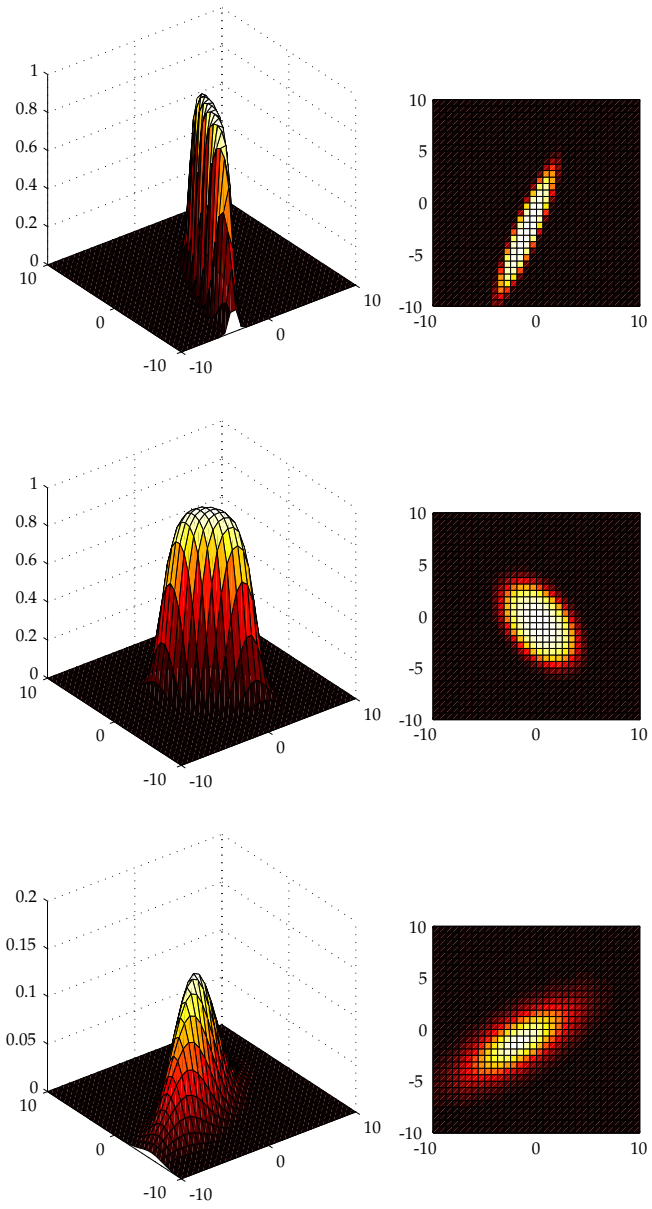
$$C_C(\mathbf{x};\mathbf{w},\mathbf{t},\omega) = \sigma(A_C(\mathbf{x};\mathbf{w},\mathbf{t},\omega)). \tag{1.95}$$

Jeszcze ciekawsze kontury stałych wartości można uzyskać rozbudowując nieco aktywacje funkcji stożkowej (patrz rys. 1.29) do:

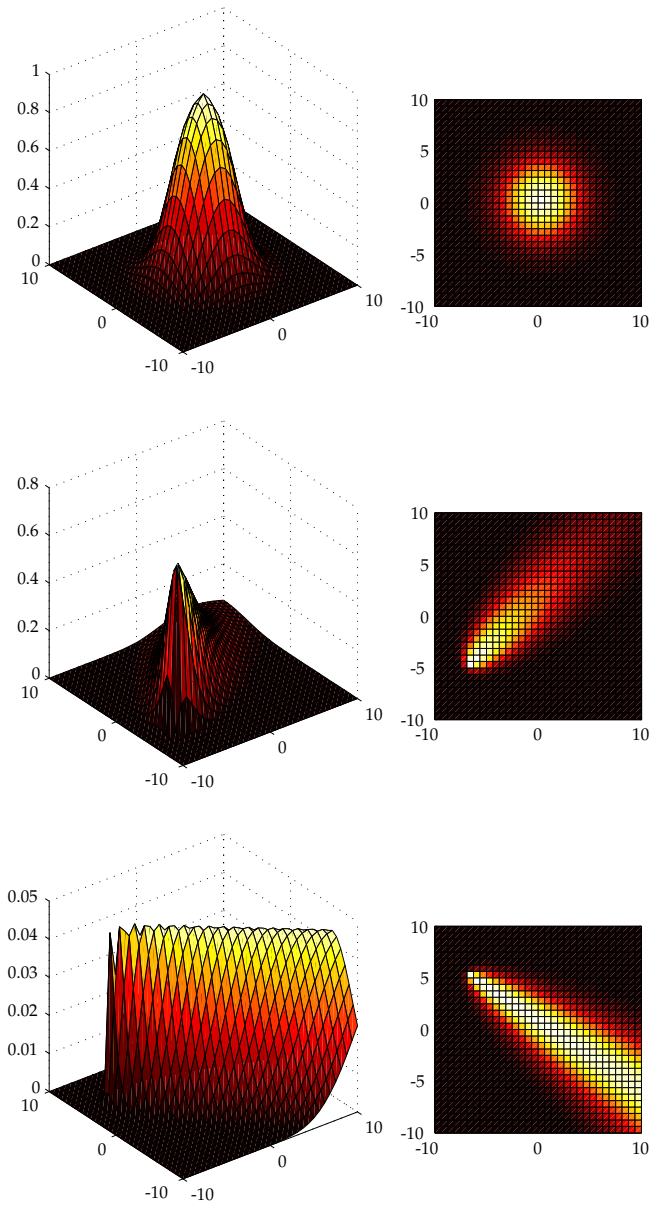
$$A_{GC}(\mathbf{x};\mathbf{w},\mathbf{t},\mathbf{b},\alpha,\beta) = -[\alpha I(\mathbf{x}-\mathbf{t};\mathbf{w}) + \beta D(\mathbf{x},\mathbf{t},\mathbf{b})]. \tag{1.96}$$

Również i ta postać funkcji stożkowej używa funkcji sigmoidalnej jako funkcji wyjścia. Funkcja może także gładko przechodzić od funkcji gaussowskiej do funkcji sigmoidalnej, lecz jest bardziej elastyczna i ma niezależne rozmycia w poszczególnych wymiarach. Można też kontrolować skos części elipsoidalnej konturów stałych wartości. Użycie takich funkcji transferu jest równoznaczne z jednoczesnym używaniem wielowymiarowych funkcji Gaussa i funkcji sigmoidalnych w jednej sieci neuronowej. Jednak tu dobór typu, czy nawet i kombinacji dokonuje się samoistnie.

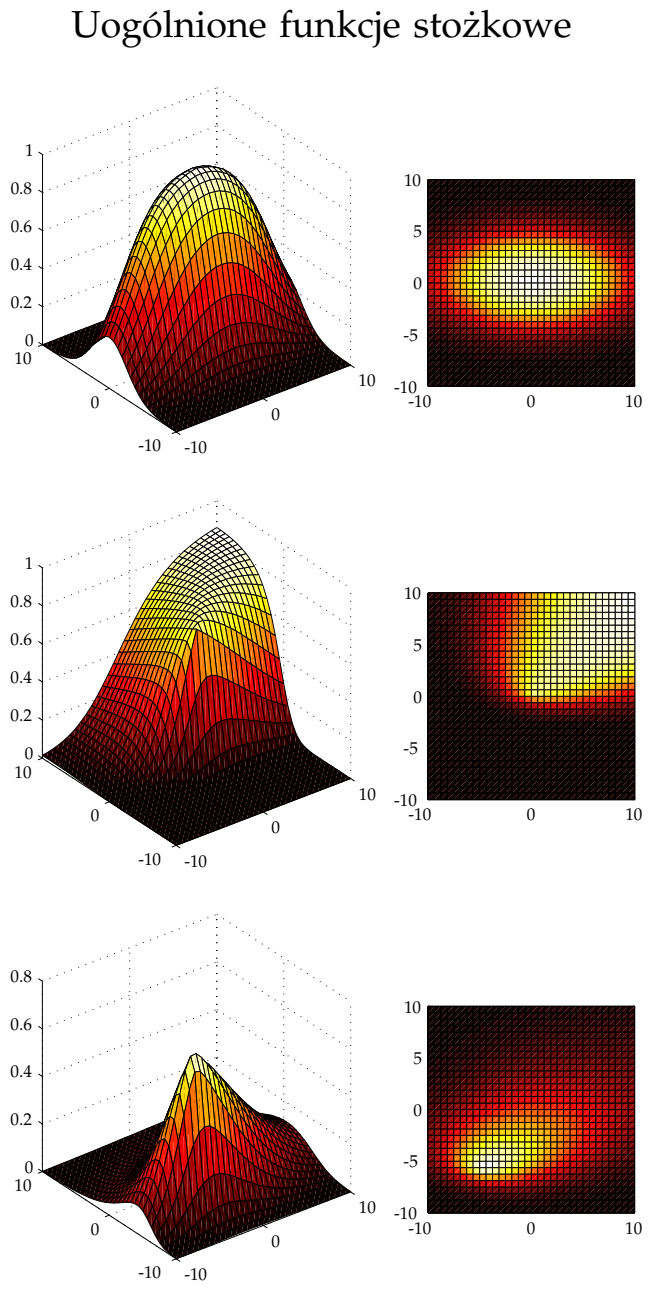
Korzystając z takich funkcji transferu i modyfikując funkcję błędu, można z jednej strony faworyzować przekształcenie się tak zdefiniowanej funkcji w funkcję gaussowską lub sigmoidalną, a z drugiej strony, gdy problem rozwiązywany przez daną sieć neuronową jest trudniejszy, dopuszczać wykorzystanie jednoczesne obu członów aktywacji. Problem ten jest szerzej opisany w rozdziale 4.4.



Rysunek 1.27: Funkcja kołowa z obrotem (1.93).



Rysunek 1.28: Funkcje stożkowe (1.95).



Rysunek 1.29: Uogólnione funkcje stożkowe (1.96).

Z powyższych rozważań można łatwo wywnioskować, że daje się skonstruować i inne funkcje uniwersalne w oparciu o liniową kombinację wejść i funkcje odległości. Na przykład $\exp(\alpha I^2 - \beta D^2)$ lub aproksymacja funkcji gaussowskiej połączona z funkcją Lorentza (1.76), dają ciekawe funkcje uniwersalne (patrz rys. 1.30):

$$C_{GL1}(\mathbf{x}; \mathbf{w}, \mathbf{t}, b, \alpha) = \frac{1}{1 + (I(\mathbf{x}; \mathbf{w}) + \alpha D(\mathbf{x}; \mathbf{t}, b))^2} \tag{1.97}$$

lub

$$C_{GL2}(\mathbf{x}; \mathbf{w}, \mathbf{t}, b, \alpha, \beta) = \frac{1}{1 + \alpha I^2(\mathbf{x}; \mathbf{w}) + \beta D^2(\mathbf{x}; \mathbf{t}, b)}. \tag{1.98}$$

Dla uproszczenia można przyjąć iż $\beta = 1 - \alpha$. Parametr α waży względny udział liniowej i nieliniowej części w całości funkcji. W tym przypadku liczba adaptacyjnych parametrów wynosi $2N + 1$ — gdy nie ma odrębnego parametru skalującego przy części z funkcją odległości; lub $3N + 1$ — gdy są odrębne parametry skalujące przy liniowej i nieliniowej części.

Należy zwrócić uwagę, że funkcje C_{GL1} i C_{GL2} są uogólnieniem funkcji Lorentza (1.76) — posługują się tą samą funkcją wyjścia.

Aktywacja $I^2 + D^2$ dała bardzo ciekawy efekt obrotu poprzez wkład iloczynu skalarnego w kombinację aktywacji. Nic nie stoi na przeszkodzie, aby tak efektywną aktywację użyć do gaussowskiej funkcji wyjścia:

$$UG(\mathbf{x}; \mathbf{w}, \mathbf{t}, b, \alpha) = e^{-[I^2(\mathbf{x};\mathbf{w}) + D^2(\mathbf{x};\mathbf{t},b)]}. \tag{1.99}$$

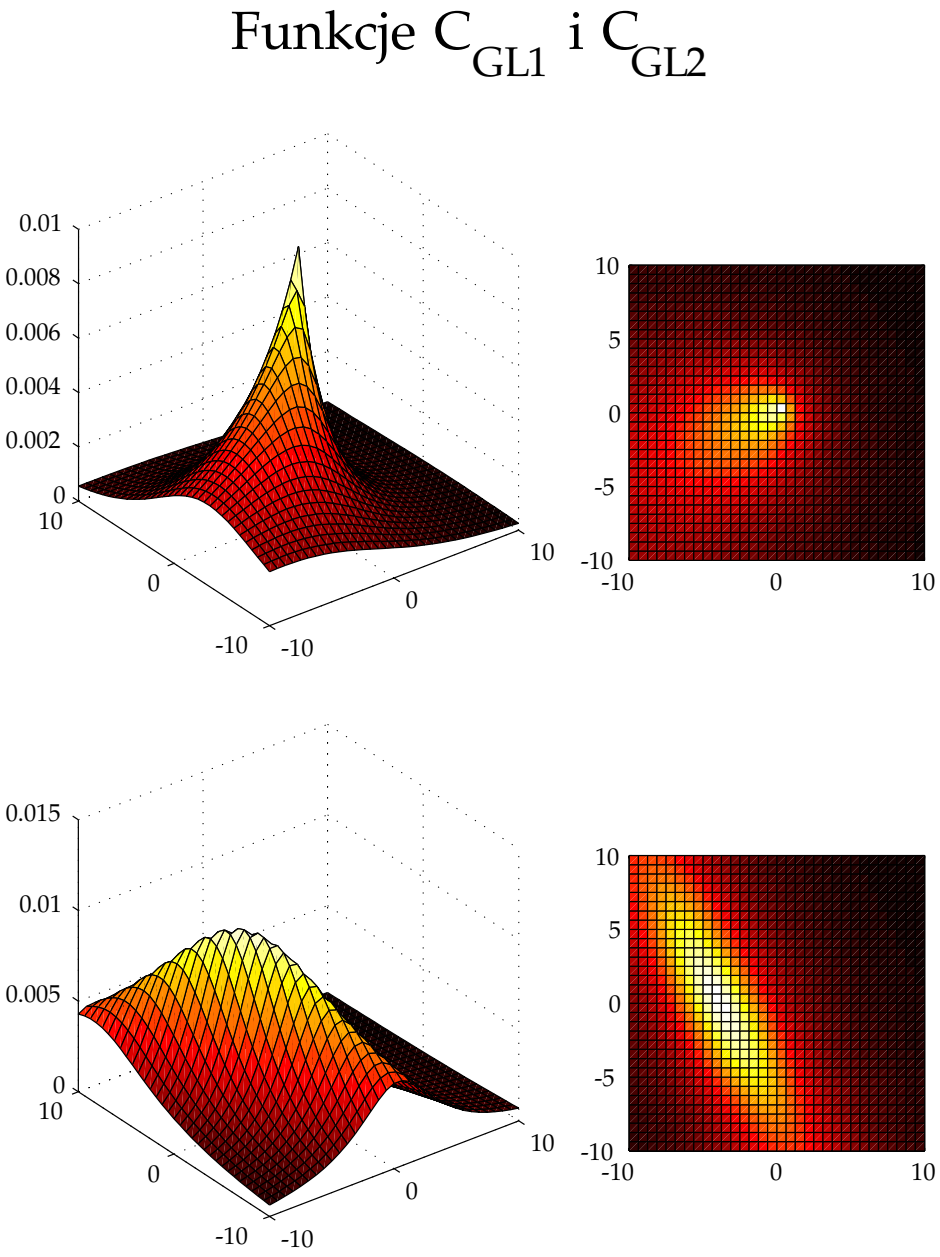
Taka funkcja transferu ma zbliżone własności do poprzedniej funkcji C_{GL2}. Funkcja może być lokalna bądź nielokalna. Podobnie jak poprzednia może rotować kontury o stałych wartości względem swojego centrum relatywnie tanim kosztem, ponieważ liczba parametrów wolnych to $2N + 1$. Przykłady funkcji uniwersalnej Gaussa można zobaczyć na rys. 1.31. Główną cechą tej funkcji jest możliwość płynnego przejścia od czystej funkcji Gaussa (1.65) do funkcji okienkującej (1.73).

Niestety funkcje uniwersalne są nieseparowalne (uniwersalna funkcja gaussowska byłaby separowalna, gdyby nie kwadrat iloczynu skalarnego). Dlatego też warto zapoznać się z bardzo ciekawymi funkcjami przedstawionymi w następnym podrozdziale, jak i tymi przedstawionymi w podrozdziale 1.4.7.

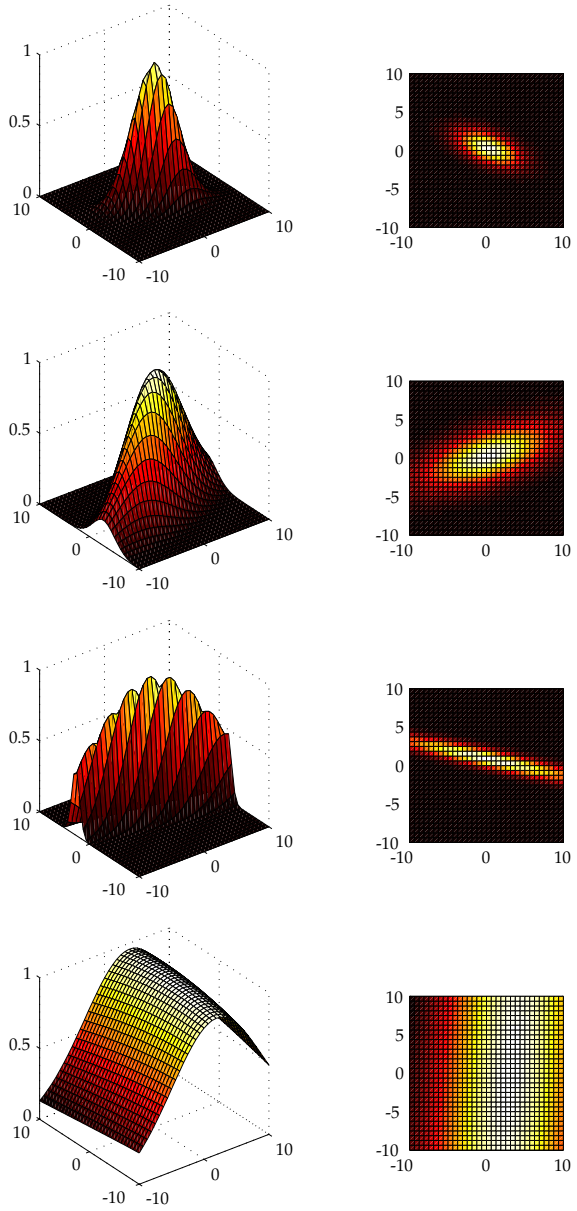
1.4.6. Funkcje bicentralne

Kombinacja dwóch funkcji sigmoidalnych daje możliwość utworzenia zlokalizowanej funkcji typu okno na kilka różnych sposobów. Dwoma najprostszymi sposobami są: różnica dwóch funkcji sigmoidalnych $\sigma(x) - \sigma(x - \theta)$ i iloczyn $\sigma(x)(1 - \sigma(x - \theta))$. Po renormalizacji obie formy stają się identyczne:

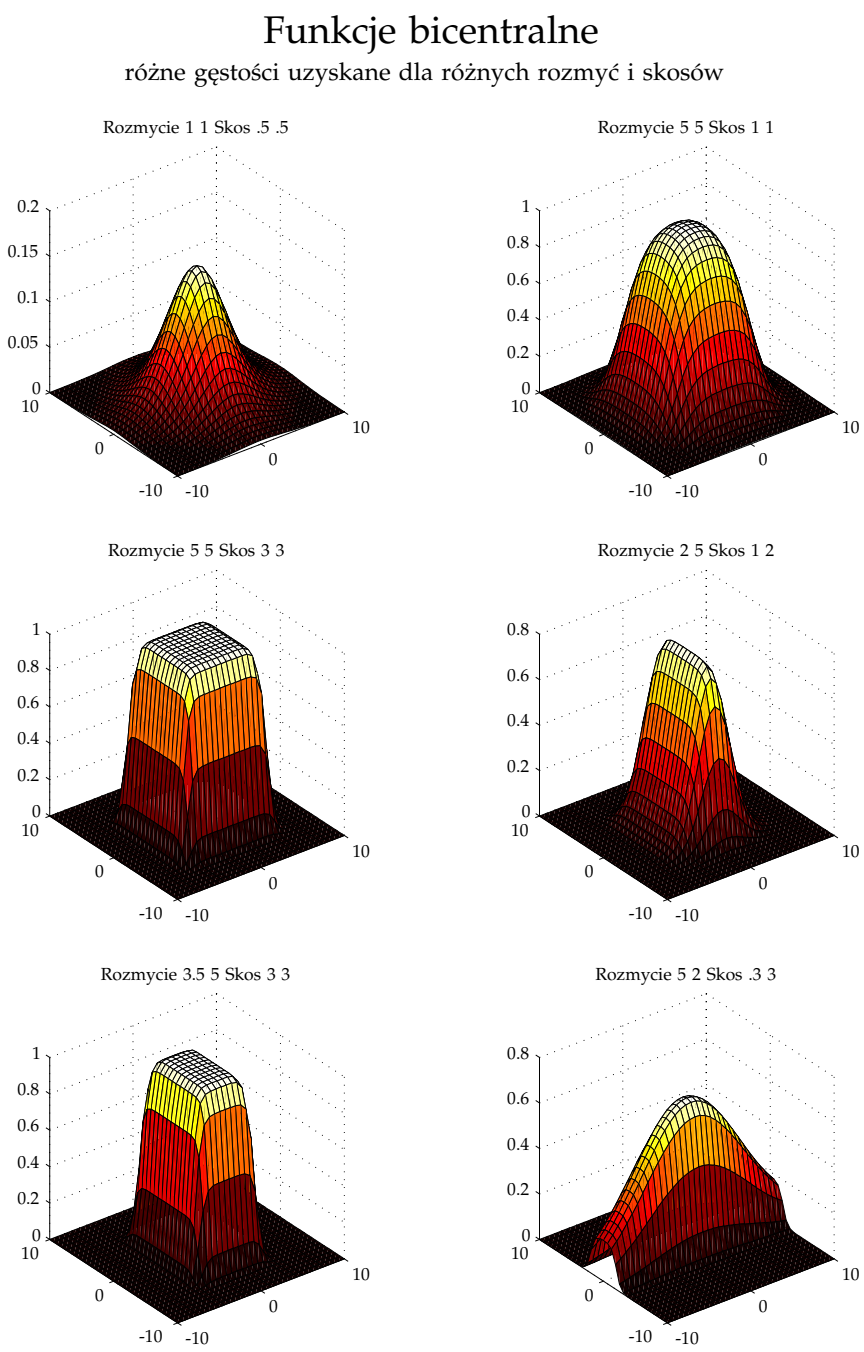
$$\frac{\sigma(x + b)(1 - \sigma(x - b))}{\sigma(b)(1 - \sigma(-b))} = \frac{\sigma(x + b) - \sigma(x - b)}{\sigma(b) - \sigma(-b)}. \tag{1.100}$$



Rysunek 1.30: Kombinacja aproksymacji funkcji gaussowskiej z funkcją Lorentza (1.97 i 1.98).



Rysunek 1.31: Uniwersalna funkcja Gaussa $G(\sqrt{I^2 + D^2})$ (1.99).



Rysunek 1.32: Kilka przykładów funkcji bicentralnych (1.101).

Gdy wziąć iloczyn takich kombinacji funkcji w każdym wymiarze, otrzymujemy wielowymiarową funkcję, która jest bardzo elastyczna, produkuje wypukłe regiony decyzyjne, wygodne dla klasyfikacji. Ogólna postać produktu w N wymiarowej przestrzeni wygląda tak (por. rys. 1.4.6):

$$\begin{aligned} Bi(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) &= \prod_{i=1}^{N}\sigma(A1_i^+)\ (1-\sigma(A1_i^-)), \\ &= \prod_{i=1}^{N}\sigma(e^{s_i}\cdot(x_i-t_i+e^{b_i}))\ (1-\sigma(e^{s_i}\cdot(x_i-t_i-e^{b_i}))), \end{aligned} \tag{1.101}$$

gdzie $\sigma(x) = 1/(1+e^{-x})$, $\mathbf{t}$ to centrum, $\mathbf{b}$ rozmycie, a $\mathbf{s}$ skos. Pierwszy człon z funkcją sigmoidalną rośnie gdy rosną wartości x_i, drugi człon będzie wtedy malał. To właśnie lokalizuje funkcję wokół punktu t_i w każdym z wymiarów. Kształt bicentralnej funkcji $Bi(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s})$ można adaptować poprzez przemieszczanie punktu centrum $\mathbf{t}$, zmianę rozmycia $\mathbf{b}$ i ustawianie skosu $\mathbf{s}$. Dzięki niezależności parametrów rozmyć i skosów, funkcje bicentralne mogą estymować bardziej skomplikowane kontury o stałych wartościach niż na przykład wielowymiarowe funkcje gaussowskie. Radialne funkcje bazowe są zdefiniowane względem tylko jednego centrum $||x-t||$. Tu są używane dwa centra $t_i+e^{b_i}$ i $t_i-e^{b_i}$. Stąd pochodzi nazwa funkcji bicentralnych. Produkt funkcji jest z kolei elastyczny i wypukły. Zostały też zastosowane funkcje eksponencjalne do rozmycia e^{b_i} i skosu e^{b_i} zamiast s_i i b_i. Zmniejszyło to oscylacje funkcji błędu podczas uczenia, co stabilizuje proces uczenia.

Liczba parametrów adaptacyjnych na jeden neuron wynosi $3N$. Jest też możliwa redukcja wymiarów, podobnie, jak w przypadku wstęgowych funkcji gaussowskich. Lecz i w tym przypadku funkcje bicentralne są bardziej elastyczne, a wykorzystują tyle samo parametrów co funkcje wstęgowe.

Funkcje bicentralne mogą zostać w prosty sposób rozszerzone do semi-lokalnych poprzez dodanie dwóch (lub jednego) parametrów:

$$\begin{aligned} SBi(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) &= \prod_{i=1}^{N}(\alpha+\sigma(A1_i^+))\ (1-\beta\sigma(A1_i^-)), \\ &= \prod_{i=1}^{N}(\alpha+\sigma(e^{s_i}\cdot(x_i-t_i+e^{b_i})))\ (1-\beta\sigma(e^{s_i}\cdot(x_i-t_i-e^{b_i}))). \end{aligned} \tag{1.102}$$

Funkcja nie zanika (jej wartości są różne od zera) dla dużych wartości $|x|$ (gdy tylko $\alpha \neq 0$ i $\beta \neq 0$). Dla $\alpha = 0$, $\beta = 0$ funkcja semi-bicentralna jest równoważna funkcji bicentralnej. Liczba parametrów adaptacyjnych funkcji semi-bicentralnej jest równa $3N+2$ jeśli parametry α i β są takie same dla wszystkich wymiarów przestrzeni wejściowej lub $5N$ w przeciwnym wypadku.

1.4.7. Rozszerzenia funkcji bicentralnych

1.4.7.1. Funkcje bicentralne z niezależnymi skosami.

Innym sposobem kontrolowania kształtu funkcji estymującej jest użycie niezależnych skosów dla kombinacji funkcji sigmoidalnych (patrz rys. 1.33):

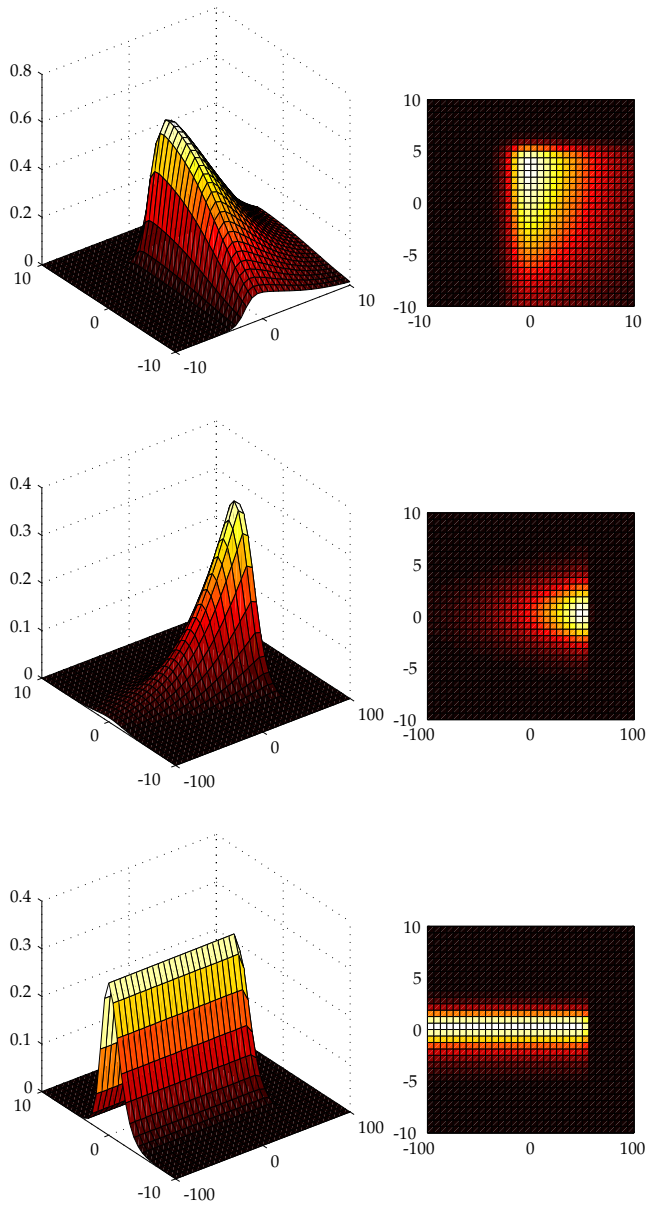
$$\begin{aligned} Bi2s(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) &= \prod_{i=1}^{N} \sigma(A2_i^+)\ (1-\sigma(A2_i^-)), \qquad (1.103)\\ &= \prod_{i=1}^{N} \sigma(e^{s_i}\cdot(x_i-t_i+e^{b_i}))\ (1-\sigma(e^{s_i'}\cdot(x_i-t_i-e^{b_i}))). \end{aligned}$$

Ustawiając skos s_i lub s_i' na małą wartość, funkcja bicentralna może się delokalizować lub też rozciągać na lewo i/lub prawo od punktu centrum t, w każdym z wymiarów niezależnie. To pozwala na uzyskanie takich konturów stałych wartości, jak półhiperwalec, półhiperelipsoida, *miękki* trójkąt i inne (patrz rys. 1.33). Choć liczba parametrów trochę wzrosła, i wynosi teraz $4N$ parametrów na jeden neuron, to jednak elastyczność adaptacji estymowanej funkcji jest widocznie większa.

Należy zwrócić uwagę, iż podobnie, jak funkcje bicentralne ta funkcja jest również separowalna co umożliwia różnym modelom analizę poszczególnych wymiarów przestrzeni wejściowej niezależnie. Można również, na przykład manipulując funkcją błędu (opisaną w następnym rozdziale równaniem 2.14), dodatkowo wymuszać preferowanie małych, łagodnych skosów. Prowadzi to do redukcji wymiarowości w poszczególnych neuronach niezależnie. Można też wymuszać preferencje dużych skosów i/lub rozciągać rozmycia, co przyczynia się bezpośrednio do możliwości interpretacji sieci jako reguł logicznych i również redukcji wymiarów. Metody interpretacji sieci opartych głównie o funkcje bicentralne były opisane w [69, 1, 64].

1.4.7.2. Funkcje bicentralne z rotacją.

Funkcja bicentralna opisana w podrozdziale 1.4.6, posiadająca $3N$ parametrów adaptacyjnych na neuron, jest wystarczająca do reprezentowania wielu różnych klas funkcji. Z kolei funkcja semi-bicentralna czy bicentralna z niezależnymi skosami może opisywać zlokalizowane, jak i niezlokalizowane funkcje. Następnym krokiem w kierunku zwiększenia elastyczności funkcji transferu jest dołączenie możliwości rotacji konturów stałych wartości poszczególnych neuronów w wielowymiarowej przestrzeni wejściowej [70, 69]. Oczywiście można użyć pełnej macierzy przekształcenia w przestrzeni wejściowej na wektorze, $\mathbf{Rx}$, lecz w praktyce macierz taka jest bardzo trudna do adaptacji (zawiera $N \times N$ parametrów), poza tym jest w niej jedynie $N-1$ niezależnych kątów obrotu (na przykład eulerowskich). Dużych kłopotów nastręczałoby też wyznaczanie pochodnych dla algorytmów uczenia opartych na metodzie spadku gradientu. Można zaproponować dwie drogi rozwiązania tego problemu. W obu będzie możliwy obrót we



Rysunek 1.33: Przykłady funkcji bicentralnych z niezależnymi skosami (1.103).

wszystkich wymiarach przestrzeni przy użyciu jedynie $N-1$ lub N parametrów obrotu. W pierwszym przypadku jest to produkt kombinacji sigmoid opartych o transformacje na wektorze wejściowym (patrz rys. 1.34):

$$\begin{aligned} C_P(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{R}) &= \prod_i^N \Big(\sigma(A3_i^+) - \sigma(A3_i^-)\Big), \qquad (1.104)\\ &= \prod_i^N \Big(\sigma(\mathbf{R}_i\mathbf{x}+t_i) - \sigma(\mathbf{R}_i\mathbf{x}+t_i')\Big),\\ SC_P(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{p},\mathbf{r},\mathbf{R}) &= \prod_i^N \Big(p_i\cdot\sigma(A3_i^+) + r_i\cdot\sigma(A3_i^-)\Big), \qquad (1.105)\\ &= \prod_i^N \Big(p_i\cdot\sigma(\mathbf{R}_i\mathbf{x}+t_i) + r_i\cdot\sigma(\mathbf{R}_i\mathbf{x}+t_i')\Big), \end{aligned}$$

gdzie $\mathbf{R}_i$ jest i-tym wierszem macierzy rotacji $\mathbf{R}$ o następującej strukturze:

$$\mathbf{R} = \begin{bmatrix} s_1 & \alpha_1 & 0 & & \cdots & 0\\ 0 & s_2 & \alpha_2 & 0 & & \\ & & & & & \\ \vdots & & & \ddots & & \vdots\\ & & & & s_{N-1} & \alpha_{N-1}\\ 0 & \cdots & & & 0 & s_N \end{bmatrix}, \qquad (1.106)$$

gdzie s_i to parametry opisujące skosy, a parametry α_i opisują obrót funkcji względem jej centrum $\mathbf{t}-\mathbf{t}'$.

Gdy $p_i = 1$ i $r_i = -1$ funkcja SC_P staje się funkcją lokalną, równoważną funkcji C_P i podobną do funkcji bicentralnej, pomijając możliwość rotacji. Przypisując inne wartości parametrom p_i i r_i funkcja SC_P delokalizuje się.

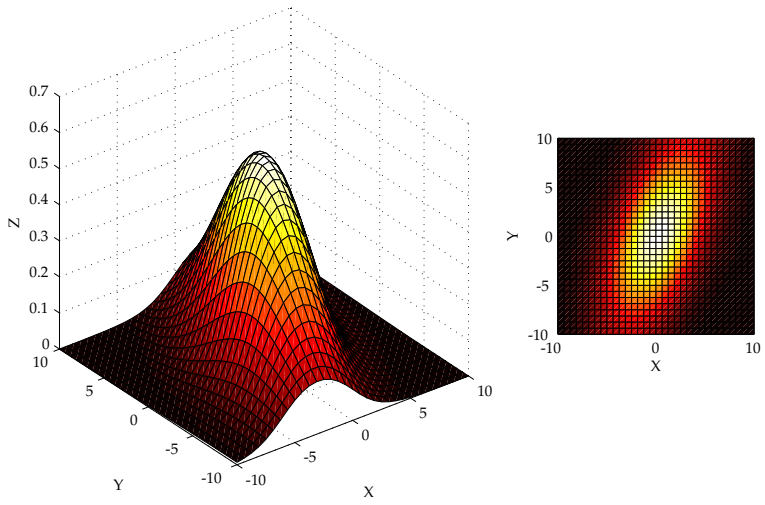
Drugi sposób uzyskania funkcji z możliwością obrotu polega na utworzeniu sumy kombinacji funkcji sigmoidalnych typu okna $L(x;t,t') = \sigma(x+t) - \sigma(x+t')$ w $N-1$ wymiarach i kombinacji obróconej przez wektor $\mathbf{K}$:

$$C_K(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{w},\mathbf{K}) = \sum_{i=1}^{N-1} w_i L(x_i,t_i,t_i') + w_N L(\mathbf{K}\mathbf{x},t,t'). \qquad (1.107)$$

Czyniąc funkcje $C_K(\cdot)$ funkcją aktywacji, a funkcje sigmoidalną funkcją wyjścia z odpowiednim progiem, na wyjściu otrzymamy gęstości prostopadłe do $\mathbf{K}$. Alternatywnie można użyć też iloczynu kombinacji a nie sumy:

$$C_{PK}(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{K}) = L(\mathbf{K}\mathbf{x},t,t') \prod_{i=1}^{N-1} L(x_i,t_i,t_i'), \qquad (1.108)$$

w tym przypadku nie musimy używać funkcji sigmoidalnej jako funkcji wyjścia.



Rysunek 1.34: Funkcje bicentralne z rotacją (1.104).

Rotacja w tych funkcjach dodaje jedynie $N-1$ parametrów dla funkcji $C_P(\cdot)$ i N parametrów dla funkcji $C_K(\cdot)$.

Rotacje (które mają możliwość adaptacji podczas procesu uczenia) zostały, jak dotąd zaimplementowane jedynie w dwóch typach sieci neuronowych, w Feature Space Mapping [65, 1, 64] i IncNet [146, 140, 141] opisanej w tej pracy.

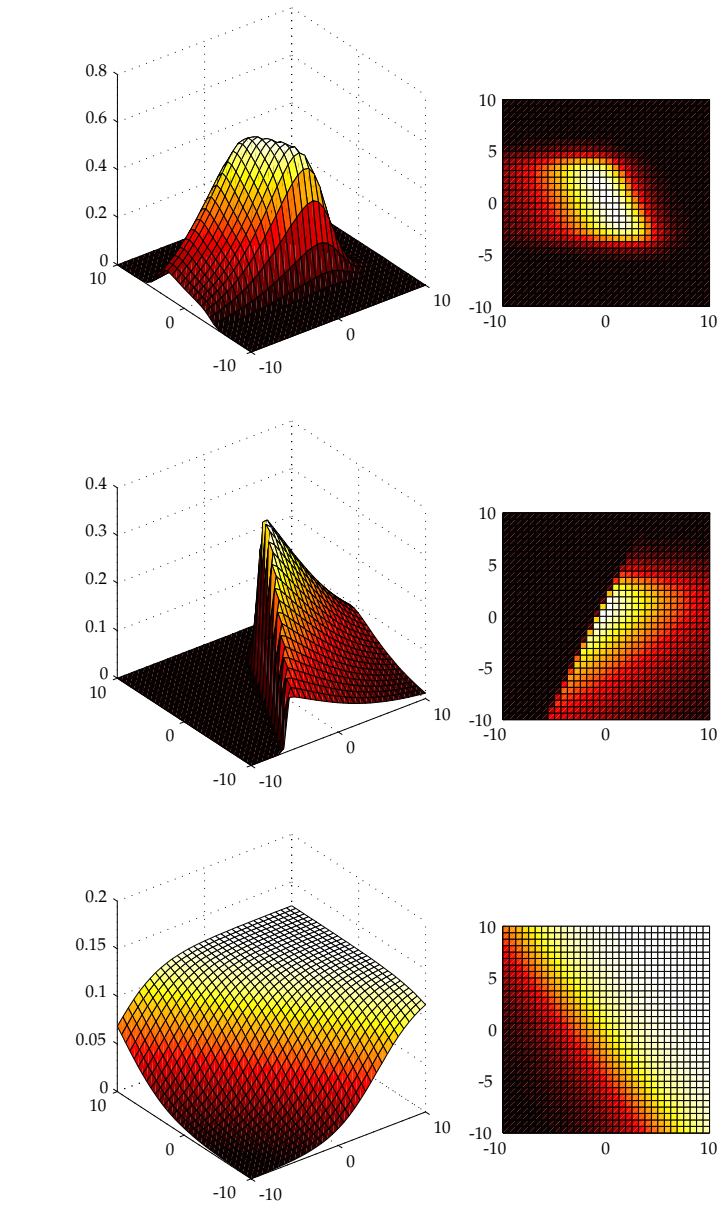
1.4.7.3. Funkcje bicentralne z rotacją i niezależnymi skosami.

Można również połączyć możliwości funkcji bicentralnych z obrotem i niezależnymi skosami, tworząc jeszcze elastyczniejszą funkcję transferu (patrz rys. 1.35):

$$BiR2s(\mathbf{x};\mathbf{t},\mathbf{t}',\alpha) = \prod_i^N \sigma(A4_i^+)(1-\sigma(A4_i^-)) \tag{1.109}$$

$$= \prod_i^N \sigma(s_i(x_i+\alpha_i x_{i+1}-t_i+b_i))(1-\sigma(s_i'(x_i+\alpha_i x_{i+1}-t_i-b_i))),$$

gdzie $\alpha_1,\ldots,\alpha_{N-1}$ definiują obrót, przyjmuje się iż $x_{N+1}=0$ i $\alpha_N=0$. Kontury stałych wartości tej funkcji mogą być obrócone, funkcja może być lokalna lub semi-lokalna, jak i niesymetryczna w poszczególnych wymiarach. Liczba parametrów adaptacyjnych wynosi $5N$ i tym samym stanowi silną alternatywę dla funkcji *SBi* (1.102).



Rysunek 1.35: Funkcje bicentralne z rotacją i niezależnymi skosami (1.109).

Bardzo ważną i zarazem pożyteczną cechą funkcji bicentralnych jest separowalność. Typowe sigmoidalne funkcje transferu nie są separowalne, a pośród typowych funkcji radialnych tylko funkcja Gaussa jest funkcją separowalną. Funkcje bicentralne, choć są skonstruowane za pomocą funkcji sigmoidalnych, są separowalne dzięki iloczynowi par sigmoid po wszystkich wymiarach wejściowych. Dzięki temu zawsze można dokonać opuszczenia pewnego wymiaru czy też kilku wymiarów i proces analizy danych modelu może być z powodzeniem kontynuowany. Właściwości te można wykorzystywać gdy mamy do czynienia z niepełnymi (brakującymi) danymi, wyciąganiem reguł logicznych z sieci neuronowych czy implementacją pamięci asocjacyjnej za pomocą sieci neuronowych [65, 1, 59, 60].

1.4.8. Hierarchia funkcji transferu pod względem ich elastyczności

Niewątpliwie próba poukładania funkcji transferu w ciąg, w którym mielibyśmy zachowaną relację mniejszości pod względem *elastyczności* danych funkcji, czyli ich możliwości adaptacyjnych funkcji poprzez parametry wolne nie jest w ogóle możliwe, ale z pewnością jest kuszące, aby przynajmniej wyodrębnić grupy funkcji o zbliżonych możliwościach aproksymacyjnych.

Z pewnością do pierwszej grupy najuboższych funkcji należą funkcje schodkowe. Poprzez ich schodkowość ich możliwości są znacznie ograniczone w porównaniu z praktycznie każdymi przedstawionymi powyżej funkcjami.

Do drugiej grupy funkcji można by zaliczyć funkcje sigmoidalne, aproksymacje funkcji sigmoidalnych, jak również wszystkie funkcje radialne i różne aproksymacje funkcji gaussowskiej.

Następną grupę mogą tworzyć funkcje, które są bardziej niezależne w poszczególnych wymiarach: funkcja gaussowska wielu zmiennych, sigmoidalna wielu zmiennych, i im podobne, ponadto funkcje wstęgowe, funkcja Lorentza i funkcja okienkująca.

Czwartą grupę tworzą funkcje uniwersalne: funkcje bicentralne, stożkowe, kołowe, ich aproksymacje.

Ostatnią grupę tworzą ogólniejsze funkcje uniwersalne do których można zaliczyć: rozszerzone funkcje bicentralne, rozszerzoną funkcje kołową i rozszerzoną funkcje stożkową.

W tabeli 1.1 zamieszczono powyższą klasyfikację jak i dodatkowe informacje wskazujące odpowiednie referencje numerów wzorów, informacje o funkcji aktywacji i funkcji wyjścia. Przez I jak i powyżej rozumie się aktywacje w postaci iloczynu skalarnego, I^+ oznacza że wektor $\mathbf{x}$ lub wektor wag został poddany dodatkowej transformacji, a I_i to $x_i w_i$. D jest odległością euklidesową; $D_i^2 = (x_i - t_i)/b_i$. G oznacza funkcje gaussowską, σ oznacza funkcje logistyczna. A w opisie funkcji wyjścia oznacza bieżącą aktywację.

W tabeli można zaobserwować grę aktywacji iloczynu skalarnego I z miara odległości D i ich różnych wariantów, a funkcjami gaussowskimi, sigmoidalny-

mi wspieranymi przez sumy i produkty różnego typu. Dla przykładu weźmy funkcję transferu logistyczną $I : \sigma$ (aktywacja – iloczyn skalarny, funkcja wyjścia – sigmoidalna) i funkcję gaussowską: $D : G$ (aktywacja – odległość Euklidesowa, funkcja wyjścia – funkcja gaussowska). Teraz łącząc je dostajemy opisaną w rozdziale funkcję stożkową: $I + D : \sigma$ (aktywacja – kombinacja iloczynu skalarnego i miary odległości, funkcja wyjścia – sigmoidalna).

1.4.9. Końcowe porównanie różnych funkcji transferu

Porównanie różnych funkcji transferu zostało przedstawione w tabeli 1.2.

W pierwszej kolumnie znajduje się nazwa funkcji transferu; druga kolumna pokazuje numer równania definiującego funkcje; trzecia kolumna pokazuje użytą funkcję aktywacji. Kolejna kolumna pokazuje liczbę adaptacyjnych parametrów dla d wymiarowej przestrzeni wejściowej. Dla funkcji wieloschodkowej k jest liczbą stopni funkcji (zazwyczaj nie podlegają one jednak adaptacji). W piątej kolumnie mamy informację, która określa czy funkcja transferu ma lokalny, czy nielokalny charakter (niektóre funkcje mogą być i lokalne i nielokalne, w zależności od stanu parametrów. Kolejna kolumna pokazuje typ(-y) parametrów adaptacyjnych: $\mathbf{w}$ – liniowe parametry wag, θ – progi, $\mathbf{t}$ – centra, $\mathbf{b}$ i b są parametrami odpowiadającymi dyspersją lub parametrami odpowiadającymi za skalowanie cech, $\mathbf{s}$ określaj skosy, $\mathbf{R}$ parametry rotacji, o i O oznaczają inne parametry adaptacyjne. Ostatnie dwie kolumny opisują, czy funkcja jest separowalna (Y), symetryczna (S), asymetryczna (A) lub niesymetryczna (N).

Zaprezentowane funkcje transferu, wyjścia i aktywacji obrazują szeroką gamę przeróżnych właściwości i możliwości, które niewątpliwie są bardzo przydatne w rozwiązywaniu złożonych problemów.

Innym sposobem zwiększenia możliwości modeli adaptacyjnych jest dodanie nowych cech wejściowych, stworzonych poprzez różne transformacje nieliniowe cech pierwotnych. Więcej informacji można znaleźć w pracy Ducha i Jankowskiego [70], jak i [201, 193, 251, 249, 227].

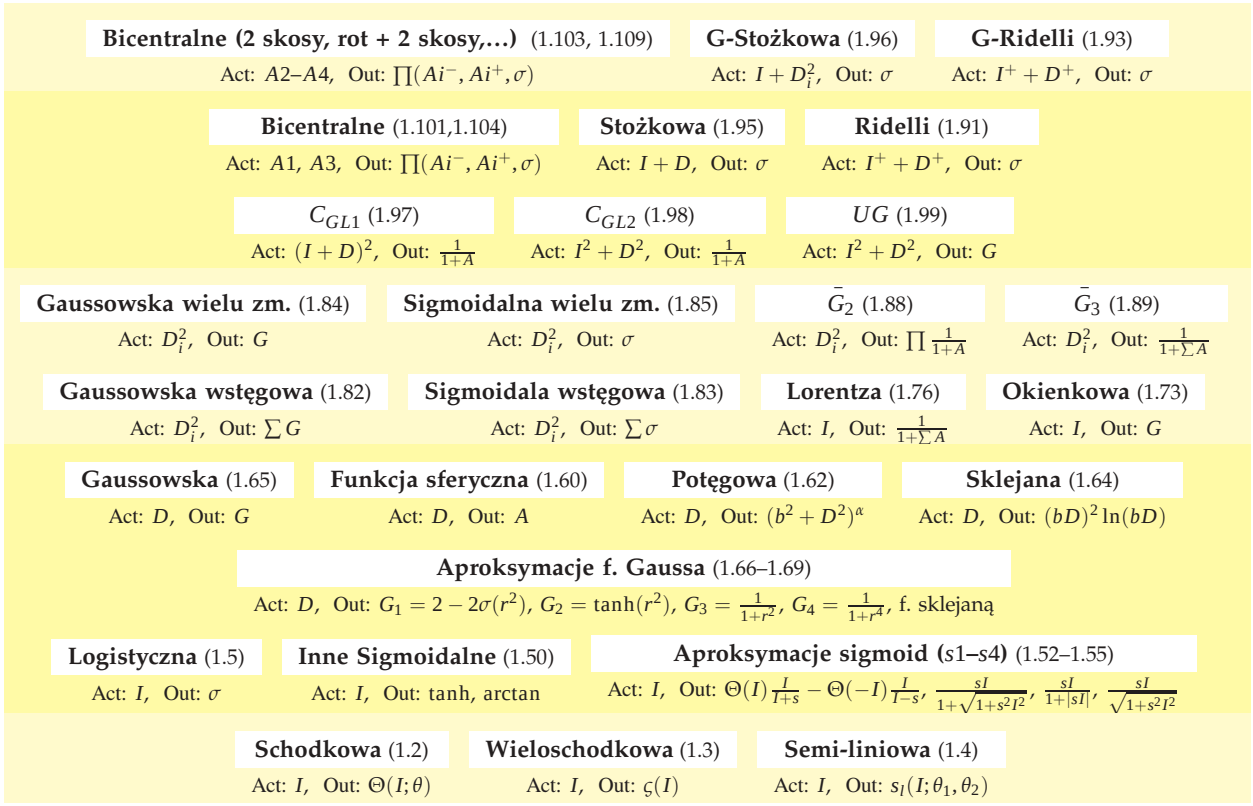


Tabela 1.1: Hierarchie elastyczności funkcji transferu. Każdy z szarych wierszy opisuje grupy funkcji o zbliżonych możliwościach pod względem ich użyteczności. Wiersz górny opisuje funkcje o największych możliwościach, natomiast dolny o najmniejszych możliwościach. Opis zawiera także informacje na temat funkcji aktywacji i funkcji wyjścia.

Funkcja	Nr równania	Aktywacja	L. parametrów	Lokalna nielokalna	Typy Parametrów	Separowalność	Symetria
Funkcje schodkowe							
Schodkowa	(1.2)	I	$d+1$	NL	$\mathbf{w}, \theta$		A
Wieloschodkowa	(1.3)	I	$d+k$	NL	$\mathbf{w}, \Theta$		A
Semi-liniowa	(1.4)	I	$d+2$	NL	$\mathbf{w}, \theta$		A
Funkcje sigmoidalne							
Logistyczna	(1.5)	I	$d+1$	NL	$\mathbf{w}, \theta$		A
tanh	(1.50)	I	$d+1$	NL	$\mathbf{w}, \theta$		A
arctan		I	$d+1$	NL	$\mathbf{w}, \theta$		A
Aproksymacje funkcji sigmoidalnej							
s_1	(1.52)	I	$d+1$	NL	$\mathbf{w}, \theta$		A
s_2	(1.53)	I	$d+1$	NL	$\mathbf{w}, \theta$		A
s_3	(1.54)	I	$d+1$	NL	$\mathbf{w}, \theta$		A
s_4	(1.55)	I	$d+1$	NL	$\mathbf{w}, \theta$		A
Radialne funkcje bazowe							
Sferyczna	(1.60)	D	d	NL	$\mathbf{t}, b$		S
Potęgowa	(1.62)	D	$d+2$	L+NL	$\mathbf{t}, b, o$		S
Sklejana	(1.64)	D	$d+1$	NL	$\mathbf{t}, b$		S
Gaussa	(1.65)	D	$d+1$	L	$\mathbf{t}, b$	Y	S
Aproksymacje funkcji Gaussa							
G_1	(1.66)	D	$d+1$	L	$\mathbf{t}, b$		S
$G_2 = \tanh(r^2)$	(1.67)	D	$d+1$	L	$\mathbf{t}, b$		S
G_3	(1.68)	D	$d+1$	L	$\mathbf{t}, b$		S
G_4	(1.69)	D	$d+1$	L	$\mathbf{t}, b$		S
RCBSpline	(1.70)	D	$d+1$	L	$\mathbf{t}, b$		S
RQBSpline	(1.71)	D	$d+1$	L	$\mathbf{t}, b$		S
Funkcje o gęstościach elipsoidalnych							
Gaussa wiel. zm.	(1.84)	D	$2d$	L	$\mathbf{t}, \mathbf{b}$	Y	S

Tabela 1.2: Porównanie funkcji transferu. Symbole użyte w tabeli zostały wyjaśnione w tekście (patrz str. 77).

Sigmoidalna w. zm.	(1.85)	D	$2d$	L	$\mathbf{t}, \mathbf{b}$		S
$\bar{G}_2$	(1.88)	D_i	$2d$	L	$\mathbf{t}, \mathbf{b}$	Y	S
$\bar{G}_3$	(1.89)	D	$2d$	L	$\mathbf{t}, \mathbf{b}$		S
Funkcje wstęgowe							
Wstęgowa Gaussa	(1.82)	D_i	$3d$	L	$\mathbf{t}, \mathbf{b}, O$	Y	S
Wstęgowa sigmoidalna	(1.83)	D_i	$3d$	L	$\mathbf{t}, \mathbf{b}, O$	Y	S
Funkcje atypowe							
$W(\mathbf{x}; \mathbf{w})$	(1.73)	I	$d+1$	NL	$\mathbf{w}, \theta$		S
Lorentza	(1.76)	I	$d+1$	NL	$\mathbf{t}, \mathbf{s}$		S
Iloczyn tensorowy	(1.77)	D_i	$2d+1$	NL	$\mathbf{w}, \theta$		N
G_R	(1.90)	D	$2d$	NL	$\mathbf{t}, \mathbf{b}$		N
Uniwersalne funkcje transferu							
Ridelli	(1.91)	A_R	$d+2$	L+NL	$\mathbf{w}, \theta$		S+N
G-Ridelli	(1.93)	A_{GR}	$2d+2$	L+NL	$\mathbf{w}, \mathbf{r}, \theta$		S+N
Stożkowa	(1.95)	A_C	$2d+2$	L+NL	$\mathbf{t}, \mathbf{w}, \theta, o$		S+N
G-Stożkowa	(1.96)	A_{GC}	$3d+2$	L+NL	$\mathbf{t}, \mathbf{w}, \mathbf{b}, \theta, o$		S+N
C_{GL1}	(1.97)	A_{GL1}	$2d+3$	L+NL	$\mathbf{t}, \mathbf{w}, \theta, o$		S+N
C_{GL2}	(1.98)	A_{GL2}	$2d+4$	L+NL	$\mathbf{t}, \mathbf{w}, \theta, o$		S+N
UG	(1.99)	$I^2 + D^2$	$2d+1$	L+NL	$\mathbf{t}, \mathbf{w}, \theta, b$		S+N
Bicentralne funkcje transferu							
Bicentralna	(1.101)	$A1$	$3d$	L	$\mathbf{t}, \mathbf{b}, \mathbf{s}$	Y	S
Semi-bicentralna	(1.102)	$A2$	$5d$	L+NL	$\mathbf{t}, \mathbf{b}, \mathbf{s}, O$	Y	S+N
Bicentral z 2 sk.	(1.103)	$A2$	$4d$	L+NL	$\mathbf{t}, \mathbf{b}, \mathbf{s}$	Y	S+N
Bicentral z rot.	(1.104)	$A3$	$4d-1$	L	$\mathbf{t}, \mathbf{b}, \mathbf{s}, R$	Y/N	S
Semi-bic. z rot.	(1.105)	$A3$	$6d-1$	L+NL	$\mathbf{t}, \mathbf{b}, \mathbf{s}, R, O$	Y/N	S+N
C_K rotation	(1.107)	$A1$, $A1(\mathbf{kx})$	$4d$	L	$\mathbf{t}, \mathbf{b}, \mathbf{s}, R$	Y/N	S
C_{PK} rotation	(1.108)	$A1$, $A1(\mathbf{kx})$	$4d$	L	$\mathbf{t}, \mathbf{b}, \mathbf{s}, R$	Y/N	S
Bicentral, rot. 2 sk.	(1.109)	$A4$	$5d-1$	L+NL	$\mathbf{t}, \mathbf{b}, \mathbf{s}, R$	Y/N	S+N

Tabela 1.3: Porównanie funkcji transferu cd.

Rozdział 2

Sieci z radialnymi funkcjami bazowymi (RBF)

W bieżącym rozdziale będzie można prześledzić wiele przeróżnych aspektów dotyczących sieci z radialnymi funkcjami bazowymi (RBF), *ang. radial basis function networks*. Przedstawione zostaną różne metody metody inicjalizacji i uczenia z nadzorem i bez nadzoru. Omówione zostaną własności poszczególnych algorytmów. Należy także wspomnieć o szerokiej gamie zastosowań sieci RBF. Jednym z pierwszych ciekawych zastosowań było użycie sieci RBF do przetwarzania obrazów [207]. Szczególnie ciekawe są ostatnie rezultaty Poggia [206]. Ciekawe są rezultaty w rozpoznawaniu wzorców [98]. Interesujące są również rezultaty uzyskane w rozpoznawaniu i przetwarzaniu mowy [194, 148]. Dobre rezultaty uzyskano też stosując sieci RBF do analizy szeregów czasowych (w tym również finansowych) [150, 188, 27, 119, 128, 129]. Jeszcze inne przykłady obejmują zastosowanie sieci RBF do różnych zagadnień medycznych (patrz rozdział 7). Interesujące było także zastosowanie sieci RBF do analizy chromosomów myszy [191]. Całą gamą zastosowań może poszczycić się model Support Vector Machines (SVM), który można traktować jako szczególny przypadek sieci RBF, jak będzie można zobaczyć w rozdziale jemu poświęconym (3).

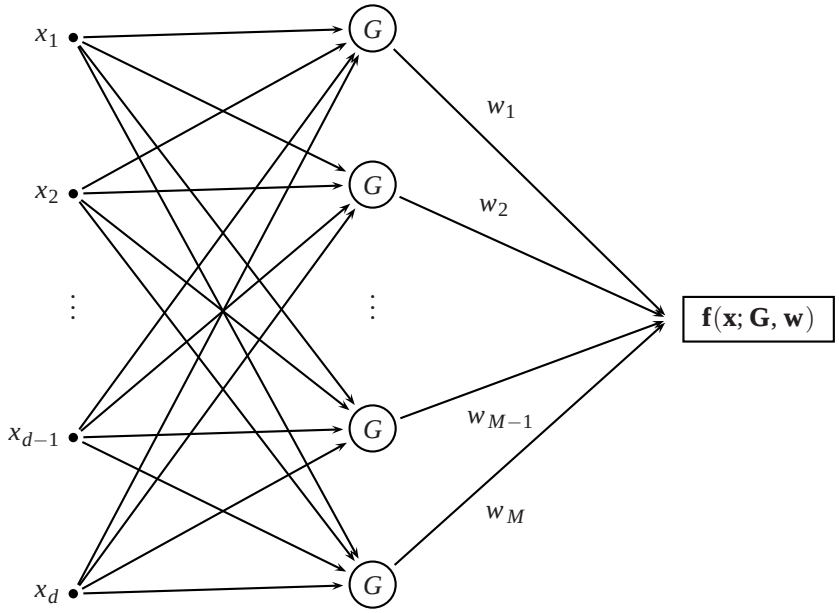
2.1. Sieci z radialnymi funkcjami bazowymi i regularyzacją

Sztuczne sieci neuronowe są wykorzystywane do rozwiązywania wielu różnych problemów takich, jak klasyfikacja, aproksymacja, rozpoznawanie wzorców, przetwarzania sygnałów, przewidywania szeregów czasowych, pozyskiwania reguł logicznych, realizacji pamięci asocjacyjnych, samoorganizacji danych, itp. Większość z tych problemów modele sieci neuronowych rozwiązują poprzez

uczenie nieznanego odwzorowania pomiędzy przestrzenią wejściową i wyjściową wektorów uczących z pewnego zbioru $\mathcal{S} = \{\langle \mathbf{x}_1, y_1 \rangle, \ldots, \langle \mathbf{x}_n, y_n \rangle\}$, gdzie $\langle \mathbf{x}_i, y_i \rangle$ jest wzorcem uczącym, który jest parą wartości wektora przestrzeni wejściowej i wartości z przestrzeni wyjściowej ($\mathbf{x}_i \in \mathcal{R}^N$, $y_i \in \mathcal{R}$). Tak przedstawione odwzorowanie $F(\cdot)$ można zapisać w poniższej formie:

$$F(\mathbf{x}_i) = y_i + \eta, \qquad i = 1, \ldots, n, \tag{2.1}$$

η jest szumem z zerową wartością oczekiwaną i wariancją równą σ_{ns}^2.



Rysunek 2.1: Sieć z radialnymi funkcjami bazowymi.

Pierwotnie sieci o radialnych funkcjach bazowych były przeznaczone głównie do problemów aproksymacji w przestrzeniach wielowymiarowych [211, 76, 84, 108, 209]. Tym samym proces uczenia rekonstruował powierzchnie nieznanego odwzorowania $F(\cdot)$ za pomocą wzorców uczących ze zbioru $\mathcal{S}$, poprzez adaptacje wolnych parametrów modelu, którymi na początku były tylko wagi połączeń warstwy ukrytej z warstwą wyjściową. Funkcje jaką realizuje sieć o radialnych funkcjach bazowych (rys. 2.1) można zapisać poniższym wzorem:

$$f(\mathbf{x}; \mathbf{w}, \mathbf{p}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x}, \mathbf{p}_i), \tag{2.2}$$

M określa liczbę neuronów warstwy ukrytej, a $G_i(\mathbf{x}, \mathbf{p}_i)$ to jedna z radialnych funkcji bazowych jako funkcja transferu i-tego neuronu warstwy ukrytej, wektory $\mathbf{p}_i$ to parametry adaptowalne i-tego neuronów warstwy ukrytej, takie jak

centrum (położenia i-tego neuronu) rozmycie i inne, zależnie od wyboru funkcji $G_i(\cdot)$. Najpopularniejszymi funkcjami radialnymi są: funkcja gaussowska, funkcja sferyczna, funkcje potęgowe i sklejane (rys. 1.10, 1.8, 1.9, 1.7):

$$h_1(\mathbf{x};\mathbf{t},b) = e^{-||\mathbf{x}-\mathbf{t}||^2/b^2}, \tag{2.3}$$
$$h_2(\mathbf{x};\mathbf{t},b) = ||\mathbf{x}-\mathbf{t}||/b^2, \tag{2.4}$$
$$h_3(\mathbf{x};\mathbf{t},b,\alpha) = (b^2+||\mathbf{x}-\mathbf{t}||^2)^{-\alpha}, \quad \alpha>0, \tag{2.5}$$
$$h_4(\mathbf{x};\mathbf{t},b,\beta) = (b^2+||\mathbf{x}-\mathbf{t}||^2)^{\beta}, \quad 0<\beta<1, \tag{2.6}$$
$$h_5(\mathbf{x};\mathbf{t},b) = (b||\mathbf{x}-\mathbf{t}||)^2\ln(b||\mathbf{x}-\mathbf{t}||). \tag{2.7}$$

Sieci wykorzystujące radialne funkcje bazowe są również aproksymatorem uniwersalnym [114, 203]. Więcej informacji o funkcjach RBF można znaleźć w poprzednim rozdziale, a szczególnie w podrozdziale 1.3.2.

Funkcje radialne, jak widać z powyższych wzorów, zawsze korzystają z normy $||\mathbf{x}-\mathbf{t}||$. W większości przypadków jest to norma euklidesowa:

$$f(\mathbf{x};\mathbf{w},\mathbf{t}) = \sum_{i=1}^{M} w_i G_i(||\mathbf{x}-\mathbf{t}_i||). \tag{2.8}$$

Przyjmując, że liczba wzorców uczących n jest równa liczbie neuronów w warstwie ukrytej M, otrzymujemy:

$$\begin{bmatrix} G_{11} & G_{12} & \cdots & G_{1n} \\ G_{21} & G_{22} & \cdots & G_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ G_{n1} & G_{n2} & \cdots & G_{nn} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \tag{2.9}$$

G_{ij} jest równe wartości funkcji G_i w punkcie $\mathbf{x}_j$, $G_{ij} = G_i(||\mathbf{x}_j - \mathbf{t}_i||)$.

Przyjmując, że $\mathbf{G}$, to macierz złożona z wartości G_{ij} i

$$\mathbf{w} = [w_1,\ldots,w_n]^T, \tag{2.10}$$
$$\mathbf{y} = [y_1,\ldots,y_n]^T \tag{2.11}$$

otrzymujemy

$$\mathbf{G}\mathbf{w} = \mathbf{y}. \tag{2.12}$$

Twierdzenie Light'a [118, 167] mówi, że dla klasy radialnych funkcji bazowych takich, iż jeśli tylko wektory $\mathbf{x}_i$ są różne, to macierz $\mathbf{G}$ jest dodatnio określona i możemy wtedy wyznaczyć wektor wag $\mathbf{w}$, wymnażając lewostronnie równanie (2.12) przez $\mathbf{G}^{-1}$

$$\mathbf{w} = \mathbf{G}^{-1}\mathbf{y}. \tag{2.13}$$

Twierdzenie Light'a można stosować między innymi do funkcji gaussowskiej (1.65,2.3) i odwrotnej funkcji potęgowej z $\alpha=1$, $h_3(\mathbf{x};\mathbf{t},b,1)$ (1.62,2.5).

W praktyce jednak teoretyczna możliwość rozwiązania równania (2.12) nie jest wystarczająca, ponieważ macierz **G** może być niedostatecznie dodatnio określona (podobne wzorce uczące), a sieć, która ma tyle samo radialnych funkcji bazowych co wzorców uczących, rzadko może okazać się dobrym rozwiązaniem, ponieważ zazwyczaj będzie to prowadziło do niesatysfakcjonującej generalizacji. Sieć będzie uczyła się nie tylko rozpoznawania wzorców lecz również szumu, który znajdzie się w danych (*ang. overfitting*) [27]. Dlatego też należy wprowadzić czynnik regularyzacyjny do macierzy **G**, $\mathbf{G} + \lambda\mathbf{I}$, który wynika z metody regularyzacji Tikhonova.

Tikhonov zaproponował metodę regularyzacji, która polega na przejściu od standardowej funkcji błędu:

$$E_0(f) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2, \tag{2.14}$$

do funkcji rozszerzonej o dodatkowy człon w celu stabilizacji rozwiązania. Stabilizację osiąga się poprzez osadzenie w modelu adaptacyjnym dodatkowej informacji *a priori*. Dla aproksymacji może to być założenie o gładkości funkcji, która będzie aproksymowała nieznane odwzorowanie. W ten sposób zadanie stanie się dobrze określone i można uniknąć przewymiarowania problemu określonego poprzez układ równań (2.12) [209].

Aby skorzystać z teorii regularyzacji trzeba będzie przedefiniować funkcje oceny błędu kwadratowego procesu minimalizacji dołączając człon regularyzacyjny

$$E_r(f) = \frac{1}{2}||\mathbf{P}f||^2, \tag{2.15}$$

co razem daje

$$\begin{aligned} E(f) &= E_0(f) + \lambda E_r(f) \\ &= \frac{1}{2}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2 + \lambda\frac{1}{2}||\mathbf{P}f||^2, \end{aligned} \tag{2.16}$$

P jest pewnym liniowym operatorem różniczkowym, który powinien być zależny od problemu f. Operator **P** będzie pełnił rolę stabilizatora rozwiązania f (tym samym wygładzając je).

Aby wyznaczyć minimum $E(f)$ Poggio i Girosi posługują się różniczką Fréchet'a funkcji błędu, przyrównując ją do zera

$$\partial E(f,h) = \partial E_0(f,h) + \lambda\partial E_r(f,h), \tag{2.17}$$

gdzie $h(\mathbf{x})$ jest pewną funkcją wektora **x**. To z kolei prowadzi do równania

$$\mathbf{P} * \mathbf{P}f(\mathbf{x}) = \frac{1}{\lambda}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))\ \delta(\mathbf{x} - \mathbf{x}_i). \tag{2.18}$$

Dzięki temu i kilku innym własnością Poggio i Girosi dostają następujący rezultat

$$f(\mathbf{x}) = \frac{1}{\lambda}\sum_{i=1}^{n}[y_i - f(\mathbf{x}_i)]G_r(\mathbf{x};\mathbf{x}_i). \tag{2.19}$$

Równanie (2.19) wyznacza jako rozwiązanie $f(\mathbf{x})$ problemu regularyzacji liniową superpozycję n funkcji Greena, położonych w punktach $\mathbf{x}_i$ z wagami $w_i = (1/\lambda)[y_i - f(\mathbf{x}_i)]$. Mamy więc

$$\mathbf{w} = \frac{1}{\lambda}(\mathbf{y} - \mathbf{f}), \tag{2.20}$$

$$\mathbf{f} = \mathbf{G}\mathbf{w}, \tag{2.21}$$

gdzie $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)]^T$, a $\mathbf{G}$ jest macierzą Greena:

$$\mathbf{G} = \begin{bmatrix} G(\mathbf{x}_1\mathbf{x}_1) & G(\mathbf{x}_1\mathbf{x}_2) & \ldots & G(\mathbf{x}_1\mathbf{x}_n) \\ G(\mathbf{x}_2\mathbf{x}_1) & G(\mathbf{x}_2\mathbf{x}_2) & \ldots & G(\mathbf{x}_2\mathbf{x}_n) \\ \vdots & \vdots & & \vdots \\ G(\mathbf{x}_n\mathbf{x}_1) & G(\mathbf{x}_n\mathbf{x}_2) & \ldots & G(\mathbf{x}_n\mathbf{x}_n) \end{bmatrix}. \tag{2.22}$$

Stąd otrzymujemy rozwiązanie

$$\mathbf{w} = (\mathbf{G} + \lambda\mathbf{I})^{-1}\mathbf{y}, \tag{2.23}$$

gdzie $\mathbf{I}$ jest macierzą jednostkową.

Poggio i Girosi [209] udowodnili, że sieć neuronowa oparta o funkcje Greena, jako bazowe funkcje radialne, jest uniwersalnym aproksymatorem, czyli może aproksymować dowolną funkcję ciągłą wielu zmiennych na wypukłym podzbiorze w $\mathcal{R}^{\mathcal{N}}$, używając wystarczającej liczby neuronów w warstwie ukrytej. Udowodnili też, że dla dowolnej nieliniowej funkcji F istnieją takie współczynniki w_i, że aproksymują lepiej niż każde inne, co oznacza, iż tak zbudowana sieć ma własność najlepszego aproksymatora. Pokazano też, że rozwiązanie sieci z taką regularyzacją jest optymalne, czyli minimalizuje funkcjonał, który mierzy, jak daleko rozwiązanie jest od danych treningowych.

2.2. Uogólniona sieć z radialnymi funkcjami bazowymi (GRBF)

Powyższa sieć RBF zdaje się jednak w pełni na możliwości regularyzacji i zużywa tyle neuronów ile jest wzorców uczących, co niestety nie zawsze gwarantuje dobre rozwiązanie. Osiągnięcie wysokiej jakości aproksymacji (przy pewnych ograniczeniach regularyzacji) to nie to samo, co osiągnięcie wysokiego poziomu generalizacji, choć oczywiście własności te nie pozostają bez związku. Przez generalizację należy rozumieć zdolność modelu do dokonywania przekształcenia jak najbardziej zbliżonego do przekształcenia jakie dokonywał by model

optymalny dla danego zbioru uczącego opisującego dany problem. Tym samym kierunek zmian powinien iść w stronę uzyskania większego związku pomiędzy złożonością danych uczących, a liczbą węzłów warstwy ukrytej sieci RBF. Najprostszym sposobem jest oczywiście utworzenie sieci, w której liczba neuronów w warstwie ukrytej będzie znacznie mniejsza, niż liczba wzorców uczących. W pierwszym podejściu liczbę neuronów warstwy ukrytej można wyznaczyć na podstawie pewnej wiedzy a priori o danych uczących. Informacje o różnych podejściach do problemu wyboru liczby jak i położeń neuronów warstwy ukrytej będzie można znaleźć w dalszej części rozdziału.

Można więc napisać równanie sieci RBF

$$f(\mathbf{x};\mathbf{w},\mathbf{p}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x},\mathbf{p}_i), \tag{2.24}$$

zakładając, że liczba neuronów M jest istotnie mniejsza niż liczba wzorców uczących n. W związku z powyższym, należy przedefiniować funkcję błędu (2.14) na

$$\begin{aligned} E(f) &= \sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{M} w_j G(||\mathbf{x_i} - \mathbf{t}_j||)\right)^2 + \lambda||\mathbf{P}f||^2 \\ &= ||\mathbf{y} - \mathbf{G}\mathbf{w}||^2 + \lambda||\mathbf{P}f||^2, \end{aligned} \tag{2.25}$$

gdzie

$$\mathbf{y} = [y_1, y_2, \ldots, y_n]^T \tag{2.26}$$

$$\mathbf{G} = \begin{bmatrix} G(\mathbf{x}_1;\mathbf{t}_1) & G(\mathbf{x}_1;\mathbf{t}_2) & \cdots & G(\mathbf{x}_1;\mathbf{t}_M) \\ G(\mathbf{x}_2;\mathbf{t}_1) & G(\mathbf{x}_2;\mathbf{t}_2) & \cdots & G(\mathbf{x}_2;\mathbf{t}_M) \\ \vdots & \vdots & \ddots & \vdots \\ G(\mathbf{x}_n;\mathbf{t}_1) & G(\mathbf{x}_n;\mathbf{t}_2) & \cdots & G(\mathbf{x}_n;\mathbf{t}_M) \end{bmatrix} \tag{2.27}$$

$$\mathbf{w} = [w_1, w_2, \ldots, w_M]^T. \tag{2.28}$$

Tak zdefiniowana macierz $\mathbf{G}$ funkcji bazowych nie jest już kwadratowa. Ma wymiary n na M, choć sam wektor $\mathbf{y}$ nie zmienił rozmiarów. Wektor $\mathbf{w}$ ma rozmiar M, lecz M jest różne od n. Jak pokazują Poggio i Girosi [209] prawą stronę wyrażenia (2.25) można rozwinąć do

$$||\mathbf{P}f||^2 = \mathbf{w}^T\mathbf{G}_0\mathbf{w}, \tag{2.29}$$

gdzie macierz $\mathbf{G}_0$ jest macierzą kwadratową M na M, zdefiniowaną jako

$$\mathbf{G}_0 = \begin{bmatrix} G(\mathbf{t}_1;\mathbf{t}_1) & G(\mathbf{t}_1;\mathbf{t}_2) & \cdots & G(\mathbf{t}_1;\mathbf{t}_M) \\ G(\mathbf{t}_2;\mathbf{t}_1) & G(\mathbf{t}_2;\mathbf{t}_2) & \cdots & G(\mathbf{t}_2;\mathbf{t}_M) \\ \vdots & \vdots & \ddots & \vdots \\ G(\mathbf{t}_M;\mathbf{t}_1) & G(\mathbf{t}_M;\mathbf{t}_2) & \cdots & G(\mathbf{t}_M;\mathbf{t}_M) \end{bmatrix}. \tag{2.30}$$

Można powrócić do minimalizacji równania (2.25) i wyznaczyć pochodną

$$\begin{aligned} \partial E(f) &= \partial||\mathbf{y} - \mathbf{G}\mathbf{w}||^2 + \partial\lambda||\mathbf{P}f||^2 \\ &= \mathbf{G}^T(\mathbf{y} - \mathbf{G}\mathbf{w}) - \lambda\mathbf{G}_0\mathbf{w} \end{aligned} \tag{2.31}$$

i przyrównać ją do zera

$$\mathbf{G}^T\mathbf{y} - (\mathbf{G}^T\mathbf{G} - \lambda\mathbf{G}_0)\mathbf{w} = 0, \tag{2.32}$$

skąd osiągamy już szukany wektor wag $\mathbf{w}$

$$\mathbf{w} = (\mathbf{G}^T\mathbf{G} - \lambda\mathbf{G}_0)^{-1}\mathbf{G}^T\mathbf{y}. \tag{2.33}$$

Dla $\lambda = 0$ rozwiązaniem minimalizacji jest iloczyn macierzy pseudoodwrotnej z wektorem wartości oczekiwanych $\mathbf{y}$

$$\mathbf{w} = (\mathbf{G}^T\mathbf{G})^{-1}\mathbf{G}^T\mathbf{y}. \tag{2.34}$$

2.3. Metody inicjalizacji i uczenia bez nadzoru sieci typu RBF

W poprzedniej części zaprezentowano metody regularyzacji dla sieci z radialnymi funkcjami bazowymi. Pozostają jednak jeszcze pewne pytania. Na przykład, jak dokonać wyboru położeń funkcji bazowych w metodzie opisanej w podrozdziale 2.2, czy też jak dobrać parametry rozmyć dla funkcji bazowych (i być może wartości innych parametrów, zależnie od wyboru funkcji bazowych). W tym podrozdziale zostaną omówione problemy inicjalizacji i beznadzorowego uczenia sieci RBF i różnych innych sieci zbliżonych do sieci RBF.

Jak widać z równania (2.2), jak i rys. 2.1, warstwy sieci RBF spełniają różne role. Pierwsza warstwa lokalizuje węzły warstwy ukrytej w przestrzeni wejściowej, pokrywając ją w okolicach obszarów, w których występują dane. Zadaniem drugiej warstwy jest wyznaczenie takiej liniowej kombinacji aktywacji neuronów warstwy ukrytej, aby estymowała ona nieznane odwzorowanie $F(\cdot)$ równania (2.1). Stąd też płynie natura uczenia sieci z radialnymi funkcjami bazowymi, która przeplata uczenie z nadzorem i uczenie bez nadzoru. Inicjalizacja pierwszej warstwy odbywa się bez nadzoru (patrz poniższe metody), natomiast inicjalizacja drugiej warstwy, jak i sam proces uczenia przebiegają już z nadzorem (patrz kolejny podrozdział 2.4 i rozdział 4).

Ważną cechą sieci o radialnych funkcjach bazowych jest to, że wagi pierwszej warstwy mogą zostać wyznaczone za pomocą wiedzy a priori o samych danych wejściowych. Dzięki wykorzystaniu informacji o danych wejściowych proces uczenia nie będzie startował z losowego punktu przestrzeni wag i ma znacznie większe szanse na pomyślny przebieg procesu uczenia sieci.

2.3.1. Inicjalizacja położeń zbiorem wektorów uczących

Najprostszy sposób wyznaczenia liczby i położeń węzłów warstwy ukrytej został już opisany w podrozdziale 2.1, w którym warstwa ukryta sieci składa się z tylu węzłów, ile jest wektorów uczących, a ich położenia wyznaczają wektory uczące.

2.3.2. Inicjalizacja położeń poprzez podzbiór zbioru uczącego

Jednym z pierwszych pomysłów było wybranie pewnej liczby M wektorów uczących i zainicjowanie mini położeń węzłów ukrytych zgodnie z rozkładem danych [27]. Takie zainicjowanie położeń, choć wydaje się bardzo proste, daje naprawdę dobre rezultaty, dzięki skupianiu węzłów w obszarach o większej gęstości danych. Widać stąd, iż błędne dane spoza właściwych obszarów, mają statystycznie nieistotny wpływ na późniejszy proces uczenia sieci. Spotyka się też propozycje nieznacznego losowego zaburzenia tak ustalonych położeń węzłów ukrytych, aby zadanie minimalizacji było lepiej określone [130]. W przypadku gdy funkcjami transferu mają być funkcje Gaussa pozostaje jeszcze dobór parametrów rozmyć. David Lowe [172] zaproponował, by uzależnić rozmycie od liczby węzłów warstwy ukrytej i maksymalnej odległości pomiędzy nimi:

$$G(||\mathbf{x}-\mathbf{t}_i||^2)=\exp\left(-\frac{M}{d^2}||\mathbf{x}-\mathbf{t}_i||^2\right),\quad i=1,2,\ldots,M, \tag{2.35}$$

M określa liczbę neuronów warstwy ukrytej, a d jest maksymalną odległością pomiędzy dowolną parą centrów funkcji bazowych. Tym samym *szerokość* każdego z neuronów to

$$\frac{d}{\sqrt{2M}}. \tag{2.36}$$

2.3.3. Inicjalizacja położeń metodą klasteryzacji k-średnich

Oczywiście nie ma powodu, aby prawdą było, iż losowe wybieranie wektorów wejściowych na położenia centrów, powiedzmy funkcji gaussowskich, neuronów ukrytych dla sieci RBF miało być optymalne. Fakt ten przyczynił się do powstania szeregu różnych, mniej i bardziej popularnych, sposobów inicjalizacji położeń centrów neuronów ukrytych. Jedną z najszerzej stosowanych metod jest metoda klasteryzacji k-średnich [74]. Jako pierwsi metodę k-średnich do sieci RBF użyli Moody i Darken [188]. Metoda ta stara się umiejscowić docelowo położenia centrów neuronów ukrytych w obszarach przestrzeni, w których znajdują się najbardziej liczne dane.

Metoda k-średnich cel ten stara się osiągnąć poprzez minimalizację następującej funkcji

$$kmean_{err}=\sum_{i=1}^{M}\sum_{j\in\mathcal{S}_i}||\mathbf{x}_j-\mathbf{t}_i||^2, \tag{2.37}$$

gdzie $\mathcal{S}_i$ to zbiór indeksów wektorów przynależnych do klastra i, a $\mathbf{t}_i$ stanowi centrum i-tego klastra.

W pierwszym etapie następuje ustalenie liczby klastrów i tym samym liczby neuronów M w warstwie ukrytej. Następnie, w losowy sposób wybieramy wstępne położenia klastrów spośród wektorów wejściowych. W procesie iteracyjnym dla każdego wektora $\mathbf{x}_j$ następuje uaktualnienie położenia najbliższego z centrów klastrów $\mathbf{t}_i$:

$$\Delta \mathbf{t}_i = \eta(\mathbf{x}_j - \mathbf{t}_i), \tag{2.38}$$

η jest parametrem określającym szybkość uczenia.

Znana jest też wersja *off-line* metody k-średnich [13]. Pierwsza z różnic między tą wersją, a opisaną wyżej polega na tym, iż centra początkowo wybierane są w losowych położeniach w przestrzeni wejściowej i następuje przypisanie każdego z wektorów do najbliższego klastra. Z kolei położenia centrów przeliczane są co epokę (po prezentacji całego zbioru uczącego):

$$\mathbf{t}_i = \frac{1}{|\mathcal{S}_i|} \sum_{j \in \mathcal{S}_i} \mathbf{x}_j, \tag{2.39}$$

gdzie $\mathcal{S}_i$ to zbiór indeksów wektorów przynależnych do klastra i, a $|\mathcal{S}_i|$ to liczba przynależnych wektorów do klastra i. Następnie dokonuje się powtórnego przypisania wszystkich wektorów do najbliższych klastrów.

Najczęściej stosowaną wersją algorytmu k-średnich jest pierwsza z podanych, choć oczywiście obie wersje algorytmu nie gwarantują optymalnego rozwiązania minimalizacji równania 2.37. Przy złych warunkach startowych, algorytm może nie umieścić żadnego klastra (centrum) w miejscach, które mogą być istotne, a w których danych jest zbyt mało (rzadko opisują daną część przestrzeni wejściowej). Zapobiec (choć raczej częściowo) takim problemom można poprzez uczynienie parametru η zmiennym w czasie procesu minimalizacji:

$$\eta = \frac{\eta_0}{1 + \frac{i}{T}}, \tag{2.40}$$

η_0 jest wartością początkową procesu, natomiast w kolejnych iteracjach i wraz z *a priori* wyznaczonym współczynnikiem T (dopasowanym dla konkretnych danych) wartość η będzie zmniejszana, najpierw niemal nieznacznie, a później coraz szybciej.

Metoda k-średnich jest też pewnym szczególnym przypadkiem metody mieszania modeli gaussowskich (*ang. Gaussian mixture models*), w której używa się do optymalizacji algorytmu EM (*ang. expectation maximization*) [13]. Również i algorytm samoorganizujących się map topograficznych Kohonena (SOFM) [157, 158] jest ogólnym algorytmem uczenia, dla którego metoda k-średnich jest szczególnym przypadkiem.

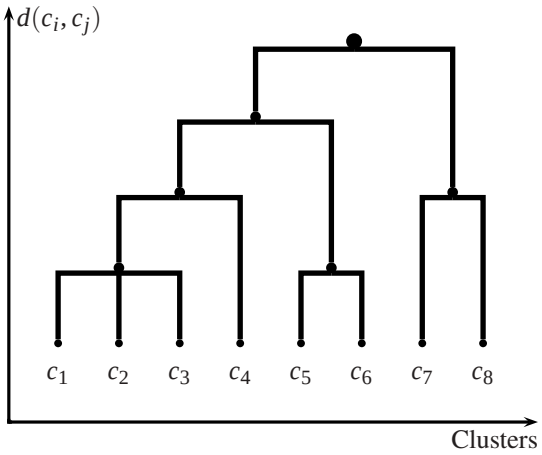
Alternatywną metodą klasteryzacji dla metody k-średnich może też być metoda klasteryzacji maksymalnej entropi [29] przy założeniu, iż przypisania do klastrów mają rozkład Gibbs'a.

2.3.4. Inicjalizacja za pomocą metody k najbliższych sąsiadów

Ta metoda również jest pewną odmianą metod samoorganizacji i stara się wyznaczyć położenia centrów funkcji radialnych tak, by znalazły się one w miejscach gdzie znajdują się najbardziej reprezentatywne wektory danych.

Do tego celu Moody i Darken [188] wykorzystali, w nieco niestandardowy sposób, metodę k najbliższych sąsiadów (kNN) [74]. Jako centra zostają wybrane te wektory wejściowe, które najczęściej są wybierane jako k najbliższe. Ta metoda, podobnie jak metoda k-średnich, pozwala wybrać położenia centrów neuronów w bardziej deterministyczny sposób, niż losowy wybór k centrów.

2.3.5. Konstruowanie klastrów za pomocą dendrogramów



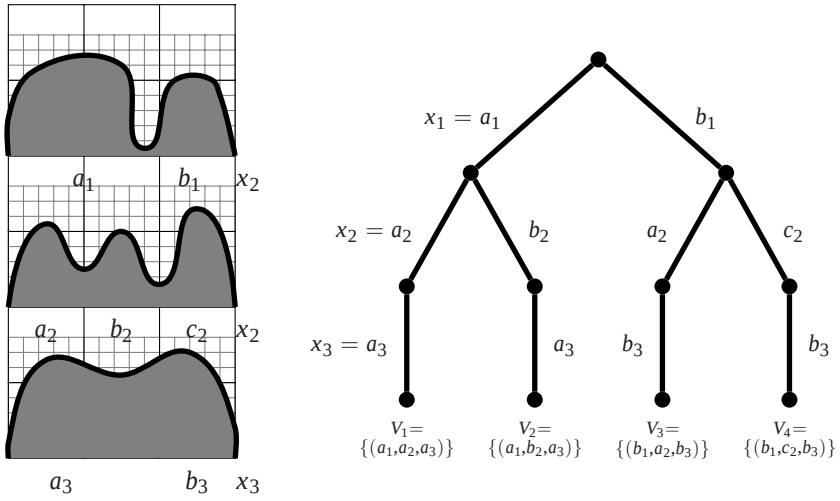
Rysunek 2.2: Dendrogramy.

Inną ciekawą metodą inicjalizacji położeń centrów, jak i ich rozmyć, które w tym przypadku mogą być różne dla różnych wymiarów, jest metoda dendrogramów. Dzięki tej metodzie można uzyskać ciekawsze rezultaty niż innymi metodami inicjalizacji [62]. Początkowo każdy z wektorów treningowych tworzy odrębny klaster. Po czym, w procesie iteracyjnym następuje łączenie najbliższych klastrów na podstawie pewnej miary odległości (patrz rys. 2.2). Taka procedura jest powtarzana do momentu aż liczba powstałych klastrów jest wystarczająco mała lub najmniejsza odległość w pewnej iteracji okaże się za duża, aby dokonywać dalszych połączeń.

Takie postępowanie wymaga jednak wstępnie wyznaczenia tablicy odległości pomiędzy każdą parą wektorów treningowych ($N(N-1)$ odległości), a następnie w każdej iteracji. Po dokonaniu połączenia odległości pomiędzy połączony-

mi klastrami a innymi klastrami stają się niepotrzebne. Z kolei należy wyznaczyć odległości pomiędzy nowym klastrem i pozostałymi klastrami i tym samym tablica odległości staje się aktualna. Bardzo ważną cechą tej metody jest to, że możemy użyć dowolnej funkcji odległości. Funkcja odległości może być zdefiniowana nie tylko jako odległość pomiędzy centrami klastrów, ale na przykład, jako minimalna odległość pomiędzy brzegami dwóch rozpatrywanych klastrów. Należy zwrócić uwagę, że podczas łączenia klastrów dokonuje się łączenia obszarów klastrów, co również może być różnie realizowane i jest bardzo zależne od przyjętej miary odległości i wstępnej obróbki danych.

2.3.6. Inicjalizacja za pomocą histogramów i drzew decyzyjnych



Rysunek 2.3: Histogramy.

Metoda dendrogramów jest, pod pewnym względem, odwrotnością metod opartych o analizę histogramów lub drzew klasyfikacyjnych (zwanych też drzewami decyzji). Odwrotność ta polega na tym, iż w metodzie dendrogramów opis danych jest z iteracji na iterację coraz ogólniejszy, natomiast drzewa klasyfikacyjne i metody oparte o histogramy lub inne podziały poszczególnych wymiarów, coraz bardziej uściślają opis danych uczących.

Pierwszym etapem tej metody jest wyznaczenie punktów w oparciu o metodę histogramów lub pewien arbitralny podział poszczególnych wymiarów. W przypadku histogramów taka dyskretyzacja uwidacznia gęstości występowania poszczególnych klas lub klastrów danych w wyznaczonych przedziałach, które tworzą klastry w jednowymiarowej przestrzeni. Histogramy informują o punktach w danym wymiarze, w których dane zmieniają przynależność do klasy lub klastra, czyli o potencjalnych granicach klas lub klastrów.

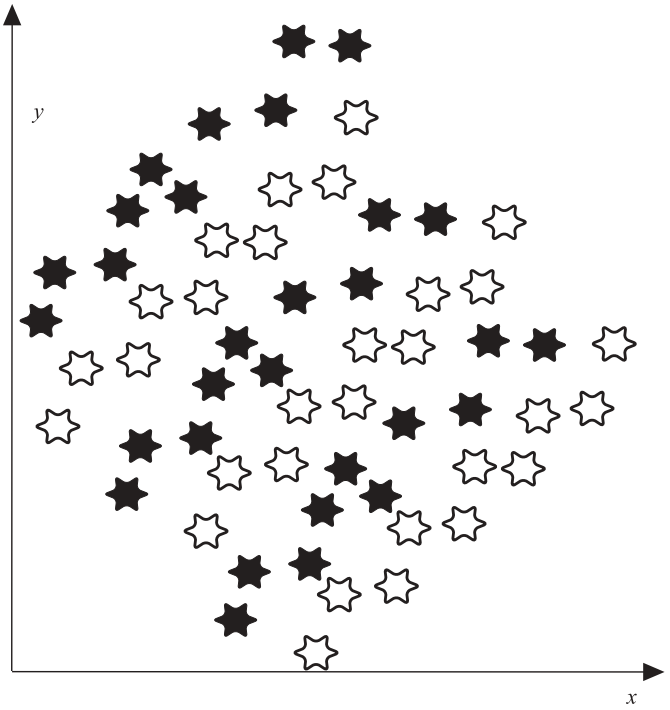
Tak wyznaczone punkty definiują nam granice wstępnych klastrów w jednowymiarowej podprzestrzeni przestrzeni d wymiarowej danych (patrz rys. 2.3). Każdy z jednowymiarowych klastrów jest rzutem pewnej liczby niezależnych klastrów w d wymiarowej przestrzeni. Każdy z klastrów w d wymiarowej przestrzeni przynależy do dokładnie jednego klastra w jednowymiarowej przestrzeni (jest zdefiniowany poprzez ciąg klastrów w poszczególnych wymiarach). W ten sposób, z histogramów powstaje drzewo klasyfikacyjne po uszeregowaniu wymiarów w ciąg (patrz rys. 2.3). Z każdym klastrem można związać informacje o liczbie wektorów, którą reprezentuje. Najczęściej nazywa się to masą klastra [62].

Jeśli liczba powstałych klastrów jest zbyt duża, szczególnie gdy dokonuje się pewnych arbitralnych podziałów w poszczególnych wymiarach, można wtedy (a nawet należy), po powyżej opisanym etapie, dokonać próby łączenia klastrów. Często zdarza się, iż w jakimś wymiarze możemy mieć do czynienia z nieefektywnym podziałem — zbyt szczegółowym — i tym samym nie wnoszącym żadnych korzyści, a z drugiej strony zwiększającym liczbę klastrów. W takich przypadkach na pewno część klastrów da się połączyć i w ten sposób otrzymujemy prostszy opis danych uczących. Tym samym zmniejszamy złożoność wstępną modelu. Etap ten można zrealizować za pomocą metody dendrogamowej, w której zostaną połączone najbliższe klastry, zgodnie z odpowiednio dobraną miarą odległości (patrz podrozdział 2.3.5) lub poprzez przeszukiwanie powstałego drzewa decyzyjnego [62] i łączenie sąsiadujących klastrów.

Bywają też sytuacje w których same histogramy nie są wystarczające. Na przykład dane o rozkładzie przedstawionym na rys. 2.4 — punkty podziałów, które wynikają z analizy histogramów, nie prowadzą do efektywnego podziału przestrzeni danych i tym samym powstałe klastry nie przynoszą pożądanych efektów.

W końcowym etapie, podobnie, jak dla metody k-średnich, następuje wyznaczenie rozmyć (szerokości) poszczególnych centrów funkcji radialnych. Należy tu wspomnieć iż warto w takiej metodzie inicjalizacji skorzystać z wielowymiarowych funkcji gaussowskich lub funkcji bicentralnych i ich rozszerzeń, które umożliwiają używanie różnych rozmyć w różnych wymiarach, co prowadzi do optymalniejszego wykorzystania informacji z klasteryzacji.

Bardzo ciekawym jest algorytm Breimana, który tworzy drzewo klasyfikacji i regresji (CART) (*ang. classification and regression tree*) [25], z powodzeniem stosowany w klasyfikacji, jak i do wyciągania reguł logicznych z danych. Algorytm ten może być również użyty do inicjalizacji sieci typu RBF. CART w każdej iteracji procesu tworzenia drzewa stara się znajdować optymalny punkt w jednym z wymiarów danych, tak, aby za pomocą jak najmniejszej liczby podziałów przestrzeni zbudować drzewo klasyfikacji. Oceny, czy w jakimś punkcie dokonać podziału, czy nie, dokonuje się na podstawie funkcji kosztu i estymacji



Rysunek 2.4: Gęstość, dla której analiza histogramów nie daje żadnych korzyści.

prawdopodobieństw danego węzła w rejonie **R**:

$$p(t) = \frac{n(t)}{n}, \tag{2.41}$$

$$p(j|t) = \frac{n_j(t)}{n(t)}, \tag{2.42}$$

n jest liczbą wektorów trenujących, $n(t)$ jest liczbą wektorów trenujących w rejonie **R**. Z kolei $n_j(t)$ jest liczbą wektorów klasy j w rejonie **R**. Funkcję kosztu można wtedy zdefiniować na kilka sposobów:

$$Q(t) = 1 - \max_j p(j|t), \tag{2.43}$$

$$Q(t) = \sum_j \sum_{i \neq j} p(i|t)p(j|t) = 1 - \sum_j [p(j|t)]^2, \tag{2.44}$$

$$Q(t) = -\sum_j p(j|t) \ln p(j|t). \tag{2.45}$$

Efektywność pierwszej funkcji jest najniższa. Może ona informować o niemożności zmniejszenia funkcji kosztu, co najczęściej nie jest prawdą. Wynika

to z uwzględniania tylko najliczniejszej klasy w danym obszarze (dla równania 2.43)[1]. Dwie pozostałe funkcje dają dość porównywalne i dobre wyniki.

Teraz, gdy chcemy w węźle t dokonać podziału na dwa węzły t_L i t_R, w punkcie v trzeba móc ocenić zmniejszenie *zanieczyszczenia* poprzez ten podział:

$$\Delta Q(v,k,t) = Q(t) - Q(t_L)p_L(t) - Q(t_R)p_R(t), \tag{2.46}$$

gdzie $p_L(t)$ i $p_R(t)$ są zdefiniowane przez:

$$p_L(t) = \frac{p(t_L)}{p(t)}, \tag{2.47}$$

$$p_R(t) = \frac{p(t_R)}{p(t)}. \tag{2.48}$$

Rekurencyjny proces podziałów przebiega aż do spełnienia pewnego warunku stopu, który zazwyczaj jest określony przez osiągnięcie pewnego progu klasyfikacji. Proces tworzenia drzewa klasyfikacji może czasami prowadzić do jego przerośnięcia, a wtedy dochodzi do klasyfikacji szumu danych uczących. CART w tym momencie uruchamia procedurę usuwania zbędnych węzłów, która minimalizuje karę za ryzyko:

$$R_{pen} = R_{emp} + \lambda|T|, \tag{2.49}$$

R_{emp} jest współczynnikiem błędu klasyfikacji dla danych treningowych, $|T|$ jest liczbą węzłów–liści w drzewie. Optymalna wartość parametru λ jest wyznaczana poprzez minimalizację błędu klasyfikacji dla różnych podzbiorów zbioru uczącego.

Do podobnego schematu inicjalizacji można wykorzystać kryterium dipolowe [17, 15, 16], jak i inną metodę zbliżoną do metody CART [110] opartą o kryterium separowalności.

2.4. Uczenie z nadzorem sieci RBF

Ten podrozdział omawia najbardziej typowy sposób uczenia sieci z radialnymi funkcjami bazowymi. Metoda ta bazuje na wcześniej ustalonej architekturze sieci, w której przede wszystkim została ustalona liczba węzłów ukrytych i typ funkcji transferu realizowany przez te węzły. Zakłada się również, iż dokonano wstępnej inicjalizacji sieci RBF, na przykład za pomocą jednej z metod inicjalizacji przedstawionych w podrozdziale 2.3. Prezentowana tu metoda uczenia z nadzorem polega na iteracyjnym wprowadzeniu korekcji, wyznaczanych poprzez wyliczanie kierunku gradientu funkcji błędu. Poggio i Girosi [208] proponują, by uczeniu podlegały wagi wyjściowe i położenia centrów. Takie sieci

[1]Warto policzyć wartości wszystkich trzech powyższych propozycji definicji $Q(t)$ dla dwóch wektorów wartości p: [0.333 0.333 0.333] i [0.1 0.1 0.8].

nazywa się uogólnionymi sieciami RBF (GRBF) (*generalized radial basis function networks*). Z kolei Lowe [172] proponuje, by wszystkie parametry były adaptowane, czyli wagi, położenia centrów i rozmycia funkcji. Same poniższe równania, opisujące algorytm uczenia, są bardzo podobne do równań dla wyznaczonych dla algorytmu LMS (*ang. least mean square*).

Punktem wyjścia jest zdefiniowanie funkcji błędu, którą następnie algorytm będzie starał się minimalizować w procesie uczenia:

$$\mathbf{E} = \frac{1}{2}\sum_{i=1}^{n} e_i^2, \tag{2.50}$$

gdzie n jest liczbą wektorów uczących, a e_i jest błędem, jaki popełnia sieć dla i-tego wektora uczącego:

$$e_i = y_i - f(\mathbf{x}_i) = y_i - \sum_{j=1}^{M} w_j G_j(||\mathbf{x}_i - \mathbf{t}_j||), \tag{2.51}$$

y_j jest oczekiwaną wartością wyjściową, a $f(\mathbf{x}_i)$ wartością zwracaną przez sieć.

Warto zwrócić uwagę, iż funkcja kosztu określona równaniem (2.50), jest wypukła ze względu na parametry wag w_i, ale nie jest wypukła ze względu na parametry położeń centrów funkcji transferu i ich rozmyć, co może być przyczyną utykania procesu minimalizacji w lokalnych minimach przestrzeni parametrów.

Proces minimalizacji opisują poniższe równania, które wyznaczają gradienty dla poszczególnych parametrów adaptacyjnych:

$$w_i(n+1) = w_i(n) - \eta_w \frac{\partial E(n)}{\partial w_i(n)}, \tag{2.52}$$

$$\mathbf{t}_i(n+1) = \mathbf{t}_i(n) - \eta_t \frac{\partial E(n)}{\partial \mathbf{t}_i(n)}, \tag{2.53}$$

$$b_i(n+1) = b_i(n) - \eta_b \frac{\partial E(n)}{\partial b_i(n)}, \tag{2.54}$$

n oznacza numer iteracji algorytmu minimalizacji, η_w, η_t, η_b są współczynnikami szybkości adaptacji parametrów wag, położeń centrów i ich rozmyć.

Jeśli funkcją transferu jest funkcja gaussowska

$$g(\mathbf{x}; \mathbf{t}, b) = e^{-||\mathbf{x}-\mathbf{t}||^2/b^2}, \tag{2.55}$$

to poszczególne gradienty są równe:

$$\frac{\partial E(n)}{\partial w_i(n)} = \sum_{j=1}^{n} e_j(n)\ g(\mathbf{x}_j; \mathbf{t}_i(n), b_i(n)), \tag{2.56}$$

$$\frac{\partial E(n)}{\partial \mathbf{t}_i(n)} = \frac{2w_i(n)}{b_i^2(n)} \sum_{j=1}^{n} e_j(n)\ g(\mathbf{x}_j; \mathbf{t}_i(n), b_i(n))\ [\mathbf{x}_j - \mathbf{t}_i(n)], \tag{2.57}$$

$$\frac{\partial E(n)}{\partial b_i(n)} = \frac{2w_i(n)}{b_i^3(n)} \sum_{j=1}^{n} e_j(n)\ g(\mathbf{x}_j; \mathbf{t}_i(n), b_i(n))\ ||\mathbf{x}_j - \mathbf{t}_i(n)||^2. \tag{2.58}$$

Przedstawiona powyżej procedura minimalizacji nazywana jest często metodą *off-line*. Bierze się to stąd, iż poprawki związane z funkcją błędu dla wektorów testowych są nanoszone po prezentacji całego zbioru uczącego, co jest oczywiście konsekwencją ujęcia całego zbioru treningowego w definicji funkcji błędu (2.50). Sieci typu RBF i podobne można też uczyć w trybie *on-line*, co oznacza, iż celem jest bezpośrednie nanoszenie poprawek po prezentacji każdego wektora uczącego. Wtedy funkcja błędu wygląda tak:

$$\mathbf{E} = \frac{1}{2}|e_i|^2 \qquad i = 1, 2, \ldots, n. \tag{2.59}$$

Adaptacja parametrów, podobnie jak i przedtem, następuje zgodnie z równaniami (2.52, 2.53, 2.54), natomiast zmianie ulegają gradienty liczone dla poszczególnych parametrów adaptacyjnych:

$$\frac{\partial E(n)}{\partial w_i(n)} = e(n)\ g(\mathbf{x}(\mathbf{n}); \mathbf{t}_i(n), b_i(n)), \tag{2.60}$$

$$\frac{\partial E(n)}{\partial \mathbf{t}_i(n)} = 2w_i(n)e(n)\ g(\mathbf{x}(n); \mathbf{t}_i(n), b_i(n))\ \frac{\mathbf{x}(n) - \mathbf{t}_i(n)}{b_i^2(n)}, \tag{2.61}$$

$$\frac{\partial E(n)}{\partial b_i(n)} = 2w_i(n)e(n)\ g(\mathbf{x}(n); \mathbf{t}_i(n), b_i(n))\ \frac{||\mathbf{x}(n) - \mathbf{t}_i(n)||^2}{b_i^3(n)}, \tag{2.62}$$

$\mathbf{x}(n)$ określa wektor treningowy, prezentowany w n-tej iteracji procesu minimalizacji, a $e(n)$ błąd, jaki popełniła dla tego wektora sieć, równy różnicy $y_i(n) - f(\mathbf{x}(n))$.

W praktyce uczenie *off-line* jest wolniejsze i stabilniejsze, a uczenie *on-line* szybsze, ale mniej stabilne. Metoda *on-line* jest też najbardziej podobna do algorytmu LMS.

Wettschereck i Dieterich [256] porównywali działanie sieci RBF z i bez adaptacji centrów z sieciami MLP (*ang. multi-layer perceptron*), uczonymi algorytmem wstecznej propagacji (BP). Z pracy wynika, że sieci RBF, w których nie dokonuje się jedynie adaptacji wag wyjściowych, generalizują gorzej niż sieci MLP, ale sieci RBF, w których dokonuje się adaptacji położeń centrów funkcji transferu, generalizują lepiej niż sieci BP.

2.5. Rozszerzenia sieci RBF

Sieci z radialnymi funkcjami bazowymi, podobnie jak i inne modele sieci neuronowych, doczekały się licznych rozszerzeń. Nie sposób wymienić wszystkie z nich, ale mam nadzieję wspomnieć te najciekawsze i częściej spotykane. Bazując już na tym, co do tej pory przedstawiono w tym rozdziale, można skonstruować bardzo wiele różnych sieci. Mogły by się one różnić metodami inicjalizacji i/lub dalszym postępowaniem. Dołączając do tego możliwość skorzystania z różnych funkcji transferu, przedstawionych w rozdziale 1, klasa możliwych sieci neuronowych coraz bardziej się rozszerza i zmienia swoje własności.

2.5.1. Rozszerzenia głównego równania sieci RBF

Dość standardowymi już rozszerzeniami są różne rozszerzenia głównego równania sieci RBF (2.2), czyli liniowej kombinacji aktywacji funkcji transferu. Pierwszym przykładem niech będzie prosty, choć bardzo użyteczny dodatek w postaci współczynnika adaptacyjnego w_0

$$f(\mathbf{x}; \mathbf{w}, \mathbf{p}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x}, \mathbf{p}_i) + w_0, \tag{2.63}$$

którego celem jest uwolnienie samej sieci od średniej wartości wyjścia (por. [13] rozdział 3.4.3). Częściej spotyka się bardziej radykalne rozszerzenia takie jak

$$f(\mathbf{x}; \mathbf{w}, \mathbf{p}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x}, \mathbf{p}_i) + \sum_{i=1}^{m} d_i p(\mathbf{x}) \qquad m \leq N, \tag{2.64}$$

gdzie $p(\mathbf{x})$ jest wielomianem pewnego stopnia k z przestrzeni $\mathcal{R}^N$ [208].

Do typowych rozszerzeń zalicza się również użycie macierzy wag $\mathbf{C}$ w normie $||\cdot||_{\mathbf{C}}$ [209]

$$f(\mathbf{x}; \mathbf{w}, \mathbf{p}) = \sum_{i=1}^{M} w_i G_i(||\mathbf{x} - \mathbf{t}_i||_{\mathbf{C}_i}), \tag{2.65}$$

gdzie $||\cdot||_{\mathbf{C}}$ jest zdefiniowana jako

$$||\mathbf{x} - \mathbf{t}_i||_{\mathbf{C}_i} = (\mathbf{x} - \mathbf{t}_i)^T \mathbf{C}_i^T \mathbf{C}_i (\mathbf{x} - \mathbf{t}_i). \tag{2.66}$$

Jednakże taka macierz jest trudna do adaptacji ze względu na liczbę parametrów adaptacyjnych, która rośnie kwadratowo z rozmiarem przestrzeni wejściowej.

Inne, ciekawe i dość proste rozszerzenie, to użycie niejednorodnych miar odległości [258], które zostały już opisane w rozdziale 1.2.1.2.

2.5.2. Regularyzacja

Niezwykle ważną częścią rozważań nad sieciami RBF (jak i innymi sieciami), jest regularyzacja tych modeli, która w dużym stopniu wpływa na stabilizację procesu uczenia sieci i jest głównym narzędziem wspomagającym uzyskanie możliwie maksymalnej generalizacji i tym samym pozwala unikać przeuczenia się podczas adaptacji. Regularyzacja sieci RBF była już wspomniana na początku rozdziału, jednakże opis ten nie wyczerpał całego tematu, szczególnie różnych innych, ciekawych podejść do regularyzacji.

Najczęściej regularyzacja sprowadza się do dodania pewnego czynnika do funkcji błędu modelu $E_0(f)$ (2.14). Należy wspomnieć, że można jako podstawowego członu funkcji błędu używać nie tylko funkcji $E_0(f)$, ale również jej

ogólniejszej formy w postaci funkcji błędu Minkowskiego, przechodząc do normy L_R:

$$E_M(f;R) = \sum_{i=1}^{n} |y_i - f(\mathbf{x}_i)|^R \tag{2.67}$$

w szczególnym przypadku dla $R = 2$ mamy tożsamość z funkcją $E_0(f)$, czyli normę L_2, a dla $R = 1$ mamy metrykę Manhattan.

Jednym z najbardziej znanych czynników regularyzacyjnych jest rozpad wag (*ang. weight decay*). Wtedy do miary błędu modelu $E_0(f)$ (2.14) zostaje dodany czynnik regularyzacyjny:

$$E_{wd}(f,\mathbf{w}) = E_0(f) + \lambda \sum_{i=1}^{M} w_i^2. \tag{2.68}$$

W aproksymacji i statystyce ten typ regularyzacji nazywany jest regresją grzbietową (*ang. ridge regression*). Uwzględnienie takiego czynnika w funkcji błędu znacznie poprawia uzyskiwane wyniki [120]. Breiman [24] twierdzi, iż taka regularyzacja sprawia, że proces uczenia jest stabilny, natomiast nie jest tak gdy do wyznaczania niektórych parametrów uczenia stosuje się techniki uczenia na podzbiorach (patrz też niżej). Przykład zastosowania regularyzacji można zobaczyć na rysunku 2.5.

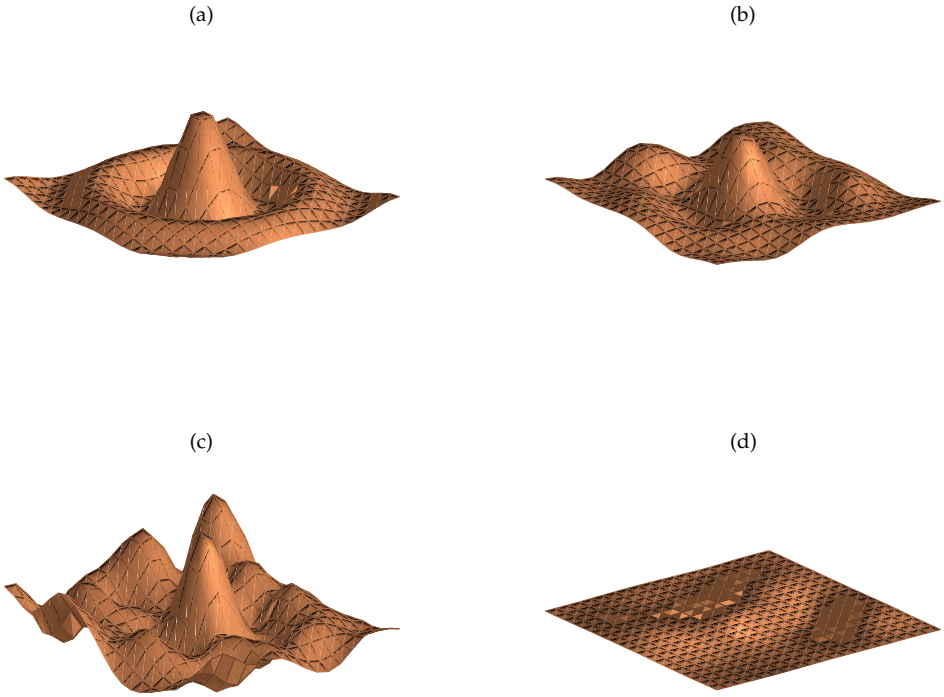
Lokalna regresja grzbietowa (*ang. local ridge regression*) jest uogólnieniem poprzedniej wersji regularyzacji:

$$E_{lrr}(f,\mathbf{w}) = E_0(f) + \sum_{i=1}^{M} \lambda_i w_i^2. \tag{2.69}$$

W przypadku takiej regresji dla lokalnych funkcji, takich, jak większość funkcji RBF, gładkość regresji nie jest kontrolowana jednym współczynnikiem, lecz każda z funkcji jest kontrolowana niezależnie. To prowadzi do lokalnej adaptacji gładkości w zależności od stanu w lokalnej części przestrzeni. Regresja nielokalna dla problemów, w których gładkość funkcji w różnych częściach przestrzeni powinna być różna, często nie daje pożądanych rezultatów [197]. Do wyznaczania parametrów regularyzacyjnych stosuje się często uczenie poprzez walidację skośną (*ang. cross-validation*, co można przetłumaczyć także jako *kroswalidację, walidację krzyżową lub rotacyjną*, ale nie ma powszechnie przyjętego terminu w języku polski) [197, 109] i różne ich odmiany. Metoda polega na podziale zbioru uczącego na k części, a następnie usuwaniu po jednej z nich, uczeniu sieci na pozostałych i testowaniu na usuniętej części. Zebrane informacje o jakości klasyfikacji lub aproksymacji na kolejno usuwanych częściach zbioru uczącego dają obraz działania algorytmu dla wcześniej ustalonych parametrów regresji (oczywiście metodę tę można wykorzystywać do wyznaczania i innych parametrów, jak i innych modeli adaptacyjnych).

Bishop w [11] zaproponował jeszcze inny człon regularyzacyjny:

$$E_{r2} = E_0(f) + \frac{1}{2}\eta^2 \sum_n \sum_i \frac{1}{f(\mathbf{x}_n)(1 - f(\mathbf{x}_n))} \left(\frac{\partial f(\mathbf{x}_n)}{\partial x_{n,i}} \right)^2. \tag{2.70}$$



Rysunek 2.5: Zastosowanie regularyzacji do aproksymacji funkcji $10\frac{\sin(|xy|)}{|xy|}$. Rysunek a) pokazuje oryginalną funkcję. Kolejne rysunki pokazują aproksymację z regularyzacją dla b) $\lambda = 1$, c) $\lambda = 10^{-4}$, d) $\lambda = 100$. Patrz równanie (2.68).

Powyższy człon regularyzacyjny nie wymaga aby funkcjami bazowymi równania sieci RBF (2.2) była pewna funkcja Green'a. Nie wymaga się również, aby liczba funkcji bazowych była równa liczbie wektorów zbioru treningowego (porównaj podrozdział 2.1).

Bardzo ciekawym wynikiem było udowodnienie przez Bishopa, iż uczenie z regularyzacją Tikhonova jest równoważne uczeniu z szumem [12]. Poprzez uczenie z szumem rozumie się dodanie losowego szumu do wejściowego wektora uczącego przed użyciem go do procesu adaptacji.

Inne metody regularyzacji zostały przedstawione w rozdziale 4.

2.5.3. Inne metody uczenia sieci RBF

Warto tu również wspomnieć o metodzie ortogonalizacji najmniejszych kwadratów (*ang. ortogonal least squares*) Chen'a (i. in.) [43, 44] do uczenia i konstrukcji sieci z radialnymi funkcjami bazowymi. Metoda najczęściej wykorzystuje algorytm ortogonalizacji Grama-Schmidta.

Rozszerzenie tej metody o regularyzację zaproponował Orr [197].

Z kolei Bishop proponuje używanie algorytmu EM (*ang. expectation maximization*) [14] do uczenia sieci RBF.

Lowe w [175] opisał specjalną wersję sieci RBF, przeznaczoną do klasyfikacji danych poprzez estymacje prawdopodobieństw rozkładów. Metoda polega na estymacji prawdopodobieństw a posteriori $p(c|\mathbf{x})$, czyli prawdopodobieństw, że dany wektor x przynależy do klasy c. Takie prawdopodobieństwo a posteriori można zrekonstruować z prawdopodobieństw cząstkowych, korzystając z twierdzenia Bayesa

$$p(c_i|\mathbf{x}) = \frac{p(c_i)p(\mathbf{x}|c_i)}{p(\mathbf{x})}. \tag{2.71}$$

Ponieważ raczej nie zdarza się, aby za pomocą pojedynczego rozkładu gaussowskiego można było estymować rozkład danych w klastrach danej klasy, używa się mieszanki rozkładów warunkowych $q(\mathbf{x}|s)$ z różnymi współczynnikami mieszania

$$p(x) = \sum_s p(s)q(\mathbf{x}|s), \tag{2.72}$$

$$p(\mathbf{x}|c_i) = \sum_s p(s;i)q(\mathbf{x}|s), \tag{2.73}$$

wtedy wykorzystując (2.72) i (2.73), równanie (2.71) przyjmuje postać

$$p(c_i|\mathbf{x}) = \sum_s \frac{p(c_i)p(s;i)}{p(s)} \cdot \frac{p(s)q(\mathbf{x}|s)}{\sum_{s'} p(s')q(\mathbf{x}|s')} \equiv \sum_j w_{ij}G(\mathbf{x}|j), \tag{2.74}$$

współczynniki $w_{ij} = p(c_i)p(s;i)\,/\,p(s)$ są wagami warstwy wyjściowej, opisującymi istotność j-tego węzła-podrozkładu dla i-tej klasy, a funkcjami bazowymi są znormalizowane funkcje $G(\mathbf{x}|j)$ (porównaj z równaniem (1.80)).

Jeszcze innym sposobem uczenia sieci RBF może być algorytm uczenia stosowany do Support Vector Machines (SVM), który dokładniej zostanie opisany w rozdziale 3.

Wilson i Martinez [258] pokazują używając sieci RBF, iż kiedy atrybuty danych są różnych typów (ciągłe, dyskretne, nominalne), należy wtedy dobrać odpowiednią miarę odległości, aby uzyskać możliwie najlepsze rezultaty. Różne miary odległości zostały już przedstawione w podrozdziale 1.2.1.

2.6. Porównanie sieci RBF z sieciami MLP

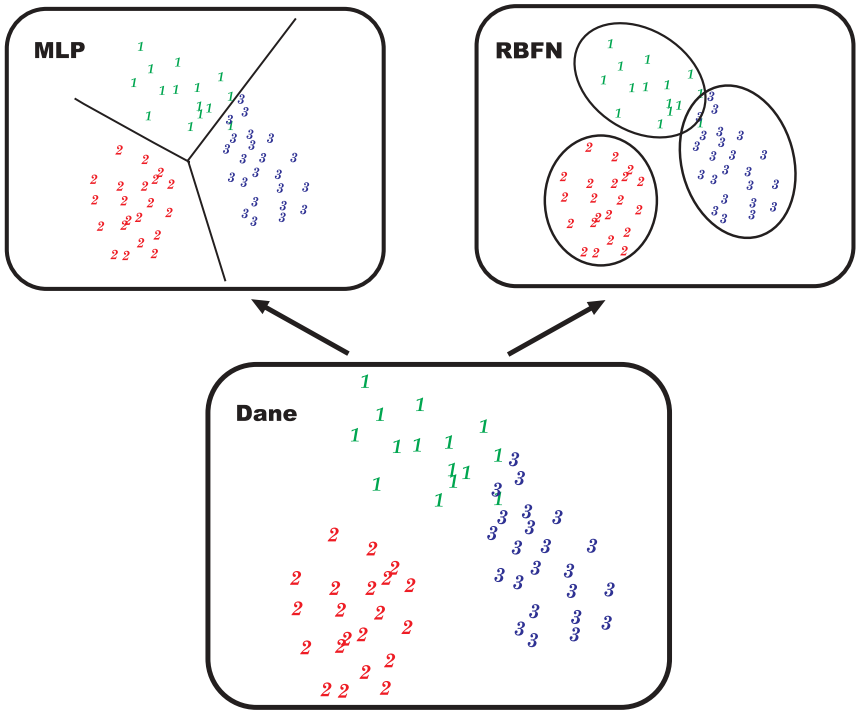
Jak widać z równania (2.2) główną różnicą pomiędzy siecią RBF i MLP są funkcje transferu neuronów warstw ukrytych. Dla sieci RBF mamy pewną funkcję radialną

$$h_i^{RBF} = \phi_1(||\mathbf{x} - \mathbf{t}_i||), \tag{2.75}$$

podczas gdy dla sieci MLP jest to produkt skalarny

$$h_i^{MLP} = \phi_2(\mathbf{x}^T \mathbf{t}_i). \tag{2.76}$$

Stąd też i sposoby działania tych sieci różnią się zasadniczo. Sieć MLP użyta do klasyfikacji będzie dzieliła wielowymiarową przestrzeń hiperpłaszczyznami. Przez to część powstałych podobszarów jest nieskończona. Natomiast sieć RBF będzie tworzyła lokalne obszary wokół klastrów danych (patrz rys. 2.6).



Rysunek 2.6: Podziały przestrzeni danych przy użyciu sieci RBF i MLP.

Również patrząc niejako *z góry* na całościowe równania sieci MLP i sieci RBF można dostrzec sporą różnicę:

$$\textbf{RBF}: \quad f_1(\mathbf{x}; \mathbf{w}) = \mathbf{w} \cdot \mathbf{G}(\mathbf{x}) \tag{2.77}$$

$$\textbf{MLP}: \quad f_2(\mathbf{x}; \mathbf{w}) = \sigma\left(\sum_{i_1} w_{i_1}^1 \sigma\left(\sum_{i_2} w_{i_2}^2 \sigma\left(\ldots \sigma\left(\sum_{i_L} w_{i_L}^L x_{i_L}\right)\ldots\right)\right)\right). \tag{2.78}$$

Wartym uwagi jest też porównanie powierzchni stałych wartości aktywacji neuronów wielowarstwowej sieci MLP i sieci RBF. W przypadku sieci typu MLP,

wartość aktywacji neuronów jest stała dla wektorów leżących na hiperpłaszczyźnie

$$\mathbf{w}^T\mathbf{x} + w_0 = const, \tag{2.79}$$

natomiast w przypadku radialnych funkcji bazowych aktywacja jest stała na powierzchni hipersfery określonej miarą odległości

$$||\mathbf{x} - \mathbf{t}||^2 = const. \tag{2.80}$$

Z kolei, gdy wektory wejściowe mają normę równą 1 ($||\mathbf{x}|| = 1$), to można to wykorzystać do nielosowej inicjacji parametrów uczenia sieci MLP i tym samym ułatwić proces uczenia się sieci. Taka inicjalizacja jest częściowym odpowiednikiem nielosowego wyboru wag startowych w sieciach RBF.

Normę wejściowych wektorów równą 1 można uzyskać poprzez transformacje wektora wejściowego, dodanie do niego dodatkowego wymiaru i wyrzutowania na sferę jednostkową. Wtedy, łącząc to z wybraną metodą klasteryzacji (patrz rozdział 2.3) możemy dokonać wyboru liczby neuronów i wstępnej inicjalizacji wag, zgodnie z procedurą opisaną w [63]. Niech $\mathbf{x} = \{x_1, , x_2 \ldots, x_d\}$, wtedy niech $\mathbf{x}' = \{x'_1, x'_2, \ldots, x'_d, x'_{d+1}\}$ będzie zdefiniowany jako [54]

$$\begin{aligned} x'_i &= \frac{x_i}{r} \qquad i = 1, 2, \ldots, d, \\ x'_{d+1} &= \sqrt{1 - \frac{1}{r^2}\sum_{i=1}^{d} x_i^2} \end{aligned} \quad , \tag{2.81}$$

gdzie $r \geq \max_{<\mathbf{x},y>\in\mathcal{S}} ||\mathbf{x}||$, $\mathcal{S}$ jest zbiorem wektorów uczących. Po takiej transformacji wszystkie punkty zostają wyrzutowane na półhipersferę. Najważniejszy jest fakt, iż podobne wektory będą wtedy tworzyły dość spójne klastry. Takie klastry, reprezentowane przez odpowiednio wybrane prototypy, można odciąć hiperpłaszczyzną od owej półhipersfery, co z kolei można zrealizować, wstawiając neuron z funkcją tanh lub funkcją sigmoidalną. Parametry początkowe tych funkcji można wyznaczyć już całkiem łatwo.

Niezależnie bardzo podobny sposób inicjalizacji zaproponowali Denoeux i Lengellé [54], a ich przykłady dowodzą, iż taki sposób inicjalizacji znacznie poprawia efektywność późniejszego procesu uczenia sieci ze wsteczną propagacją błędu.

Istnieje możliwość uzyskania efektywniejszej transformacji, poniższa transformacja ma na celu wykorzystanie jak największej powierzchni hipersfery, a nie jej połowy lub części połowy. Z drugiej strony nie wykorzystuje całej hipersfery, aby bardzo odległe dane nie mogły ulec błędnemu połączeniu — rozpinanie płaszczyzny na kule powoduje, iż krawędzie spotykają się. Przekształcenie następuje niezależnie dla każdej i-tej ($i = 1, 2, \ldots, d$) współrzędnej p-tego wektora

($\mathbf{x}_p = \{x_{p,i}, x_{p,i}, \ldots, x_{p,i}\}$):

$$x'_{p,i} = x_{p,i} - (\max_s x_{s,i} + \min_s x_{s,i})/2, \tag{2.82}$$

$$x''_{p,i} = \eta \cdot 2 \cdot x'_{p,i} / \max_s ||\mathbf{x}'_s||, \tag{2.83}$$

$$x'''_{p,i} = \begin{cases} -|x''_{p,i} - 1| + 1 & x''_{p,i} \geq 0 \\ |x''_{p,i} + 1| - 1 & x''_{p,i} < 0 \end{cases}, \tag{2.84}$$

$$x'''_{p,d+1} = \text{sign}\left(1 - ||\mathbf{x}''_p||^2\right) \sqrt{\left|1 - ||\mathbf{x}'''_p||^2\right|}, \tag{2.85}$$

η określa jaką część obwodu hipersfery będzie rozpinać transformacja. Finalny wektor $\mathbf{x}'''_p$ jest określony poprzez $\{x'''_{p,1}, x'''_{p,2}, \ldots, x'''_{p,d+1}\}$ gdzie p określa indeks wektora uczącego.

Należy się jednak liczyć z tym, że transformacja jest niezależna dla każdego wymiaru i tym samym ulegają zmianie zależności pomiędzy poszczególnymi wymiarami. Na rysunku 2.7 można zobaczyć przykład użycia powyższej transformacji dla przestrzeni dwu wymiarowej. Na przykładzie widać jak 4 klastry zostały rozmieszczone na hipersferze. Po wykonaniu takiego przekształcenia danych można przejść do następnych etapów opisanych w [63], czyli dokonać klasteryzacji, która umożliwi wyznaczenie prototypów, na których podstawie zostaną obliczone wartości wag w nowej przestrzeni $d+1$ wymiarowej.

2.7. Probabilistyczne sieci neuronowe

Probabilistyczne sieci neuronowe (PSN) (*ang. probabilistic neural networks*) zaproponowane przez Spechta [235] mają bardzo zbliżoną architekturę do sieci typu RBF. Celem PSN było zminimalizowanie współczynnika ryzyka podczas podejmowania decyzji, która z klas c ($c = 1, \ldots, C$) dla danego problemu odpowiada pewnemu wektorowi $\mathbf{x}$. W tym celu należy wyznaczyć klasę k dla której mamy:

$$P(c_k|\mathbf{x}) > P(c_j|\mathbf{x}) \qquad k \neq j. \tag{2.86}$$

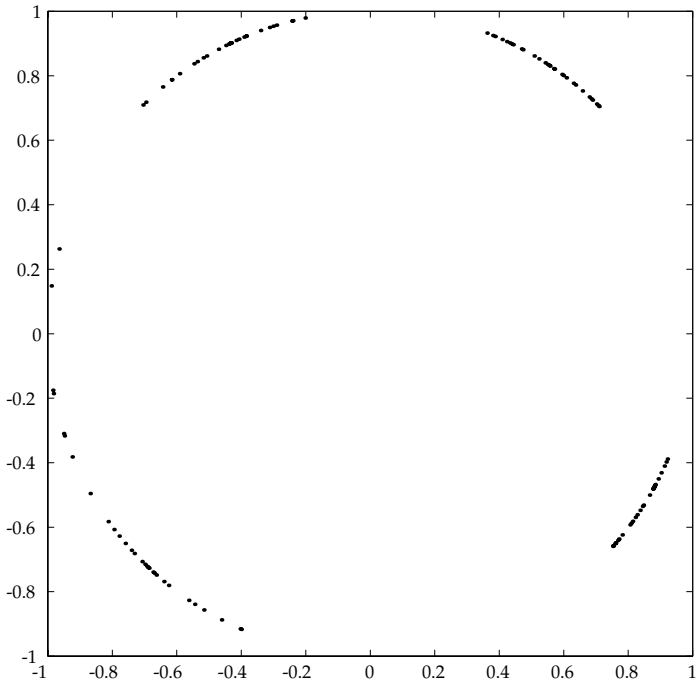
Aby wyznaczyć prawdopodobieństwa $P(c_i|\mathbf{x})$ skorzystano z reguły Bayesa:

$$P(c_i|\mathbf{x}) = \frac{P(\mathbf{x}|c_i)P(c_i)}{P(\mathbf{x})}. \tag{2.87}$$

W celu estymacji $P(\mathbf{x}|c_i)$ posłużono się poniższą definicją estymatora:

$$f_n(\mathbf{x}; c_k) = \frac{1}{(2\pi)^{d/2}\sigma^d} \frac{1}{n} \sum_{i=1, y_i=c_k}^{n} \exp[-(\mathbf{x} - \mathbf{x}_i)^\top (\mathbf{x} - \mathbf{x}_i)/2\sigma^2]. \tag{2.88}$$

Funkcja jądrowa (ang. *kernel function*) użyta wewnątrz powyższej sumy nie musi być wyłącznie funkcją gaussowską, w ogólności funkcja jądrowa K musi



Rysunek 2.7: Przykładowa transformacja danych wejściowych z czterema klastrami na hipersferę.

spełniać warunki:

$$\sup_{\mathbf{x}} |K(\mathbf{x})| < \infty, \tag{2.89}$$

$$\int |K(\mathbf{x})|\, d\mathbf{x} < \infty, \tag{2.90}$$

$$\lim_{||\mathbf{x}||\to\infty} ||\mathbf{x}||^d |K(\mathbf{x})| = 0, \tag{2.91}$$

$$\int K(\mathbf{x})\, d\mathbf{x} = 1. \tag{2.92}$$

Warto zwrócić uwagę, że Cacoullos [35] wykazał poniższą własność:

$$\lim_{n\to\infty} E[f_n(\mathbf{x}) - f_{n-1}(\mathbf{x})]^2 = 0, \tag{2.93}$$

w punktach ciągłości funkcji f. $E[\cdot]$ oznacza wartość oczekiwaną.

Sumowanie we wzorze 2.88 następuje po wszystkich wektorach zbioru uczącego, które należą do klasy c_i. W sieci PSN wektory uczące są zakodowane w wagach pomiędzy warstwą wejściową i ukrytą jako wektory–wzorce (tj. $\mathbf{w}_i = \mathbf{x}_i$

dla danej klasy c_i). Zakłada się, że wektory $\mathbf{x}$ i $\mathbf{x}_i$ są znormalizowane. To prowadzi do następującej zależności:

$$f_n(\mathbf{x}; c_k) = \frac{1}{(2\pi)^{d/2}\sigma^d n} \sum_{i=1, y_i=c_k}^{n} \exp[(\mathbf{x}^\top \mathbf{x}_i - 1)/\sigma^2]. \tag{2.94}$$

Jak widać z powyższego równania sieć PNS ma tyle wyjść, ile jest różnych klas w danym zbiorze, a w warstwie ukrytej każdy neuron ma połączenia do każdego z wejść. Natomiast każdy neuron ukryty ma tylko jedno połączenie do wyjścia, które odpowiada klasie jaką koduje wektor jego wag pomiędzy nim i wejściem.

Podjęcie decyzji przez sieć, to wyznaczenie klasy zwycięskiej poprzez:

$$c = \arg\max_i P(c_i) f_n(\mathbf{x}; c_i). \tag{2.95}$$

Dzięki kodowaniu wzorców uczących w wagach pomiędzy wejściem i warstwą ukrytą, a następnie ważeniu, jak we wzorze powyżej nie ma tak naprawdę procesu uczenia. Można jedynie powiedzieć, że następuje *zapamiętywanie* wzorców. Tym samym oszczędzamy na procesie uczenia, ale sieć musi pamiętać wszystkie wzorce.

Probabilistyczne sieci neuronowe mogą być użyte również w regresji. Jak pokazano w [221] estymator funkcji regresji przyjmuje wtedy postać:

$$F_n(\mathbf{x}) = \frac{\sum_{i=1}^{n} y_i \exp[(\mathbf{x}^\top \mathbf{x}_i - 1)/\sigma^2]}{\sum_{i=1}^{n} \exp[(\mathbf{x}^\top \mathbf{x}_i - 1)/\sigma^2]}. \tag{2.96}$$

W [265] zaproponowano także rekurencyjny estymator funkcji gęstości. Ciekawe zastosowanie w rozpoznawaniu obrazów za pomocą zmodyfikowanych probabilistycznych sieci neuronowych przedstawiono w [221].

Rozdział 3

Support Vector Machines (SVM)

Innym, bardzo dobrze określonym matematycznie modelem, jest Support Vector Machine (SVM), czyli maszyna wektorów podpierających (bądź wspierających) zaproponowana przez Vapnika w [249, 250, 251]. Pierwotnie SVM przeznaczony był do klasyfikacji i regresji. Dziś jednak został rozszerzony w bardzo wielu kierunkach [229]. Jak zobaczymy poniżej równanie modelu SVM będzie takie samo jak równanie opisujące sieć RBF. Jednak ogromna różnica drzemie w procesie wyznaczania *wag* funkcji bazowych (z punktu widzenia sieci RBF), jak i (częściowa różnica) określeniu centrów funkcji bazowych.

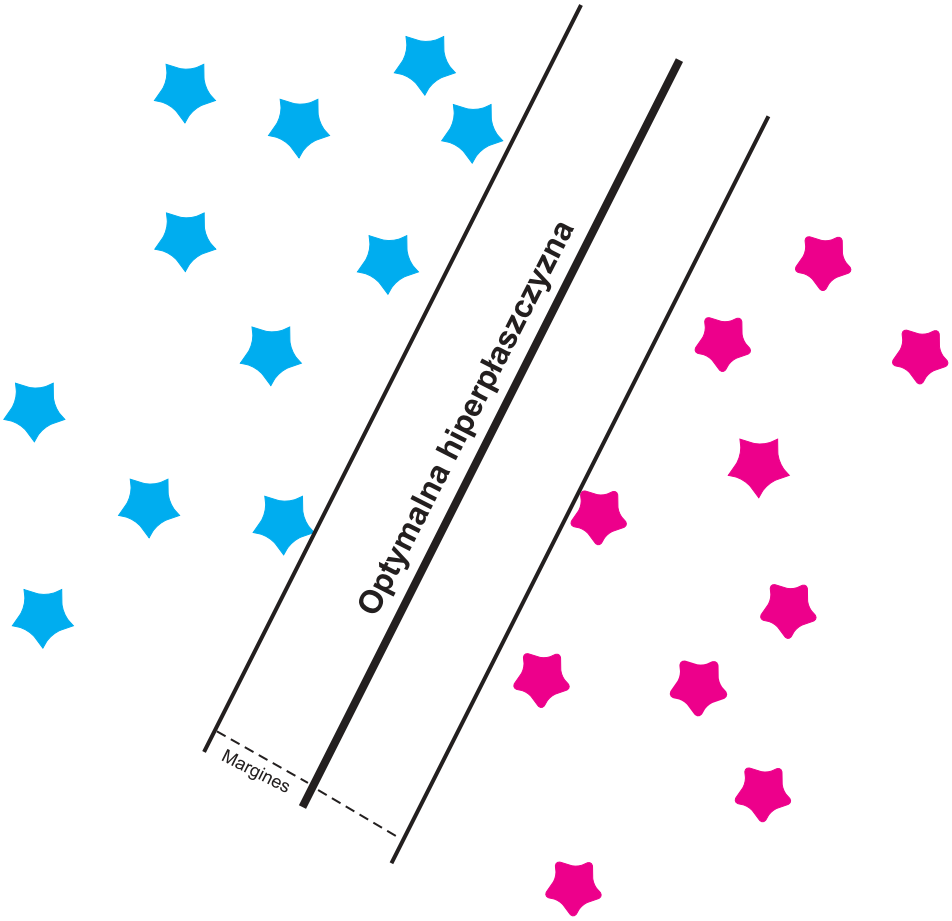
Istotą metody SVM jest konstrukcja *optymalnej hiperpłaszczyzny*, której zadaniem jest rozseparowanie danych, należących do przeciwnych klas, z możliwie największym **marginesem zaufania**. Poprzez margines zaufania rozumie się tu odległość pomiędzy optymalną hiperpłaszczyzną i najbliższym jej wektorem. Przykład takiej hiperpłaszczyzny ilustruje rysunek 3.1. Jak widać z rysunku płaszczyzn separujących dwie klasy może być wiele, lecz z punktu widzenia optymalności najciekawsze rozwiązanie może być zdefiniowane przez uzyskanie możliwie największego marginesu zaufania. Tak zdefiniowana optymalna hiperpłaszczyzna, określona przez współczynniki $\mathbf{w}$ i b, spełnia poniższą nierówność

$$\frac{y_k F(\mathbf{x}_k)}{||\mathbf{w}||} \geq \tau \qquad k = 1, 2, \ldots, n, \tag{3.1}$$

oczywiście przy założeniu, że istnieje margines ufności τ, a $F(\mathbf{x})$ jest zdefiniowane (na początek) przez:

$$F(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b. \tag{3.2}$$

Mamy wtedy tak naprawdę zadanie liniowej dyskryminacji. To jednak, jak wiadomo, nie pozwala rozwiązywać naprawdę trudnych problemów (nieseparowalnych liniowo).



Rysunek 3.1: Optymalna hiperpłaszczyzna.

3.1. Funkcje jądrowe

Z powodu możliwości wystąpienia liniowej nieseparowalności w przestrzeni wejściowej ideą SVM nie stała się konstrukcja optymalnej hiperpłaszczyzny w przestrzeni wejściowej, lecz w pewnej wysokowymiarowej przestrzeni cech $\mathcal{Z}$, która najczęściej jest nieliniowym produktem pewnych funkcji bazowych $\phi_i(\mathbf{x})$ (wybranych *a priori*), określonych w przestrzeni wejściowej. Wtedy równanie optymalnej hiperpłaszczyzny przyjmuje postać:

$$F(\mathbf{x}) = \sum_{i=1}^{n} a_i y_i K(\mathbf{x}_i, \mathbf{x}) + b, \tag{3.3}$$

gdzie $K_i(\mathbf{x}_i, \mathbf{x})$ jest jądrem iloczynu skalarnego (*ang. inner product kernel*) funkcji bazowych (przestrzeni cech $\mathcal{Z}$) $\phi_j(\mathbf{x}), j = 1, 2, \ldots, m$. Łatwo zauważyć, że w miejsce $w_i x_i$ z równania 3.2 mamy $a_i y_i K(\mathbf{x}_i, \mathbf{x})$. To powoduje, że nie szukamy już optymalnej hiperpłaszczyzny bezpośrednio w oparciu o wektory przestrzeni wejściowej. W ogólności m może być nieskończone. Iloczyn skalarny może być zdefiniowany przez poniższe równanie

$$K(\mathbf{x}, \mathbf{x}') = \phi_i(\mathbf{x})^\top \phi_i(\mathbf{x}'). \tag{3.4}$$

Wtedy poszukujemy rozwiązania — nadal liniowej dyskryminacji — w zupełnie innej przestrzeni. W tej przestrzeni, jak zobaczymy, poszukiwanie rozwiązania okazało się znacznie efektywniejsze. Można bez problemu rozwiązywać takie zagadnienia, które w wejściowej przestrzeni nie były liniowo separowalne. Co więcej, jak zobaczymy nieco później, będziemy mogli rozwiązywać nawet takie problemy, które nie są liniowo separowalne w przestrzeni cech $\phi(\cdot)$.

Zgodnie z teorią przestrzeni Hilberta, funkcje jądrowe K reprezentujące iloczyn skalarny muszą być dodatnio określone:

$$\int\int K(\mathbf{x}, \mathbf{x}') f(\mathbf{x}) f(\mathbf{x}')\ d\mathbf{x}\ d\mathbf{x}' > 0 \qquad \text{dla} \quad f \neq 0, \int f^2(\mathbf{x}) d\mathbf{x} < \infty. \tag{3.5}$$

Iloczyn skalarny $K(\cdot)$ może przyjmować bardzo różne postaci, np. dla wielomianów stopnia q mamy:

$$K(\mathbf{x}, \mathbf{x}') = [\gamma(\mathbf{x}^\top \mathbf{x}') + \theta]^q, \tag{3.6}$$

w najprostszej wersji mamy:

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'. \tag{3.7}$$

Jednak nie każda postać K z 3.6 jest dodatnio określona. I tak dla $\theta = 0$ i $\theta = 1$, gdy q jest naturalne, to, jak pokazano w [229], K jest dodatnio określone.

Może to być także funkcja gaussowska (w praktyce ta postać daje najpewniejsze wyniki):

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{\gamma}\right), \tag{3.8}$$

gdzie $\gamma > 0$. Wtedy równanie 3.3 przyjmuje postać sieci RBF.

Często stosuje się także funkcję tangensa hiperbolicznego jako $K(\cdot)$:

$$K(\mathbf{x}, \mathbf{x}') = \tanh(\gamma[\mathbf{x}^\top \mathbf{x}'] + \theta). \tag{3.9}$$

Dla powyższej definicji funkcji jądrowej $K(\cdot)$ równanie 3.3 definiuje sieć MLP (dokładniej jej szczególny jednowarstwowy przypadek …).

W [229] pokazano także funkcję jądrową w postaci wielomianu Vovka:

$$K(\mathbf{x}, \mathbf{x}') = \frac{1 - [\mathbf{x}^\top \mathbf{x}']^q}{1 - [\mathbf{x}^\top \mathbf{x}']}. \tag{3.10}$$

Powyższa funkcja jądrowa także jest dodatnio określona gdy q jest dowolną wartością naturalną.

3.2. Konstrukcja optymalnej hiperpłaszczyzny

Dla uproszczenia powróćmy jednak na pewien czas do przestrzeni wejściowej, aby w niej poszukiwać optymalnej hiperpłaszczyzny.

Uzyskanie możliwie największego marginesu pomiędzy dwiema klasami wymaga maksymalizacji odległości punktów (najbliższych) od hiperpłaszczyzny. Taki cel może być zapisany w następujący sposób:

$$\max_{\mathbf{w},b} \quad \min\{||\mathbf{x}-\mathbf{x}_i|| \ : \ \mathbf{w}^\top\mathbf{x}+b=0, \quad i=1,\ldots,N\}. \tag{3.11}$$

Teraz przeskalowujemy $\mathbf{w}$ i b tak, aby najbliższe punkty hiperpłaszczyzny $\mathbf{w}^\top\mathbf{x}+b=0$ leżały na hiperpłaszczyźnie zdefiniowanej przez:

$$\mathbf{w}^\top\mathbf{x}+b=\pm 1. \tag{3.12}$$

Wtedy dla wszystkich wektorów $\mathbf{x}_i$ mamy:

$$y_i[\mathbf{w}^\top\mathbf{x}_i+b]\geq 1. \tag{3.13}$$

Stąd łatwo wyznaczyć szerokość marginesu. Weźmy dwa najbliższe punkty $\mathbf{x}_1$ i $\mathbf{x}_2$ do hiperpłaszczyzny, po jednym z każdej klasy. Następnie zrzutujmy je wzdłuż prostej prostopadłej używając wektora normalnego $\mathbf{w}/||\mathbf{w}||$ i policzmy odległość pomiędzy zrzutowanymi punktami (porównaj rysunek 3.2):

$$\left[\frac{\mathbf{w}}{||\mathbf{w}||}\right]^\top[\mathbf{x}_1-\mathbf{x}_2]=\frac{2}{||\mathbf{w}||}. \tag{3.14}$$

Teraz 3.11 można przekształcić do funkcji celu (*ang. objective function*):

$$\min_{\mathbf{w},b} \ \tau(\mathbf{w})=\frac{1}{2}||\mathbf{w}||^2, \tag{3.15}$$

przy warunkach:

$$y_i[\mathbf{w}^\top\mathbf{x}_i+b]\geq 1 \quad i=1,\ldots,N. \tag{3.16}$$

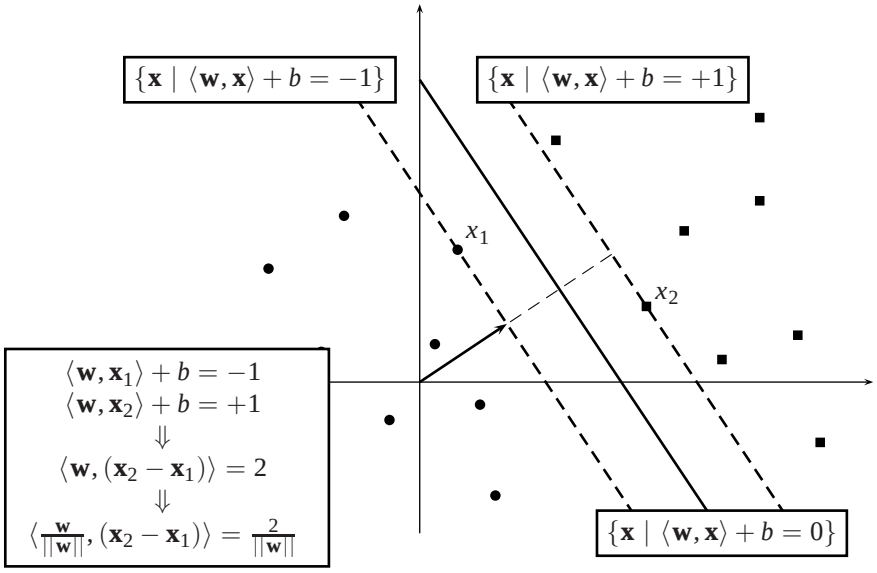
Wyrażenia 3.15 i 3.16 składają się na *problem optymalizacyjny z ograniczeniami*. Pomoc w rozwiązaniu takich problemów niesie metoda mnożników Lagrange'a. Najpierw definiujemy lagrangian:

$$L(\mathbf{w},b,\boldsymbol{\alpha})=\frac{1}{2}||\mathbf{w}||^2-\sum_{i=1}^{m}\alpha_i(y_i[\mathbf{x}_i^\top\mathbf{w}+b]-1), \tag{3.17}$$

gdzie $\alpha_i>0$ są mnożnikami Lagrange'a. Teraz celem jest maksymalizacja lagrangiana L ze względu na współczynniki α_i i minimalizacja ze względu na $\mathbf{w}$ i b. To prowadzi do warunków, w których pochodne L ze względu na powyższe współczynniki zanikają:

$$\frac{\partial}{\partial b}L(\mathbf{w},b,\boldsymbol{\alpha}) \quad = \quad 0, \tag{3.18}$$

$$\frac{\partial}{\partial w}L(\mathbf{w},b,\boldsymbol{\alpha}) \quad = \quad 0, \tag{3.19}$$



Rysunek 3.2: Konstrukcja optymalnej hiperpłaszczyzny separującej kółka od kwadratów.

co prowadzi do:

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \tag{3.20}$$

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i. \tag{3.21}$$

Należy jednak zauważyć, że rozwiązanie takiego zadania może nie istnieć (brak liniowej separowalności).

Wektory $\mathbf{x}_i$, dla których $\alpha_i > 0$ nazywane są wektorami podpierającymi bądź wspierającymi (*ang. support vectors*).

Zgodnie z twierdzeniem Karush-Kuhn-Thucker teorii optymalizacji w punkcie siodłowym lagrangiana L (3.17) niezerowe są tylko te współczynniki α_i, dla których mamy:

$$\alpha_i(y_i[\mathbf{x}_i^\top \mathbf{w} + b] - 1) = 0, \quad i = 1, \ldots, N. \tag{3.22}$$

Własność 3.22 pokazuje, że wektory podpierające leżą dokładnie na marginesie. Tym samym pozostałe wektory stają się nieistotne i dla nich nierówności 3.16 są oczywiście spełnione.

Wykorzystując powyższą własność i jednocześnie podstawiając 3.20 i 3.21 do lagrangiana L 3.17 eliminujemy zmienne $\mathbf{w}$ i b. W ten sposób otrzymujemy

dualny problem optymalizacyjny:

$$\max_{\boldsymbol{\alpha}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \tag{3.23}$$

z ograniczeniami (warunkami KKT):

$$\alpha_i \geq 0 \quad i = 1, \ldots, N, \tag{3.24}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0. \tag{3.25}$$

Po powyższych przekształceniach funkcją decyzyjną (klasyfikacyjną) jest:

$$\hat{F}(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{N} \alpha_i y_i \mathbf{x}^\top \mathbf{x}_i + b \right). \tag{3.26}$$

Powracając do przestrzeni funkcji $\phi(\cdot)$ możemy przeformułować i uogólnić dualny problem 3.23:

$$\max_{\boldsymbol{\alpha}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j). \tag{3.27}$$

Ograniczenia 3.24 pozostają bez zmian. Tak określony problem poszukuje optymalnej hiperpłaszczyzny w przestrzeni funkcji $\phi(\cdot)$ (3.4).

Z kolei funkcja decyzyjna przyjmuje formę:

$$\hat{F}(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) + b \right). \tag{3.28}$$

Wartość b powyższego zbioru łatwo wyznaczyć na podstawie jednej z równości 3.22. W praktyce b często przyjmuje średnią wartość, z wartości jakie można wyznaczyć z równości 3.22 w celu poprawy dokładności.

3.3. Konstrukcja hiperpłaszczyzny dla przypadków nieseparowalnych (C-SVC)

Jak już wspomniano rozwiązanie powyższego zadania może nie istnieć. Po prostu nie zawsze istnieje płaszczyzna idealnie separująca dwie klasy nawet w przestrzeni funkcji $\phi(\cdot)$.

Aby rozwiązać ten problem Cortes i Vapnik [47] wprowadzili istotne zamiany do definicji problemu.

Nieseparowalność oznacza niemożność spełnienia warunków 3.16. Zaproponowane rozwiązanie polega na wprowadzeniu zmiennych ξ_i rozluźniających więzi nierówności 3.16:

$$y_i[\mathbf{w}^\top \mathbf{x}_i + b] \geq 1 - \xi_i \quad i = 1, \ldots, N, \tag{3.29}$$

$$\xi_i \geq 0. \tag{3.30}$$

Jak widać z 3.29 ξ_i dopuszczają, by pewne wektory $\mathbf{x}_i$ (te dla których $\xi > 0$) leżały po niewłaściwej stronie płaszczyzn określających margines. Jednak aby algorytm dobrze generalizował należy zadbać, aby współczynniki ξ_i były pewnymi karami, czyli klasyfikator powinien kontrolować szerokość marginesu $||\mathbf{w}||$ jak i wysokość kary: $\sum_i \xi_i$.

Powyższe rozważanie prowadzi do nowej funkcji celu:

$$\min_{\mathbf{w},b,\xi} \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N} \xi_i \tag{3.31}$$

z ograniczeniami 3.29 i 3.30. C musi być większe od 0.

Zgodnie z twierdzeniem Karush-Kuhn-Tuckera lagrangian 3.31 przyjmie postać:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i(y_i[\mathbf{x}_i^\top \mathbf{w} + b] - 1 + \xi_i) - \sum_{i=1}^{N} \mu_i \xi_i, \tag{3.32}$$

μ_i są mnożnikami Lagrange'a wymuszającymi dodatniość ξ_i.

To, tak jak poprzednio, prowadzi do warunków, w których pochodne L zanikają:

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\mu}) = 0, \tag{3.33}$$

$$\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\mu}) = 0, \tag{3.34}$$

mamy wtedy:

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{3.35}$$

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{3.36}$$

z warunkami:

$$\frac{\partial}{\partial \xi_i} L(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\mu}) = C - \alpha_i - \mu_i = 0 \tag{3.37}$$

$$y_i[\mathbf{x}_i^\top \mathbf{w} + b] - 1 + \xi_i \geq 0, \tag{3.38}$$

$$\xi_i \geq 0, \tag{3.39}$$

$$\alpha_i \geq 0, \tag{3.40}$$

$$\mu_i \geq 0, \tag{3.41}$$

$$\alpha_i(y_i[\mathbf{x}_i^\top \mathbf{w} + b] - 1 + \xi_i) = 0, \tag{3.42}$$

$$\mu_i \xi_i = 0. \tag{3.43}$$

Ostatecznie prowadzi to do dualnego problemu optymalizacyjnego:

$$\max_{\boldsymbol{\alpha}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \tag{3.44}$$

z ograniczeniami:

$$0 \leq \alpha_i \leq C \quad i = 1, \ldots, N, \tag{3.45}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0. \tag{3.46}$$

3.4. ν-SVC

W powyższym problemie ważną rolę odgrywa współczynnik C, który jak napisali Schölkopf i Smola [229] jest „nieintuicyjny", choć spełnia rolę złotego środka pomiędzy generalizacją (szerokością marginesu) i liczbą błędów na zbiorze treningowym. Proszę zauważyć, że wpływ współczynnika C jest raczej eksponencjalny niż liniowy. Z powyższych powodów zaproponowano nowy algorytm ν-SVM w [228], gdzie parametr C zastąpiono przez ν. Pierwotny problem optymalizacyjny zdefiniowano przez:

$$\min_{\mathbf{w}, b, \boldsymbol{\xi}, \rho} \quad \tau(\mathbf{w}, \boldsymbol{\xi}, \rho) = \frac{1}{2}||\mathbf{w}||^2 - \nu\rho + \frac{1}{N} \sum_{i=1}^{N} \xi_i \tag{3.47}$$

z ograniczeniami:

$$y_i[\mathbf{x}_i^\top \mathbf{w} + b] \geq \rho - \xi_i, \tag{3.48}$$

$$\xi_i \geq 0, \tag{3.49}$$

$$\rho \geq 0 \tag{3.50}$$

ρ, jak widać, jest dodatkowym parametrem optymalizacyjnym. Przy założeniu, że $\boldsymbol{\xi} = \mathbf{0}$ klasy są odseparowane od siebie marginesem szerokości $2\rho / ||\mathbf{w}||$.

Parametr ν charakteryzuje się ciekawymi własnościami. Przypuśćmy, że w rezultacie optymalizacji finalna wartość ρ jest większa od 0, mamy wtedy:

1. ν jest ograniczeniem górnym na procent wektorów leżących wewnątrz marginesu:

$$\frac{1}{N}|\{i : \ y_i F(\mathbf{x}_i) < \rho\}|, \tag{3.51}$$

2. ν jest ograniczeniem dolnym na procent wektorów podpierających (*support vectors (SVs)*) względem całego zbioru wektorów.

Powyższe własności ilustruje tabela 3.1.

ν	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
współczynnik błędów	0.00	0.07	0.25	0.32	0.39	0.50	0.61	0.71
współczynnik SV	0.29	0.36	0.43	0.46	0.57	0.68	0.79	0.86
margines $\rho/\|\|\mathbf{w}\|\|$	0.005	0.018	0.115	0.156	0.36	0.42	0.46	0.55

Tabela 3.1: Zależności pomiędzy ν, współczynnikiem błędu, ilością wektorów podpierających (support vectors) i szerokością marginesu. Wartości tabeli zaczerpnięte z [229].

Aby przejść do postaci dualnej, podobnie jak poprzednio budujemy lagrangian:

$$\begin{aligned} L(\mathbf{w}, b, \boldsymbol{\xi}, \rho, \boldsymbol{\alpha}, \boldsymbol{\beta}, \delta) =& \frac{1}{2}||\mathbf{w}||^2 - \nu\rho + \frac{1}{N}\sum_{i=1}^{N}\xi_i \\ & - \sum_{i=1}^{N}(\alpha_i(y_i[\mathbf{x}_i^\top \mathbf{w} + b] - \rho + \xi_i) + \beta_i\xi_i) - \delta\rho \end{aligned}, \tag{3.52}$$

gdzie $\alpha_i, \beta_i, \delta \geq 0$ są mnożnikami Lagrange'a. Funkcja L ma być minimalizowana ze względu na zmienne $\mathbf{w}, \boldsymbol{\xi}, b, \rho$ i minimalizowana ze względu na zmienne (dualne) $\boldsymbol{\alpha}, \boldsymbol{\beta}, \delta$.

Wyznaczmy teraz pochodne cząstkowe:

$$\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\xi}, \rho, \boldsymbol{\alpha}, \boldsymbol{\beta}, \delta) = 0, \tag{3.53}$$

$$\frac{\partial}{\partial \boldsymbol{\xi}} L(\mathbf{w}, b, \boldsymbol{\xi}, \rho, \boldsymbol{\alpha}, \boldsymbol{\beta}, \delta) = 0, \tag{3.54}$$

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\xi}, \rho, \boldsymbol{\alpha}, \boldsymbol{\beta}, \delta) = 0, \tag{3.55}$$

$$\frac{\partial}{\partial \rho} L(\mathbf{w}, b, \boldsymbol{\xi}, \rho, \boldsymbol{\alpha}, \boldsymbol{\beta}, \delta) = 0, \tag{3.56}$$

skąd otrzymujemy:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i, \tag{3.57}$$

$$\alpha_i + \beta_i = \frac{1}{N}, \tag{3.58}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \tag{3.59}$$

$$\sum_{i=1}^{N} \alpha_i - \delta = \nu. \tag{3.60}$$

Wstawiając 3.57 i 3.58 do L (3.57) otrzymujemy funkcję celu problemu dualnego:

$$\max_{\boldsymbol{\alpha}} \quad W(\boldsymbol{\alpha}) = -\frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{3.61}$$

z ograniczeniami:

$$0 \leq \alpha_i \leq \frac{1}{N}, \tag{3.62}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \tag{3.63}$$

$$\sum_{i=1}^{N} \alpha_i \geq \nu. \tag{3.64}$$

Ogólna postać funkcji decyzyjnej jest taka sama jak 3.28:

$$\hat{F}(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) + b \right). \tag{3.65}$$

Pozostaje jeszcze wyznaczenie współczynników b i ρ. W tym celu musimy zdefiniować dwa zbiory S_+ i S_-:

$$S_+ = \{\mathbf{x}_i : \ 0 < \alpha_i < 1 \ \wedge \ y_i = +1\}, \tag{3.66}$$

$$S_- = \{\mathbf{x}_i : \ 0 < \alpha_i < 1 \ \wedge \ y_i = -1\}. \tag{3.67}$$

Wtedy nierówność 3.48 staje się równością z $\xi_i = 0$ i mamy:

$$b = -\frac{1}{2s} \sum_{\mathbf{x} \in S_+ \cup S_-} \sum_{j=1}^{N} \alpha_j y_j K(\mathbf{x}, \mathbf{x}_j), \tag{3.68}$$

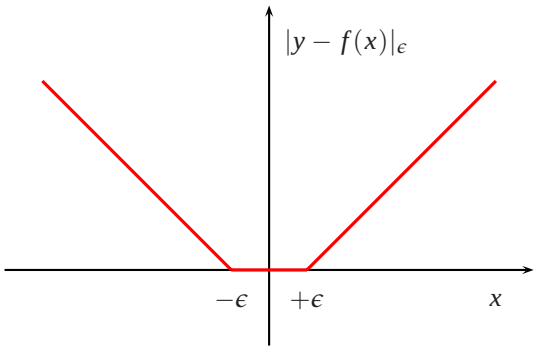
$$\rho = \frac{1}{2s} \left(\sum_{\mathbf{x} \in S_+} \sum_{j=1}^{N} \alpha_j y_j K(\mathbf{x}, \mathbf{x}_j) - \sum_{\mathbf{x} \in S_-} \sum_{j=1}^{N} \alpha_j y_j K(\mathbf{x}, \mathbf{x}_j) \right), \tag{3.69}$$

gdzie $s = |S_+| = |S_-|$.

3.5. Problem regresji (ϵ-SVR)

SVM może być także sformułowany w kontekście problemów regresyjnych. Punktem wyjścia jest tutaj zdefiniowanie istotnie innej funkcji błędu w porównaniu na przykład do funkcji 2.14. Najistotniejszą cechą tej $\epsilon-$*niewrażliwej* funkcji (*ang.* $\epsilon-$*insensitive error function*) jest jest zerowy wpływ przy odpowiednio małym błędzie absolutnym:

$$c(\mathbf{x}, y, f(\mathbf{x})) \;=\; |y - f(\mathbf{x})|_\epsilon \;=\; \max\{0, |y - f(\mathbf{x})| - \epsilon\}. \tag{3.70}$$



Rysunek 3.3: Ilustracja funkcji błędu 3.70.

Podobnie jak i w przypadku klasyfikacji, na początku można zawęzić poszukiwania do przypadku liniowego i poszukiwać rozwiązania problemu regresji liniowej:

$$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w} + b \tag{3.71}$$

poprzez minimalizację

$$\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N} |y_i - f(\mathbf{x}_i)|_\epsilon. \tag{3.72}$$

Podobnie jak w przypadku konstrukcji *miękkiej* optymalnej hiperpłaszczyzny wprowadza się zmienne łagodzące nierówności. W tym przypadku wprowadza się dwa typy zmiennych ($\boldsymbol{\xi}$ i $\boldsymbol{\xi}^*$) jedną dla przypadku $f(\mathbf{x}_i) - y_i > \epsilon$ i drugą gdy $y_i - f(\mathbf{x}_i) > \epsilon$. Wtedy problem regresji można przedstawić jako problem optymalizacyjny:

$$\min_{\mathbf{w},\boldsymbol{\xi},\boldsymbol{\xi}^*,b} \quad \tau(\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\xi}^*) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \tag{3.73}$$

przy ograniczeniach:

$$\begin{aligned} f(\mathbf{x}_i) - y_i &\leq \epsilon + \xi_i \quad i = 1, \ldots, N, && (3.74)\\ y_i - f(\mathbf{x}_i) &\leq \epsilon + \xi_i^*, && (3.75)\\ \xi_i, \xi_i^* &\geq 0. && (3.76) \end{aligned}$$

Wracając do postaci SVM z funkcjami jądrowymi i przechodząc do postaci dualnego problemu optymalizującego otrzymujemy:

$$\begin{aligned} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha}^*} \quad W(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) = &- \epsilon \sum_{i=1}^{N} (\alpha_i^* + \alpha_i) + \sum_{i=1}^{N} (\alpha_i^* - \alpha_i) y_i \\ &- \frac{1}{2} \sum_{i,j=1}^{N} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) K(\mathbf{x}_i, \mathbf{x}_j) \end{aligned} \tag{3.77}$$

z ograniczeniami:

$$0 \leq \alpha_i, \alpha_i^* \leq C \quad i = 1, \ldots, N, \tag{3.78}$$

$$\sum_{i=1}^{N} (\alpha_i - \alpha_i^*) = 0. \tag{3.79}$$

Funkcja regresji przyjmuje postać:

$$f(\mathbf{x}) = \sum_{i=1}^{N} (\alpha_i^* - \alpha_i) K(\mathbf{x}_i, \mathbf{x}) + b. \tag{3.80}$$

3.6. Problem regresji dla ν-SVM (ν-SVR)

Problem regresji może być rozwiązany także poprzez ν-SVM co zaproponowali Schölkopf i in. w [226, 228].

W ϵ-SVR ϵ jest dopuszczalnym błędem. Parametr ten jednak najczęściej należy dobierać empirycznie (prawie nigdy nie wiemy, jak dokładnej aproksymacji możemy oczekiwać). Dlatego w ν-SVR parametr ten będzie wyznaczany automatycznie (choć z kolei ręcznie będzie trzeba dobrać parametr ν).

Wstępnie problem zdefiniowano przez ($\nu \geq 0$):

$$\min_{\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \epsilon, b} \quad \tau(\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \epsilon) = \frac{1}{2} ||\mathbf{w}||^2 + C \left(\nu\epsilon + \frac{1}{N} \sum_{i=1}^{N} (\xi_i + \xi_i^*) \right) \tag{3.81}$$

przy ograniczeniach:

$$\begin{aligned} f(\mathbf{x}_i) - y_i &\leq \epsilon + \xi_i \quad i = 1, \ldots, N, && (3.82)\\ y_i - f(\mathbf{x}_i) &\leq \epsilon + \xi_i^*, && (3.83)\\ \xi_i, \xi_i^* &\geq 0, && (3.84)\\ \epsilon &\geq 0. && (3.85) \end{aligned}$$

Po przekształceniach ([228]) powyższego problemu pierwotnego otrzymujemy dualny problem optymalizacyjny :

$$\max_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \quad W(\boldsymbol{\alpha},\boldsymbol{\alpha}^*) = \sum_{i=1}^{N}(\alpha_i^* + \alpha_i)y_i - \frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)K(\mathbf{x}_i,\mathbf{x}_j) \tag{3.86}$$

z ograniczeniami:

$$\sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0 \quad i = 1,\ldots,N, \tag{3.87}$$

$$0 \le \alpha_i, \alpha_i^* \le \frac{C}{N}, \tag{3.88}$$

$$\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) \le C\nu. \tag{3.89}$$

Funkcja regresji przyjmuje postać 3.80.

Wartości b i ϵ można wyznaczyć z 3.82 i 3.83 dla $\xi_i, \xi_i^* = 0$ (wtedy $0 < \alpha_i, \alpha_i^* < C/N$).

3.7. Optymalizacja problemów programowania kwadratowego (QP)

Rozwiązania problemów optymalizacyjnych 3.23 3.27 3.44 3.61 3.77 wymagają rozwiązania problemu optymalizacji programowania kwadratowego (*ang. quadratic programming (QP)*). Wiele metod użytych w implementacjach SVM rozwiązują zagadnienia QP bazując na komercyjnych pakietach programowania kwadratowego. Warto tu wspomnieć o LOQO [247, 248] i MINOS [190]. Pewną popularnością cieszyły się także odmiany metody sprzężonego spadku gradientu [223, 30]. Dodatkowe informacje można znaleźć w [229]. Implementacje te były jednak zbyt wolne lub wręcz nieużyteczne dla zadań, w których liczba elementów zbioru uczącego wykraczała poza kilkaset.

Warto także zwrócić uwagę na opracowanie, w którym Lin i Lin porównali algorytmu SMO (*ang. Sequential Minimal Optimization*), z poprawkami Keerthiego, z innymi algorytmami redukującymi problem optymalizacyjny (raczej na ich niekorzyść) [171].

3.7.1. Dekompozycja

Punktem wspólnym algorytmów efektywnych, które powstały w ostatnich latach jest niewątpliwie metoda dekompozycji problemu QP [200, 142, 205] (*ang. decomposition method*, a czasem także: *coordinate search*, *method of alternating variables* lub *coordinate descent method*). Ogólny problem optymalizacyjny:

$$\max_{\boldsymbol{\alpha}} \quad W(\boldsymbol{\alpha}) = \mathbf{p}^\top\boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^\top Q\boldsymbol{\alpha} \tag{3.90}$$

z ograniczeniami:

$$0 \leq \alpha_i \leq C \quad i = 1, \ldots, N, \tag{3.91}$$
$$\mathbf{y}^\top \boldsymbol{\alpha} = \Delta, \tag{3.92}$$

(gdzie $Q_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$) zostaje przekształcony do:

$$\max_{\boldsymbol{\alpha}_B} \quad W(\boldsymbol{\alpha}_B) = (\mathbf{p} - Q_{BR}\boldsymbol{\alpha}_R)^\top \boldsymbol{\alpha}_B - \frac{1}{2}\boldsymbol{\alpha}_B^\top Q_{BB}\boldsymbol{\alpha}_B \tag{3.93}$$

z ograniczeniami:

$$0 \leq \alpha_{B,i} \leq C \quad \forall\, i \in B, \tag{3.94}$$
$$\mathbf{y}_B^\top \boldsymbol{\alpha}_B = \Delta - \mathbf{y}_R^\top \boldsymbol{\alpha}_R, \tag{3.95}$$

gdzie $\begin{bmatrix} Q_{BB} & Q_{BR} \\ Q_{RB} & Q_{RR} \end{bmatrix}$ jest pewną permutacją macierzy Q.

Zasadniczą ideą dekompozycji jest podział całego zbioru uczącego na dwa: roboczy B i resztę zbioru $R = \{1, \ldots, N\} \setminus B$. Wtedy wektor $\boldsymbol{\alpha}_B$ podlega optymalizacji, a wektor $\boldsymbol{\alpha}_R$ jest zamrożony. Natomiast funkcja celu przyjmuje postać 3.93 (czynnik $-\frac{1}{2}\boldsymbol{\alpha}_R^\top Q_{RR}\boldsymbol{\alpha}_R + p_R^\top \boldsymbol{\alpha}_R$ jest stały i dlatego nie znalazł się w 3.93).

Ogólny schemat algorytmu rozwiązywania problemu optymalizacji przez dekompozycję może wyglądać jak poniżej:

1. Wyznaczyć wektor $\boldsymbol{\alpha}^1$ jako rozwiązanie początkowe.
2. Jeśli wektor $\boldsymbol{\alpha}^k$ jest optymalny: **STOP**. W przeciwnym przypadku wyznaczyć zbiór roboczy (indeksów) $B \subset \{1, \ldots, N\}$ rozmiaru q, $R = \{1, \ldots, N\} \setminus B$. Wektory $\boldsymbol{\alpha}_B^k$ i $\boldsymbol{\alpha}_R^k$ są odpowiednimi, względem zbiorów indeksów B i R, częściami wektora $\boldsymbol{\alpha}^k$.
3. Rozwiązać podproblem optymalizacyjny 3.93 względem $\boldsymbol{\alpha}_B^k$.
4. $\boldsymbol{\alpha}_B^{k+1}$ ustawić na rozwiązanie optymalne 3.93, a $\boldsymbol{\alpha}_R^{k+1}$ na $\boldsymbol{\alpha}_R^k$. Następnie skok do 2.

Powyższy schemat wymaga jednak sprecyzowania odpowiedzi na trzy pytania:

1. Jak wybierać zbiór roboczy?
2. Na jakiej podstawie decydować, czy wektor $\boldsymbol{\alpha}^k$ jest optymalny?
3. Jak rozwiązywać podproblemy 3.93?

Dwa najciekawsze algorytmy korzystające z powyższej dekompozycji problemu to SVM^{light} Joachimsa [142] i SMO Platta [205] z modyfikacjami Keerthiego i in. [154]. SVM^{light} w zbiorze roboczym umieszcza parzystą liczbę elementów. Najczęściej jest to kilka elementów (w implementacji domyślnie 10). Można

zauważyć, że SVM^{light} dla zbioru roboczego wielkości 2 jest równoważny algorytmowi SMO z poprawkami Keerthiego [168].

Oba algorytmy korzystają tak naprawdę z tej samej transformacji algorytmu dekompozycji:

$$\min_{d} \quad \nabla g(\boldsymbol{\alpha})^{\top}\mathbf{d} \tag{3.96}$$

z ograniczeniami:

$$\mathbf{y}^{\top}\mathbf{d} = 0, \tag{3.97}$$
$$-1 \leq d \leq 1, \tag{3.98}$$
$$d_i \geq 0 \quad \text{gdy} \quad \alpha_i = 0, \tag{3.99}$$
$$d_i \leq 0 \quad \text{gdy} \quad \alpha_i = C, \tag{3.100}$$
$$|\{d_i | d_i \neq 0\}| = q, \tag{3.101}$$

gdzie $g(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^{\top}Q\boldsymbol{\alpha} + \mathbf{p}^{\top}\boldsymbol{\alpha}$. Z 3.101 łatwo zauważyć, że minimalizacja dotyczy tylko zbioru roboczego.

Choć tu tak naprawdę kończą się podobieństwa. SVM^{light} musi korzystać z procedur programowania kwadratowego do rozwiązania 3.96. Natomiast Platt w algorytmie SMO zaproponował analityczne rozwiązanie problemu w przypadku dwuelementowego zbioru roboczego. Dzięki temu nie zachodzi już potrzeba używania wewnętrznie procedury rozwiązywania ogólnego problemu QP. To jest właśnie naprawdę dużą zaletą, ponieważ standardowe procedury QP są wrażliwe numerycznie i skomplikowane w implementacji (setki linii kodu).

3.7.2. Wybór zbioru roboczego dla C-SVM

Pierwotny wybór zbioru roboczego w SMO zaproponowany przez Platta [205] został zmieniony na bardziej efektywny przez wspomnianego już kilkakrotnie Keerthiego [154]. Mechanizm wyboru zbioru roboczego $B = \{i, j\}$ indeksów mnożników zdefiniowany jest poniżej:

$$\begin{aligned} i \;=\; \arg\min_k \;\{\; & \nabla g(\boldsymbol{\alpha})_k d_k \;|y_k d_k = 1 \wedge\, , \\ & d_k \geq 0 \text{ gdy } \alpha_k = 0 \;\vee d_k \leq 0 \text{ gdy } \alpha_k = C\}, \end{aligned} \tag{3.102}$$

$$\begin{aligned} j \;=\; \arg\min_k \;\{\; & \nabla g(\boldsymbol{\alpha})_k d_k \;|y_k d_k = -1 \wedge\, , \\ & d_k \geq 0 \text{ gdy } \alpha_k = 0 \;\vee d_k \leq 0 \text{ gdy } \alpha_k = C\}. \end{aligned} \tag{3.103}$$

Powyższy wybór indeksów i i j wybiera te, które *najbardziej łamią* warunki KKT.

3.7.3. Kryterium stopu

Aby zdefiniować kryterium stopu zdefiniujemy pomocniczo g_i i g_j:

$$g_i = \begin{cases} -\nabla g(\boldsymbol{\alpha})_i, & \text{gdy } y_i = 1 \ \wedge \ \alpha_i < C \\ \nabla g(\boldsymbol{\alpha})_i, & \text{gdy } y_i = -1 \ \wedge \ \alpha_i > 0 \end{cases}, \tag{3.104}$$

$$g_j = \begin{cases} -\nabla g(\boldsymbol{\alpha})_j, & \text{gdy } y_j = -1 \ \wedge \ \alpha_j < C \\ \nabla g(\boldsymbol{\alpha})_j, & \text{gdy } y_j = 1 \ \wedge \ \alpha_j > 0 \end{cases}. \tag{3.105}$$

Kiedy nierówność

$$g_i \leq -g_j \tag{3.106}$$

jest spełniona, wektor $\boldsymbol{\alpha}$ jest wektorem optymalnym. W praktyce dopuszcza się pewną niedokładność, absolutnie nie zmieniając w istotny sposób ostatecznego rozwiązania:

$$g_i \leq -g_j + \epsilon, \tag{3.107}$$

gdzie ϵ jest bardzo małą dodatnią wartością (najczęściej stosuje się 0.001).

3.7.4. Wybór zbioru roboczego dla ν-SVM

Powyższy sposób (podrozdział 3.7.2) doboru zbioru roboczego jest odpowiedni dla C-SVC i C-SVR, nie jest jednak odpowiedni dla wersji ν-SVC i ν-SVR.

Informacje na temat wariacji wyboru zbioru roboczego dla problemów regresji można znaleźć w [166].

Autorem wyboru zbioru roboczego jest również Keerthi i Gilbert [154], ale warto zajrzeć również do publikacji Changa i Lina [40, 41].

Dla $\nabla g(\boldsymbol{\alpha})_i < \nabla g(\boldsymbol{\alpha})_j$ mamy więc:

$$i = \arg \min_k \{\nabla g(\boldsymbol{\alpha})_k d_k \mid y_k = 1 \ \wedge \ \alpha_k < C\}, \tag{3.108}$$

$$j = \arg \max_k \{\nabla g(\boldsymbol{\alpha})_k d_k \mid y_k = 1 \ \wedge \ \alpha_k > 0\}, \tag{3.109}$$

a gdy $\nabla g(\boldsymbol{\alpha})_i \geq \nabla g(\boldsymbol{\alpha})_j$ mamy:

$$i = \arg\min_k \{\nabla g(\boldsymbol{\alpha})_k d_k \mid y_k = -1 \ \wedge \ \alpha_k < C\}, \tag{3.110}$$

$$j = \arg\max_k \{\nabla g(\boldsymbol{\alpha})_k d_k \mid y_k = -1 \ \wedge \ \alpha_k > 0\}. \tag{3.111}$$

3.7.5. Kryterium stopu dla ν-SVM

Dla ν-SVM nie tylko wybór zbioru roboczego jest nieco inny, ale również kryterium stopu :

$$\max\{-G_1 + G_2, -G_3 + G_4\} < \epsilon, \tag{3.112}$$

gdzie $G_1 = \nabla g(\boldsymbol{\alpha})_i$ dla i z równania 3.108, $G_2 = \nabla g(\boldsymbol{\alpha})_j$ dla j z równania 3.109, $G_3 = \nabla g(\boldsymbol{\alpha})_i$ dla i z równania 3.110, $G_4 = \nabla g(\boldsymbol{\alpha})_j$ dla j z równania 3.111.

3.7.6. Analityczne rozwiązanie problemu dekompozycji

Platt w [205] zaproponował analityczne rozwiązanie problemu dekompozycji 3.93. Szczegóły wyprowadzenia (choć nieskomplikowane) wykraczają poza ramy tego opracowania, dokładną analizę można znaleźć w [205]. Analityczne rozwiązanie stało się możliwe po zmniejszeniu zbioru roboczego B z 3.93 do dwóch elementów. Wtedy problem optymalizacyjny przyjmuje postać:

$$\min_{\alpha_i,\alpha_j} \quad \frac{1}{2}[\alpha_i\ \alpha_j]\begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ji} & Q_{jj} \end{bmatrix}\begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} + (Q_{i,R}\alpha_R - 1)\alpha_i + (Q_{j,R}\alpha_R - 1)\alpha_j \tag{3.113}$$

z ograniczeniami:

$$y_i\alpha_i + y_j\alpha_j = \Delta - y_R^\top \alpha_R, \tag{3.114}$$

$$0 \le \alpha_i, \alpha_j \le C. \tag{3.115}$$

Jak wiadomo wartości mnożników $\boldsymbol{\alpha}$ są ograniczone (patrz 3.45), czyli wartości α_i i α_j muszą leżeć w kwadracie $[0, C] \times [0, C]$. Natomiast równości 3.45 wymuszają położenie punktu określonego przez mnożniki α_i i α_j na diagonali.

Zależnie od wartości y_i i y_j można wyznaczyć ograniczenia dolne i górne na wartości nowe mnożnika α_j^{new}. Dla $y_i \neq y_j$ mamy więc:

$$L = \max\{0, \alpha_j - \alpha_i\}, \tag{3.116}$$

$$H = \min\{C, C + \alpha_j - \alpha_i\}, \tag{3.117}$$

natomiast gdy $y_i \neq y_j$ mamy:

$$L = \max\{0, \alpha_i + \alpha_j - C\}, \tag{3.118}$$

$$H = \min\{C, \alpha_i + \alpha_j\}. \tag{3.119}$$

Z 3.114 można wyznaczyć α_i:

$$\alpha_i = y_i(\Delta - y_R^\top \alpha_R - y_j\alpha_j). \tag{3.120}$$

Po podstawieniu 3.120 i minimalizacji bez ograniczeń 3.113 otrzymuje się:

$$\alpha^* = \begin{cases} \alpha_j + \frac{-G_i - G_j}{Q_{ii} + Q_{jj} + 2Q_{ij}} & y_i \neq y_j, \\ \alpha_j + \frac{G_i - G_j}{Q_{ii} + Q_{jj} - 2Q_{ij}} & y_i = y_j, \end{cases} \tag{3.121}$$

gdzie G_i i G_j są zdefiniowane przez:

$$G_i = \nabla g(\boldsymbol{\alpha})_i, \tag{3.122}$$

$$G_j = \nabla g(\boldsymbol{\alpha})_j. \tag{3.123}$$

Teraz można wyznaczyć ostateczną nową wartość α_j, α_j^{new}:

$$\alpha_j^{new} = \begin{cases} H & \alpha^* \ge H \\ \alpha_* & L < \alpha^* < H \\ L & \alpha^* \le L \end{cases}. \tag{3.124}$$

Natomiast nową wartość α_i wyznaczamy przez:

$$\alpha_i^{new} = \alpha_i + y_i y_j (\alpha_j - \alpha_i^{new}). \tag{3.125}$$

3.7.7. Wyznaczenie wartości b i ρ

Po zakończeniu procesu optymalizacji należy jeszcze wyznaczyć wartości b i ρ. Co prawda wyznaczenie wartości ρ nie jest konieczne, ale wartość b jest niezbędna w funkcji decyzyjnej.

Zdefiniujmy r_+ i r_- zgodnie z [40, 41]:

$$r_+ = \frac{\sum_{0<\alpha_i<C, y_i=1} \nabla g(\boldsymbol{\alpha})_i}{\sum_{0<\alpha_i<C, y_i=1} 1}, \tag{3.126}$$

$$r_- = \frac{\sum_{0<\alpha_i<C, y_i=-1} \nabla g(\boldsymbol{\alpha})_i}{\sum_{0<\alpha_i<C, y_i=-1} 1}. \tag{3.127}$$

Jeśli nie ma żadnego i, dla którego $y_i = \pm 1$ i α_i spełnia $0 < \alpha_i < C$ to:

$$r_\pm = \frac{1}{2}\left[\max_{\alpha_i=C, y_i=\pm 1} \nabla g(\boldsymbol{\alpha})_i + \min_{\alpha_i=0, y_i=\pm 1} \nabla g(\boldsymbol{\alpha})_i\right]. \tag{3.128}$$

Mając r_+ i r_- można zdefiniować b i ρ:

$$\rho = \frac{r_+ + r_-}{2}, \tag{3.129}$$

$$b = \frac{r_+ - r_-}{2}. \tag{3.130}$$

3.7.8. Dalsze sposoby przyspieszenia rozwiązywania problemów QP dla SVM

Macierz Q nie zawsze można przechowywać w pamięci. Jeśli wielkość Q uniemożliwia przechowywanie całej macierzy to można *próbować* przechowywać te jej części, które potencjalnie mogą być przydatne w kolejnych krokach optymalizacji. Sposób ten (*ang. caching*) zaproponowany został przez Joachimsa [142]. Praktycznie im bliżej końca procedury optymalizacji, tym mniej kolumn macierzy Q jest używanych i tym większe prawdopodobieństwo, że odpowiednie z nich zostaną zatrzymane w pamięci. Dlatego warto zostawiać ostatnio używane kolumny macierzy Q.

Shrinking to następny sposób na usprawnienie algorytmu optymalizacji. Polega on na zmniejszaniu zbioru, który jest używany w procesie optymalizacji. Jak sugeruje Joachims w [142] tylko *wolne* wektory SV mogą brać czynny udział w procedurze optymalizacji. Dokładniej jeszcze definiuje to Lin w [170], gdzie pokazuje, że w końcowych iteracjach te α_i, które mogą ulec zmianie należą do

zbioru:

$$\begin{aligned}\{i \mid -y_i \nabla g(\bar{\boldsymbol{\alpha}})_i &= \max(\max_{\bar{\alpha}_i<C, y_i=1} -\nabla g(\bar{\boldsymbol{\alpha}})_i \ , \ \max_{\bar{\alpha}_i>0, y_i=-1} \nabla g(\bar{\boldsymbol{\alpha}})_i) \\ &= \min(\min_{\bar{\alpha}_i<C, y_i=-1} \nabla g(\bar{\boldsymbol{\alpha}})_i \ , \ \min_{\bar{\alpha}_i>0, y_i=1} -\nabla g(\bar{\boldsymbol{\alpha}})_i)\}\end{aligned}, \tag{3.131}$$

gdzie $\lim_{k\to\infty} \boldsymbol{\alpha}^k = \bar{\boldsymbol{\alpha}}$ ($\bar{\boldsymbol{\alpha}}$ jest optymalnym rozwiązaniem, jak pokazano w [169]).

3.8. Zbieżność algorytmów dekompozycji QP

Nie tylko same algorytmy przeżywały swój rozwój, ale i twierdzenia na temat ich zbieżności.

Keerthi pokazał zbieżność algorytmu SMO w [153] bez konieczności użycia poprawek przez niego zaproponowanych (wystarcza, aby wybierać pary indeksów wektora mnożników, które nie spełniają warunków problemu). Zbieżność obejmuje wersję C-SVC i ν-SVC.

Asymptotyczną zbieżność SMO pokazał Lin w [169]. To twierdzenie gwarantuje spełnienie warunków KKT problemu.

Z kolei Lin w [168] pokazał liniową zbieżność algorytmu dekompozycji używanego przez SVM^{light} (czyli także SMO z poprawkami) zakładając, że macierz Q jest pozytywnie określona i problem nie jest zdegenerowany.

3.9. SVM a RBF

Niewątpliwe związki pomiędzy SVM a RBF ujawniają się już przy spojrzeniu na funkcję decyzyjną SVM 3.28 i z drugiej strony na równanie sieci RBF 2.2. Warto jednak choć zasygnalizować jeszcze silniejszy związek.

Girosi [105] pokazał równoważność pomiędzy metodą *Sparse Approximation* sieci typu RBF i SVM. Jedyną, choć niewątpliwie bardzo istotną zmianą w sieci RBF jest definicja funkcji błędu:

$$E[\mathbf{w}, \boldsymbol{\xi}] = \left|\left| f(\mathbf{x}) - \sum_{i=1}^{n} \xi_i w_i G_i(\mathbf{x}) \right|\right|_{L_2}^2 + \lambda \left(\sum_{i=1}^{n} \xi_i \right)^p, \tag{3.132}$$

gdzie ξ_i przyjmują wartości z przedziału $[0, 1]$, p jest stałą dodatnią, zazwyczaj równą 1.

Jak widać z powyższych własności modelu SVM, może on być skutecznie używany do konstrukcji sieci RBF.

3.10. Meta-SVM

W podrozdziale 3.8 pisałem o tym, że odpowiednie realizacje SVM są optymalne (podobnie jak i ogólna wersja opisanej przez Vapnika [250]). Jednak wszystko

w matematyce ma swoje założenia. Udowodnione optymalności we wspomnianym podrozdziale zakładają, że parametr C bądź ν są wybrane (stałe), podobnie jak parametry wybranej funkcji jądrowej (kernela). To jednak nie daje nam odpowiedzi, ani wskazówki, jak wybrać parametr C (lub ν) i parametry funkcji jądrowej tak, aby uzyskać największy poziom generalizacji (największą możliwą poprawność na zbiorze testowym odpowiadającym poszukiwanemu rozkładowi danych).

Jednak dzięki szybkości algorytmu SMO (z poprawkami), o którym można przeczytać powyżej (3.7), można pokusić się o uczenie tych parametrów przez walidację skośną.

Ręczny dobór parametrów czasem może nie być banalny i wymaga szeregu testów i obserwacji. Dlatego w [138] zaproponowaliśmy *meta*-poziom uczenia modelu SVM, który automatycznie wyznaczyłby niemal optymalne położenie parametru C i parametrów funkcji jądrowej, osiągając możliwie najwyższy poziom generalizacji. Meta-poziom uczenia, to poziom ponad zasadniczym algorytmem uczenia, który może nadzorować dobór parametrów uczenia algorytmu.

Poniżej zobaczymy, że można skonstruować algorytm, który automatycznie dobiera parametry i uzyskuje bardzo dobry poziom poprawności w predykcji.

W poniższym algorytmie wykorzystana zostanie walidacja skośna do poszukiwania parametrów uczenia. Wewnętrzna walidacja skośna będzie poszukiwała dokładnych modeli o możliwie małej złożoności. Wyznaczanie małych modeli powoduje, że końcowy model jest nie tylko szybszy w podejmowaniu decyzji (co może być ważne), ale w małym modelu trudniej o przeuczenie, co prosto prowadzi do lepszej poprawności [107, 195, 140] w predykcji. Jak zobaczymy w 3.10.2, modele o mniejszej złożoności będą niegorsze lub lepsze od bardziej złożonych.

3.10.1. Walidacja skośna stosowana do uczenia

Główna pętla algorytmu będzie wykonywała p-krotną walidację skośną do uczenia modeli SVM. W tym celu zbiór danych $\mathcal{S}$ dzielimy na p możliwie równych części $\mathcal{S}_i,\ i = 1, \ldots, p$. Każda iteracja walidacji skośnej będzie używała algorytmu SVM z innymi parametrami konfiguracyjnymi (różne wartości parametru C i parametrów funkcji jądrowej). Uczenie i-tej iteracji odbywa się na zbiorze $\hat{\mathcal{S}}_i = \bigcup_{k=1,\ldots,p,\ k\neq i} \mathcal{S}_k$. Natomiast walidacja jakości uczenia będzie się odbywać na zbiorze $\mathcal{S}_i$.

Do sprawdzenia różnych ustawień parametru C (3.31) i γ (3.8) definiujemy nad nimi siatkę, która pokrywa region $[C_{min}, C_{max}] \times [\gamma_{min}, \gamma_{max}]$, z krokiem C_{step} i γ_{step}.

Obserwując zmienności rezultatów podczas zmian C i γ łatwo zauważyć, że mają one raczej charakter wykładniczy, niż liniowy. Tym samym lepiej będzie, gdy C_{min}, C_{max}, γ_{min} i γ_{max} będą reprezentowały potęgi 2 niż wartości liniowe. Do testów z dalszej części siatka parametrów C i γ była zdefiniowana przez $[-5, 5] \times [-10, 3]$ z krokiem 1. Taka siatka parametrów pokrywa większość interesujących obszarów parametrów C i γ, co było obserwowane na wszystkich

przeprowadzonych testach przy założeniu, że dane były normalizowane.

Niech $[C_1, \ldots, C_m]$, $[\gamma_1, \ldots, \gamma_n]$ będą wektorami parametrów C i γ, które definiują powyższą siatkę. W każdej iteracji walidacji skośnej do uczenia dla każdego C_i i każdego γ_j zostanie skonstruowany niezależny model SVM. Każda k-ta iteracja zachowuje poprawności na zbiorach walidacyjnych dla każdej pary parametrów i i j w

$$Acc^k_{\mathcal{S}_k}[i,j], \tag{3.133}$$

liczby wektorów podpierających (*support vectors*) przechowywane są w $SV^k[i,j]$, a liczby niewolnych wektorów podpierających ($\alpha_i = C$) są przechowywane w $bSV^k[i,j]$.

Możemy teraz dla każdej macierzy $Acc^k_{\mathcal{S}_k}$ znaleźć parę $\langle i,j \rangle$ (odpowiednich wartości C_i i γ_j), dla której poprawność predykcji jest największa:

$$\langle p,q \rangle = \arg\max_{\langle i,j \rangle} \sum_k Acc^k_{\mathcal{S}_k}[i,j]. \tag{3.134}$$

Powyższe kryterium może być rozszerzone o dodatkowy składnik, który nakładałby pewne warunki na liczby wektorów podpierających i niewolnych wektorów podpierających:

$$\langle p,q \rangle = \arg\max_{\langle i,j \rangle} \sum_k \left[Acc^k_{\mathcal{S}_k}[i,j] - \frac{\alpha \cdot SV^k[i,j] + \beta \cdot bSV^k[i,j]}{|\hat{\mathcal{S}}_k|} \right], \tag{3.135}$$

gdzie $|\hat{\mathcal{S}}_k|$ oznacza moc zbioru $\hat{\mathcal{S}}_k$. Dodatkowy składnik w powyższym kryterium ma regularyzować schemat poszukiwania ostatecznego modelu SVM. Oczywiście dla $\alpha = 0$ i $\beta = 0$ oba powyższe kryteria są jednakowe. W praktyce sensownymi wartościami dla α i β są małe dodatnie wartości rzeczywiste (na przykład $\alpha = 0.15$ i $\beta = 0.05$).

Kryterium ze wzoru 3.135 narzuca niejako *pozbycie się* wektorów podparcia, jeśli tylko nie ulegnie (istotnemu) pogorszeniu predykcja poprawności. Jak można zobaczyć w podrozdziale 3.10.2 nierzadko rozmiar modelu uzyskanego dzięki kryterium 3.135 będzie znacznie mniejszy od rozmiaru osiągniętego kryterium ze wzoru 3.134), a także dla wartości standardowych C i γ (szczegóły patrz 3.10.2).

Aby zredukować koszty obliczeniowe, czyli całkowitą liczbę modeli SVM, które trzeba utworzyć, można rozłożyć algorytm wyznaczania optymalnych wartości C i γ na 2 bądź 3 fazy. W pierwszej fazie można zdefiniować siatkę parametrów 4×4 ($4C \times 4\gamma$). W drugiej fazie definiujemy dwie siatki tego samego rozmiaru środkami określonymi przez najlepsze dwie pary, zgodnie z kryterium 3.135. Najczęściej wystarczy, gdy w drugiej fazie siatka będzie miała wymiary 3×3. Tak zmodyfikowany algorytm wymaga zbudowania 48 (lub tzlko 34) modeli SVM, natomiast bazowy sposób dla siatki $[-5,5] \times [-10,3]$ z krokiem 1 wymagał 154 modeli. Rezultaty uzyskane na oba sposoby powinny być bardzo zbliżone.

3.10.2. Wyniki algorytmu Meta-SVM

Rezultaty z tabeli 3.2 to wyniki przeprowadzonych obliczeń dla problemów klasyfikacji baz danych z UCI repository [182]. Zbiory danych znacznie różniły się pod względem liczby wektorów (od kilkuset do kilku tysięcy), cech i klas. Informacje o liczbie wektorów, cech i klas dostępne są w tabeli 3.2. Dokładne opisy zbiorów są dostępne w [182].

Każdy zbiór przed wzięciem udziału w obliczeniach był normalizowany z odrzuceniem 5% danych przy wyznaczeniu parametrów normalizacji (wartości największej i najmniejszej dla danej cechy). Tam gdzie nie było wstępnego podziału na zbiór treningowy i testowy rezultaty są wynikiem uśrednienia 10 wyników z testu 10-krotnej walidacji skośnej. Także standardowe odchylenia poprawności zostały umieszczone w tabeli 3.2. W przypadku gdy istniał podział na część treningową i testową następuje uczenie i testowanie powtarzane także 10 razy (ze względu na wewnętrzną walidację skośną do uczenia). Następnie w tabeli zostały umieszczone uśrednione rezultaty.

W przypadku, gdy dany zbiór opisuje dane więcej niż dwuklasowe, budowany jest komitet *k*-klasyfikatorów (jeden model na jedną klasę, patrz rozdział 5.1). Jak wynika z [127], różnice przy innych dobrych rozwiązaniach są minimalne. Warto zwrócić uwagę, że w przypadku problemów wieloklasowych, każdy z podmodeli wyznacza dla siebie optymalne wartości C i γ, które mogą być znacznie różne dla różnych klas.

Ważne jest też to, że każdy test dla każdego zbioru był przeprowadzany z dokładnie takimi samymi ustawieniami modeli. Czyli nie były dobierane ręcznie **żadne** parametry.

Dla każdego testu w tabeli 3.2 średnia wartość poprawności dla zbioru testowego znajduje się w kolumnie TES, a dla zbioru treningowego w TRS. Kolumna *std* zawiera średnie odchylenie standardowej poprawności dla zbioru testowego i treningowego. Ostatnia kolumna zawiera informacje o średniej liczbie wektorów podpierających (SV).

Po prawej stronie nazwy zbioru w tab. 3.2 napis typu "(C/W/Ce)" informuje o liczbie klas (C), wektorów (W) i cech (Ce) danego zbioru testowego. Górny wiersz opisu rezultatów każdego ze zbiorów prezentuje wyniki uzyskane z $\alpha = 0.15, \beta = 0.05$, środkowy z $\alpha = 0, \beta = 0$ (por. 3.135), a ostatni dla SVM który był uczony z $C = 1$ and $\gamma = 0.1$ (tj. bez Meta-poziomu).

Każdy zbiór był testowany na trzy sposoby. Pierwszy wiersz (pod wierszem z nazwą zbioru) prezentuje rezultaty dla $\alpha = 0.15, \beta = 0.05$, środkowy dla $\alpha = 0, \beta = 0$ (patrz wyrażenie 3.135). Dolny wiersz został uzyskany po wykorzystaniu bazowego algorytmu SMO (z poprawkami) dla wartości parametrów $C = 1$ i $\gamma = 0.1$ (γ z 3.8). Takie wartości parametrów C i γ nie są przypadkowe i dają ciekawe rezultaty dla wielu zbiorów (choć, jak widać, nierzadko nie są optymalne).

Siatka parametrów z podrozdziału 3.10 była zawsze taka sama: $C \in [-5, 5]$ z krokiem 1 i $\gamma \in [-10, 3]$ z krokiem 1. Każda wewnętrzna walidacja skośna była trzykrotna.

Zbiór	Poprawność (%)				#SV
	TES	std	TRS	std	
APPENDICITIS (2/106/7)					
(0.15, 0.05)	**86.69**	0.09	89.69	0.009	32
(0, 0)	**86.26**	0.09	90.89	0.15	46
–	**87.71**	0.082	88.88	0.009	35
AUSTRALIAN CREDIT (2/690/14)					
(0.15, 0.05)	**84.97**	0.039	89.8	0.016	211
(0, 0)	**85.68**	0.04	86.49	0.017	383
–	**85.34**	0.04	89.79	0.005	260
DIABETES (2/768/8)					
(0.15, 0.05)	**76.7**	0.046	79.2	0.014	367
(0, 0)	**77.15**	0.044	78.7	0.009	394
–	**76.6**	0.045	79.6	0.006	387
DNA (3/2000+1180/60)					
(0.15, 0.05)	**94.23**	0.0019	99.41	0.003	1268
(0, 0)	**94.27**	0.0038	99.84	0.001	2379
–	**64.58**	1e-16	1	0	5803
FLAG (8/193/28)					
(0.15, 0.05)	**42.2**	0.098	71.2	0.074	405
(0, 0)	**41.6**	0.11	77.7	0.063	513
–	**32.8**	0.098	74.2	0.018	867
GLASS (6/214/9)					
(0.15, 0.05)	**65.2**	0.09	87.34	0.036	293
(0, 0)	**64.26**	0.09	87.55	0.038	424
–	**56.30**	0.056	63.4	0.02	394
HEART (2/303/13)					
(0.15, 0.05)	**82.54**	0.07	86.48	0.009	118
(0, 0)	**82.5**	0.073	85.7	0.018	142
–	**80.86**	0.076	89.2	0.005	146
HEPATITIS (2/155/19)					
(0.15, 0.05)	**82.73**	0.072	94	0.031	66
(0, 0)	**80.8**	0.087	94.54	0.033	73
–	**79.38**	6.6	96	0.01	97
IONOSPHERE (2/200+151/34)					
(0.15, 0.05)	**98**	0	99	0	73
(0, 0)	**98**	0	99	0	125
–	**98**	0	95	0	100

Tabela 3.2: Porównanie rezultatów uczenia algorytmu Meta-SVM. Dokładny opis znajduje się w tekście.

Zbiór	Poprawność (%)				#SV
	TES	std	TRS	std	
KR-VS-KP (2/3196/36)					
$(0.15, 0.05)$	**99.65**	0.003	99.93	0.0006	302
$(0, 0)$	**99.53**	0.046	99.95	0.0004	517
–	**96.76**	0.008	97.1	0.001	697
SONAR (2/208/60)					
$(0.15, 0.05)$	**87.02**	0.073	99.78	0.005	104
$(0, 0)$	**87.04**	0.074	99.8	0.004	125
–	**78.12**	0.088	83.92	0.015	151
WISCONSIN BREAST (2/699/9)					
$(0.15, 0.05)$	**96.54**	0.004	97.45	0.003	59
$(0, 0)$	**96.68**	0.004	97.17	0.006	110
–	**96.81**	0.019	97.29	0.0021	76
VOTES (2/435/16)					
$(0.15, 0.05)$	**94.25**	0.034	97.54	0.009	89
$(0, 0)$	**94.5**	0.03	96.9	0.008	105
–	**90.6**	0.045	99.2	0.002	245
VOWEL (6/871/3)					
$(0.15, 0.05)$	**85.3**	0.036	90.1	0.009	579
$(0, 0)$	**85.0**	0.034	90.4	0.01	969
–	**72.5**	0.042	73.1	0.007	1158

Tabela 3.3: Porównanie rezultatów uczenia algorytmu Meta-SVM cd.

Czasami zdarza się, że rezultaty dla $C = 1$ i $\gamma = 0.1$ są lepsze, niż dla Meta-SVM. Powodem jest to, że, jak już pisałem, wartości $C = 1$ i $\gamma = 0.1$ nie są przypadkowe i czasem rzeczywiście bardzo odpowiednie dla danego zbioru danych. Natomiast znalezienie ich przez Meta-SVM jest nietrywialne, głównie ze względu na to, że dotyczy to zbiorów o małej liczbie wektorów.

Rysunki .1 i .2 umieszczone na stronie 306 prezentują wyniki dwóch testów: *wisconsin breast cancer* (rys. .1) i *glass* (rys. .2). Pokazywane są cztery rysunki z konturami stałych wartości. Pierwszy (lewy górny róg) pokazuje kontury zgodnie z kryterium Meta-SVM (3.135). Drugi (górny prawy róg) prezentuje predykcję poprawności, a kontury 3 i 4 odpowiadają liczbą wektorów podpierających (SV) i niewolnych wektorów podpierających (bSV) (rogi dolne, lewy i prawy odpowiednio). Czerwony kwadrat pokazuje punkt optymalny ze względu na kryterium 3.135 (o wartościach: $\log_2 C$ i $\log_2 \gamma$).

Jak pokazują przedstawione wyniki złożoność modeli warto kontrolować.

3.10.3. Podsumowanie

Warto na koniec zwrócić uwagę już tylko na jedną cechę algorytmu Meta-SVM. Nie wymaga on żadnego ręcznego doboru parametrów, a znajdowane rozwiązania są optymalne lub bliskie są optymalnym. Wystarczy go tylko uruchomić. Taka wersja modelu SVM została zaimplementowana w systemie GhostMiner [139].

Rozdział 4

Ontogeniczne modele sieci neuronowych

Do ontogenicznych modeli sieci neuronowych zalicza się takie modele, które mogą zmieniać swoją strukturę w procesie uczenia się. Struktura sieci to po prostu układ warstw neuronów i połączeń pomiędzy neuronami, co wraz z funkcjami transferu, odpowiadającymi neuronom, determinuje możliwości całego modelu adaptacyjnego[1]. Dzięki automatycznemu doborowi struktury sieci przez algorytm uczący został wyeliminowany problem *ręcznego* doboru liczby warstw jak i liczby neuronów. Co prowadzi do znacznie bardziej automatycznego i efektywnego procesu uzyskiwania dobrze działających sieci neuronowych.

Struktura sieci neuronowej	+	Funkcje Transferu	⇒	Złożoność Modelu

Modele, które nie mogą modyfikować swojej struktury muszą ją mieć dobrze ustaloną na podstawie wiedzy *a priori* przed rozpoczęciem procesu uczenia. To niestety jest często bardzo trudne, ponieważ zazwyczaj nie jest znana złożoność problemu z jakim mamy do czynienia i trudno wtedy oszacować dobrze liczbę warstw i neuronów. Najczęściej stosuje się metodę prób i błędów testując różne konfiguracje struktury sieci.

Modele ontogeniczne, których struktura ewoluuje w czasie procesu uczenia, muszą posiadać mechanizmy oceny wystarczalności bądź niewystarczalności swojej struktury, a także jej nadmiarowości . Daną strukturę sieci neuronowej można uznać za wystarczającą jeśli dla wybranego algorytmu uczenia można przeprowadzić proces uczenia z sukcesem (np. uzyskać zakładany próg poprawności). Struktura sieci neuronowej jest nadmiarowa, gdy dla innej mniejszej

[1] Abstrahując od algorytmu adaptacji sieci.

struktury zgodnej z wybranym algorytmem uczenia możemy uzyskać nie gorszą wynikową sieć neuronową (np. uzyskującą zakładany próg poprawności). Jakość funkcji oceny struktury sieci neuronowej bardzo silnie warunkuje możliwości kolejnych części procesu uczenia sieci neuronowej i tym samym w bardzo istotny sposób wpływa na ostateczne możliwości modelu. Jeśli tylko mechanizm oceny struktury nie jest zbyt kosztowny obliczeniowo, to algorytm uczenia może niemal na bieżąco korygować strukturę sieci i dopasowywać ją do zaistniałej sytuacji.

Oprócz mechanizmu oceny struktury sieci musi istnieć także metoda korekcji struktury, w związku z niewystarczalnością bądź nadmiarowością struktury sieci neuronowej.

Należy także zwrócić uwagę, że dla sieci nieontogenicznych nawet jeśli uda się dobrać *a priori* poprawną wielkość struktury, to sam proces uczenia może nie być łatwy, gdy nie jest on dobrze zainicjowany (jako przykład można podać nierzadkie kłopoty ze zbieżnością algorytmu wstecznej propagacji). Natomiast start z mniejszej struktury i jej sukcesywne powiększanie może okazać się znacznie łatwiejsze.

$$\boxed{\begin{array}{c}\text{Złożoność}\\ \text{Modelu}\end{array} \approx \begin{array}{c}\text{Złożoność}\\ \text{Problemu}\end{array}}$$

Związek złożoności problemu i złożoności modelu nie jest nowy, był on stawiany od dawna przy okazji problemów estymacji i aproksymacji. Zależności pomiędzy złożonością problemu i złożonością modelu wynikają w znacznej mierze ze związku pomiędzy wariancją modelu i jego obciążeniem (w literaturze angielskiej problem znany jest głównie jako *bias–variance dilemma*).

Z kolei **związek pomiędzy obciążeniem modelu i jego wariancją** wynika prosto z najczęściej używanej miary błędu używanej w aproksymacji czy estymacji, czyli błędu średniokwadratowego. Należy spojrzeć na wartość oczekiwaną takiego błędu względem wszystkich możliwych zbiorów uczących $\mathcal{S}$. Załóżmy, że $f(\mathbf{x};\mathbf{S})$ oznacza wartość estymowaną przez pewien model $f(\cdot)$ nauczony na zbiorze $\mathcal{S}$, natomiast $F(\cdot)$ oznaczać będzie nieznane odwzorowanie, które ma być estymowane przez model $f(\cdot)$. Wtedy można dokonać dekompozycji wartości oczekiwanej kwadratu błędu na część związaną z kwadratem obciążenia i wariancją modelu:

$$E[(f(\mathbf{x};\mathbf{S})-F(\mathbf{x}))^2] = \underbrace{(E[f(\mathbf{x};\mathbf{S})-F(\mathbf{x})])^2}_{\text{obciążenie}^2} + \underbrace{E[(f(\mathbf{x};\mathbf{S})-E[f(\mathbf{x};\mathbf{S})])^2]}_{\text{wariancja}} \quad (4.1)$$

Jak widać z powyższego równania oba człony prawej strony mają dodatni wkład do końcowej wartości oczekiwanej błędu. Dlatego też celem powinno być znalezienie złotego środka pomiędzy wartością obciążenie i wartością wariancji. Małe wartości obciążenie to dokładne odwzorowanie zbioru uczącego jednak

powodują duże wartości wariancji (jak i przeuczenie). Małe wartości wariancji oznaczają z kolei duży błąd estymacji, czyli duże obciążenie.

Łatwo to zilustrować na przykładzie aproksymacji przy użyciu wielomianów. Gdy stopień wielomianu będzie duży obciążenie będzie małe jednak wariancja będzie duża i mogą wystąpić spore problemy z generalizacją. Gdy wielomian będzie niskiego stopnia (czy wręcz za niskiego) wtedy obciążenie będzie wysokie (niemały błąd na zbiorze uczącym), ale mała wariancja.

Analogiczna sytuacja zachodzi w kontekście sieci neuronowych. Miejsce wielomianów zajmują funkcje transferu, np. jedna z radialnych funkcji transferu. Wtedy gdy sieć jest za mała nie może nauczyć się odpowiednio dobrze interpolować dane ze zbioru uczącego, natomiast gdy jest za duża uczy się łatwo lecz nie wykazuje dobrej poprawności na zbiorze testowym (brak generalizacji).

Stąd wyraźny i jednoznaczny związek pomiędzy obciążeniem/variancją a złożonością modelu i problemu. Dlatego właśnie tak istotne jest aby dokonać jak najodpowiedniejszego doboru struktury sieci.

CEL:	Jak najlepsza GENERALIZACJA

Z powyższych rozważań widać, że celem nie powinna być minimalna liczba neuronów, czy połączeń między neuronowych, lecz znalezienie takiej struktury sieci, która umożliwi uzyskanie jak największego poziomu generalizacji (czyli jak największej poprawności na danych walidacyjnych reprezentujących ten sam problem). Dlatego, aby sieć mogła ewoluować w kierunku uzyskania jak najlepszej generalizacji, musi posiadać pewien margines swobody, który leży w parametrach adaptacyjnych i pozwala zmieniać płynnie stany modelu adaptacyjnego.

Dobrze dobrane marginesy swobody wraz z kryteriami kontroli złożoności modelu, pozwalają także walczyć z problemem lokalnych minimów. Model zmieniając swoją strukturę przechodzi do innych funkcji błędów, z pomocą których następuje kontynuacja procesu uczenia, po czym mogą nastąpić kolejne zmiany struktury sieci. Taki sposób postępowania algorytmu daje możliwość penetracji pewnej przestrzeni funkcji błędu. Jest to nie co podobne do poszukiwania struktur sieci za pomocą programowania genetycznego [219], jednak w tym przypadku proces jest mniej losowy i zawsze (z dokładnością do jakości funkcji oceniających wystarcaalność/niewystarczalność struktury) wykonuje kroki, które powinny przynieść poprawę (programowanie genetyczne dopuszcza w procesie ewolucji pewną losować)[2].

Wielką rolę w całym mechanizmie odgrywają mechanizmy kontroli złożoności struktury sieci, które powinny jak najefektywniej czerpać informacje z przebiegu procesu uczenia, ze zbioru treningowego jak i wszelkiej dostępnej wiedzy

[2]Nie oznacza to jednak, że programowanie genetyczne nie może prowadzić do lepszych rozwiązań.

a priori. Na przykład, dodając do modelu informacje *a priori* o dokładności danych (tj. dokładności dokonanych pomiarów), na których opiera się proces adaptacji modelu, możemy doprowadzić do znacznie innych, zazwyczaj lepszych, wyników niż te, które można by uzyskać, nie dodając takiej informacji.

W przeciągu ostatnich lat powstały różnorodne modele, które modyfikują swoją architekturę. Wszystkie metody kontroli złożoności architektur sieci można podzielić na następujące grupy:

- powiększające — do tych modeli należą algorytmy, umożliwiające dokładanie nowych neuronów, nowych połączeń pomiędzy neuronami lub umożliwiające dodawanie neuronów i połączeń, a także sieci, których liczba warstw również może rosnąć;

- zmniejszające — do których z kolei należą metody, które pozwalają na usuwanie zbędnych neuronów czy połączeń miedzy neuronami lub algorytmy, które potrafią konkatenować grupy neuronów lub połączenia pomiędzy neuronami[3];

- powiększająco-zmniejszające — te metody łączą w sobie dwie powyższe grupy umożliwiając powiększanie się struktury sieci jak i jej zmniejszanie w miarę potrzeb;

- komitety modeli — te systemy, to grupy modeli, z których każdy indywidualnie ma za zadanie rozwiązywać ten sam problem lub jakąś jego część, a następnie system zarządzający decyduje jaka będzie ostateczna decyzja. System zarządzający może automatycznie dobierać nie tylko liczbę modeli, ale i odpowiednie kombinacje stworzonych modeli (to właśnie stanowi o związku komitetów i modeli ontogenicznych). Tej grupie modeli poświęcony został oddzielny rozdział (rozdział 5).

Najczęściej jednak spotyka się systemy, które należą do jednej z powyższych grup, czyli gdy mają możliwość powiększania swojej struktury, często startując z pustej struktury (lub bardzo prostej), to zazwyczaj nie usuwają neuronów ani połączeń pomiędzy neuronami. Stosunkowo rzadko spotyka się systemy, które mogą powiększać i pomniejszać swoją strukturę. Z kolei komitety najczęściej komponowane są z stosunkowo prostych modeli.

PYTANIA:	Czy/Kiedy	usuwać dodawać	?
	Co	usuwać dodawać	?

[3]Konkatenację wag czasem określa się współdzieleniem wag.

Jednakże najważniejszą część modeli ontogenicznych stanowią kryteria, które decydują o tym, czy i/lub kiedy należy dokonać zmian w strukturze modelu. Z kolei inne kryteria umożliwiają ustalenie, której części struktury ma dotyczyć dana zmiana oraz jak jej dokonać.

Od skuteczności tych kryteriów w prostej linii uzależnione jest osiągnięcie żądanego poziomu generalizacji lub totalnej klęski podczas uczenia. Należy zwrócić uwagę, iż zmiany dokonywane na już istniejącej strukturze nie powinny prowadzić do pogorszenia już istniejącego poziomu *reprezentacji* problemu, lecz umożliwić jej polepszenie w dalszej procedurze adaptacji. Z kolei wybór części struktury, która ma podlegać zmianie, neuron(-y) lub połączenie(-nia), też nie jest trywialny. To, iż dany neuron lub pewne połączenie w danej chwili nie odgrywa istotnej roli w sieci neuronowej wcale nie musi oznaczać jego zbędności. Czasami proces adaptacji może po prostu nie zdążyć wykorzystać owego neuronu lub połączenia. Innymi słowy uczenie może być jeszcze po prostu w dość intensywnej fazie.

Niektóre systemy ontogeniczne mogą sprostać jeszcze ciekawszym i trudniejszym problemom. Mogą mianowicie estymować rozkład danych (opisujących pewien problem) zmieniający się w czasie. Złożoność samego problemu może również podlegać zmianie, co wymaga bieżących zmian architektury sieci, tak, aby model mógł znów odzwierciedlać zmienioną złożoność problemu. Znacznie więcej informacji o estymacji rozkładów zmiennych można znaleźć w [151]. Warto także zwrócić uwagę na model rozwijany przez Fritzke [97]. Aby dany model uczący mógł estymować rozkłady, które zmieniają się w czasie, musi sam kontrolować wystarczalność lub niewystarczalność aktualnej w danej chwili struktury modelu, tak aby mógł ją zmieniać w zależności od obserwowanej złożoności problemu. Większość spotykanych algorytmów jednak albo tylko dopuszcza do usuwania neuronów podczas uczenia, albo też w modelach rozrastających się, neurony dodane nierzadko są zamrażane i nie podlegają dalszej adaptacji. Takie postępowanie ogranicza dość znacznie modele. W przyszłości do zaawansowanych symulacji systemów kognitywnych najbardziej przydatne będą właśnie takie systemy, które będą potrafiły estymować rozkłady zmienne. Bo przecież trudno wyobrazić sobie zaawansowane systemy kognitywne, które nie mogły by się rozwijać i kontynuować procesu adaptacji, który często może wymagać zmian w dotychczasowej, wyuczonej wiedzy.

4.1. Modele zmniejszające strukturę

Modele, które potrafią zmniejszać swoją strukturę poprzez usuwanie połączeń lub usuwanie neuronów, próbują wykorzystywać przeróżną informację jaka drzemie w samych danych, w aktualnym w danej chwili stanie sieci (wartości wag, progów, rozmyć, etc.), jak i stanie struktury sieci neuronowej.

Na samym początku należy wspomnieć o najprostszym chyba sposobie na usuwanie nieprzydatnych neuronów, który, jak i część z poniżej opisanych metod, wyznacza współczynniki *istotności* lub *przydatności* każdego neuronu w bie-

żącej strukturze sieci. W tym algorytmie są one opisane wzorem

$$s_i = E(\text{bez neuronu } i) - E(\text{z neuronem } i), \tag{4.2}$$

który wyznacza różnice pomiędzy błędem sieci uzyskanym *bez* i *z* udziałem neuronu *i*. Wyznaczone w ten sposób wartości s_i dla poszczególnych neuronów opisują ich przydatność w strukturze sieci. Jednak sposób ten wymaga dość sporych nakładów obliczeniowych — do wyznaczenia każdego współczynnika s_i równania (4.2) należy wyznaczyć błąd dla każdego wektora ze zbioru treningowego. Opierając się na powyżej zdefiniowanych współczynnikach istotności, można wybrać te neurony, dla których współczynniki te są znacznie mniejsze od innych, a następnie usunąć tak wybrane neurony.

Zbliżony (również pasywny[4]) sposób wyznaczania współczynników istotności został użyty w sieci neuronowej FSM (*ang. Feature Space Mapping*) [1, 64, 71, 65]. Jego krótki opis został zamieszczony w podrozdziale 4.2.7.

Współczynniki istotności wyznacza się dla każdego neuronu warstwy ukrytej po przerwaniu procesu uczenia w następujący sposób:

$$Q_i = \frac{C_i(\mathbf{X})}{|\mathbf{X}|}, \tag{4.3}$$

gdzie $|\mathbf{X}|$ jest liczbą wzorców uczących, $C_i(\mathbf{X})$ określa liczbę poprawnie sklasyfikowanych wzorców ze zbioru $\mathbf{X}$ przez *i*-ty neuron. W sieci FSM każdy neuron warstwy ukrytej odpowiada pewnemu obszarowi przestrzeni (tj. klastrowi), z którą związana jest informacja o jego przynależności do jakiejś klasy. Mając wyznaczone współczynniki Q_i, można już dokonać usunięcia tych neuronów, dla których wartości te są najbliższe zeru.

4.1.1. Modele zmniejszające strukturę a regularyzacja

O regularyzacji wspominałem już w rozdziale 2. W tym podrozdziale zobaczymy, że regularyzacja może być bardzo przydatna w zmniejszaniu struktury sieci neuronowych.

4.1.1.1. Rozpad wag

Metody zmniejszające strukturę sieci neuronowej często można rozpatrywać jako metody regularyzacji. Najbardziej wzorcowym przykładem może być rozpad wag (*ang. weight decay*) [120] przedstawiony już w podrozdziale 2.5.2, gdzie do miary błędu modelu $E_0(f)$ (2.14) zostaje dodany czynnik regularyzacyjny

$$E_{wd}(f, \mathbf{w}) = E_0(f) + \lambda \sum_{i=1}^{M} w_i^2. \tag{4.4}$$

[4]Przez pasywność rozumie się tu brak związku z samym algorytmem uczenia, jak i samą funkcją błędu.

Ustalając pewien próg θ, możemy dokonać usunięcia połączeń czy neuronów, których wartości wag są mniejsze od progu θ po zakończeniu lub przerwaniu procesu uczenia. Jednakże powyższa funkcja błędu przejawia tendencje do tego, aby istniało wiele małych wag i wręcz doprowadza do sytuacji, w której nie pozostaje nawet mała część wag z dużymi wartościami. Taka sytuacja jest spowodowana karą za **każdą** znaczącą wartość wagi, w tym i istotną wagę zgodnie z (4.4).

4.1.1.2. Eliminacja wag

Aby uciec od powyższego problemu Weigend (i in.) [252, 253] zaproponował eliminację wag (*ang. weight elimination*), stosując istotnie inny człon regularyzacyjny

$$E_{we}(f, \mathbf{w}) = E_0(f) + \lambda \sum_{i=1}^{M} \frac{w_i^2 / w_0^2}{1 + w_i^2 / w_0^2}, \tag{4.5}$$

gdzie w_0 jest pewnym ustalonym współczynnikiem. Tak sformułowany człon regularyzacyjny przestaje zmuszać, by wartości wszystkich wag były małe i pozwala, aby istniała pewna liczba wag przyjmujących wartości zbliżone lub znacznie większe od w_0. Wynika to z własności powyższego członu: waga w_i dla której $|w_i| >> w_0$ ma wkład bliski wartości λ, natomiast dla wag w_i dla których $|w_i| << w_0$, wkład jest bliski zeru.

Parametr λ może podlegać adaptacji podczas uczenia. Adaptacje można przeprowadzać raz na epokę, zgodnie z poniższym schematem:

- $\lambda = \lambda + \Delta\lambda$ gdy $E_n < D \vee E_n < E_{n-1}$
- $\lambda = \lambda - \Delta\lambda$ gdy $E_n \geq E_{n-1} \wedge E_n < A_n \wedge E_n \geq D$
- $\lambda = 0.9\lambda$ gdy $E_n \geq E_{n-1} \wedge E_n \geq A_n \wedge E_n \geq D$

E_n jest błędem ostatniej (n-tej) epoki uczenia dla całego zbioru treningowego, $A_n = \gamma A_{n-1} + (1 - \gamma)E_n$ ($A_0 = 0$, γ jest bliskie 1), natomiast D określa błąd docelowy dla całego procesu uczenia.

4.1.1.3. MLP2LN

Z kolei poniższy człon dodany do standardowej funkcji błędu $E_0(f)$ (2.14)

$$E_{mlp2ln}(f, \mathbf{w}) = E_0(f) + \lambda \sum_{i=1}^{M} w_i^2 (w_i - 1)^2 (w_i + 1)^2, \tag{4.6}$$

został użyty do budowania i uczenia sieci MLP2LN, służącej do ekstrakcji reguł logicznych [60].

Warto zwrócić uwagę, że metoda eliminacji wag czy rozpadu wag może być stosowana dla pewnego podzbioru wag sieci neuronowej. Można także używać tych metod dla oddzielnych grup wag w ramach jednej sieci neuronowej, na

przykład niezależnie dla warstw. Z kolei wykorzystanie takiej regularyzacji wag dla pierwszej warstwy wag w połączeniu z jedna z metod współdzielenia wag daje w rezultacie całkiem ciekawą metodę selekcji cech. Połączenie z metodą współdzielenia wag nie jest konieczne, może jednak zmniejszyć ostateczną liczbę cech.

4.1.1.4. Lokalna regresja grzbietowa

Innymi przykładami metod regresji, umożliwiających kontrolę struktury sieci, może być lokalna regresja grzbietowa (*ang. local ridge regression*), opisana już w podrozdziale 2.5.2, czy też metody współdzielenia wag poprzez różne neurony.

4.1.1.5. Metody współdzielenia wag

Inną grupę metod regularyzacyjnych stanowią metody współdzielenia wag (czy też konkatenacji wag). Ideą tych metod jest tworzenie grup wag, w ramach każdej grupy wszystkie wagi będą posiadały tą samą wartość (czasami wartości zbliżone). Jedną z takich metod jest metoda miękkiego współdzielenia wag (*ang. soft weight sharing*), która nie wymaga, aby wagi danej grupy były sobie równe, lecz dopuszcza pewien rozrzut [196] σ_j. Funkcja błędu z dodatkowym członem regularyzacyjnym przyjmuje wtedy postać

$$E_{sws}(f, \mathbf{w}) = E_0(f) - \lambda \sum_i^M \ln \left(\sum_{j=1}^k a_j \phi_j(w_i) \right), \tag{4.7}$$

gdzie $\phi(w_i)$ jest zdefiniowane poprzez

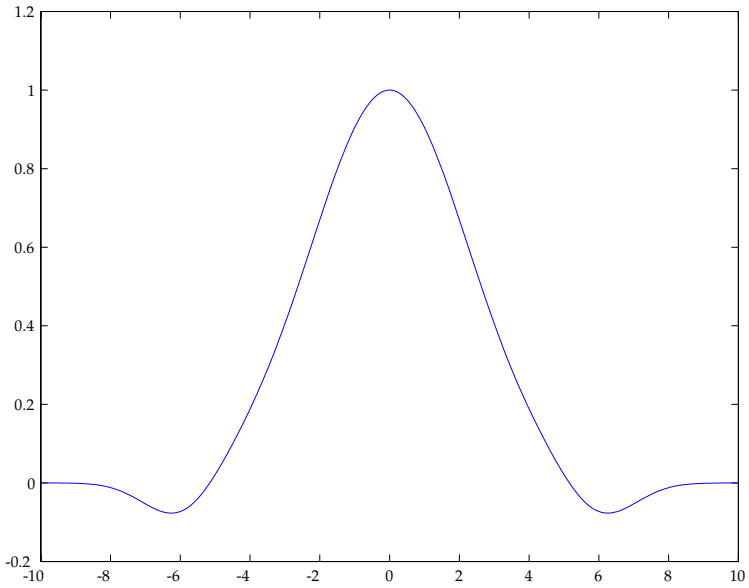
$$\phi_j(w) = \frac{1}{(2\pi\sigma_j^2)^{1/2}} \exp\left(-\frac{(w-\mu_j)^2}{2\sigma_j^2} \right). \tag{4.8}$$

Taki człon również może pełnić role nie tylko czysto regularyzacyjną, ale i umożliwiać usuwanie połączeń. Parametry σ_j podlegają adaptacji, co może doprowadzić do stanu w którym pewne grupy wag będą odpowiadały bardzo podobnym wagom, a w takim przypadku można dokonać ich ostatecznego połączenia.

4.1.2. Usuwanie wag metodami Optimal Brain Damage (OBD) i Optimal Brain Surgeon (OBS)

Niestety, nie zawsze użycie metody rozpadu wag, jak i eliminacji wag daje wystarczająco dobre rezultaty. Czasem aby uzyskać naprawdę mały błąd, niezbędne okazują się i małe wagi niektórych połączeń. Dla przykładu popatrzmy na rysunek 4.1 funkcji o kształcie meksykańskiego kapelusza. Aby dokonać aproksymacji takiej funkcji, można by użyć jednej funkcji gaussowskiej, jednakże wtedy

błąd aproksymacji nie będzie mały. Natomiast, gdy użyć trzech funkcji gaussowskich, w tym dwie miałyby znacznie mniejsze wagi, błąd aproksymacji byłby bardzo mały.



Rysunek 4.1: Meksykański kapelusz.

Z powyższych powodów celem pracy Le Cun'a (i in.)[51] było wyznaczenie parametrów *istotności* dla poszczególnych wag sieci neuronowej. Na podstawie tych parametrów można by dokonywać prób usuwania wag, których współczynniki istotności były najmniejsze, nie powodując istotnych zmian w funkcji błędu. Dlatego też Le Cun (i in.) nazwali tą metodę *Optimal Brain Damage* (OBD) [51].

Droga rozwiązania problemu wiedzie przez analizę różnicy funkcji błędu δE, jaka powstaje pod wpływem zaburzenia wektora wag sieci o $\delta \mathbf{w}$. Analizuje się funkcję błędu, rozwiniętą w szereg Taylora

$$\delta E = \left(\frac{\partial E}{\partial \mathbf{w}}\right)^T \cdot \delta \mathbf{w} + \frac{1}{2}\delta \mathbf{w}^T \cdot \frac{\partial^2 E}{\partial \mathbf{w}^2} \cdot \delta \mathbf{w} + O(||\delta \mathbf{w}||^3), \tag{4.9}$$

gdzie $\frac{\partial^2 E}{\partial \mathbf{w}^2}$ tworzy Hessian H.

Gdy proces adaptacji zbiegł się, czyli parametry wag znajdują się w minimum funkcji błędu, to pierwszy człon sumy prawej strony równania (4.9) można pominąć. Również i trzeci składnik dla uproszczenia można pominąć. Le Cun zakłada również, iż tylko przekątna macierzy Hessianu jest rzeczywiście istotna,

co prowadzi do uproszczenia równania (4.9) do postaci

$$\delta E = \frac{1}{2}\sum_{i}^{M} H_{ii}\delta w_i^2, \tag{4.10}$$

gdzie H_{ii} to i-ty element diagonalny macierzy Hessianu H. Z równania (4.10) widać, jak zmiana poszczególnych wag w_i może wpływać na zmiany funkcji błędu. W szczególnym przypadku można zobaczyć, jak wpływa usunięcie i-tej wagi. Wtedy $w_i = \delta w_i$, co prowadzi do wyznaczenia współczynników istotności s_i dla każdej wagi w_i

$$s_i = H_{ii} w_i^2. \tag{4.11}$$

Algorytm składa się wtedy z następujących etapów:

1. Wstępny wybór architektury,
2. Uczenie sieci do etapu, w którym zmiany funkcji błędu nie będą już istotne,
3. Wyznaczenie współczynników istotności zgodnie z (4.11),
4. Usunięcie wag, których współczynnik istotności jest niski,
5. Jeśli któraś z wag została usunięta, to następuje skok do punktu 2.

Metoda ta została pomyślnie użyta do rozpoznawania pisma ręcznego, uzyskując błąd rzędu 3.4% na zbiorze testowym (2700 wektorów). Sieć była uczona na 9840 wektorach. Sama sieć składała się z pięciu wyspecjalizowanych warstw [50].

Następnym ciekawym krokiem była metoda zaprezentowana przez Hassibiego i Storka nazwana Optimal Brain Surgeon [115, 116].

Ta metoda usuwania zbędnych połączeń, podobnie jak metoda OBD, używa rozwinięcia funkcji błędu w szereg Taylora (4.9). Jednakże Hassibi i Stork twierdzą, iż zazwyczaj macierz Hessianu wag jest niediagonalna w skutek czego może dochodzić do usuwania dobrych, czyli istotnych wag.

Metoda Optimal Brain Surgeon również zakłada, że pierwszą i trzecią część prawej strony równania (4.9) można zaniedbać. Tym samym, jako cel metody proponuje się minimalizację

$$\min_{q}\min_{\delta \mathbf{w}} \{\frac{1}{2}\delta \mathbf{w}^T H \delta \mathbf{w}\} \quad \text{pod warunkiem } \delta w_q + w_q = 0, \tag{4.12}$$

która następnie jest rozwiązywana za pomocą mnożników Lagrange'a

$$L = \frac{1}{2}\delta \mathbf{w}^T H \delta \mathbf{w} + \lambda(\delta w_q + w_q). \tag{4.13}$$

Co dalej prowadzi do rozwiązania i wyznaczenia istotności dla wag s_q:

$$s_q = \frac{1}{2}\frac{w_q^2}{H_{qq}^{-1}} \tag{4.14}$$

$$\delta\mathbf{w} = -\frac{w_q}{H_{qq}^{-1}} H^{-1} \cdot \mathbf{i}_q, \tag{4.15}$$

$\mathbf{i}_q$ jest wektorem składającym się z zer i jedynki na q-tej pozycji. Z kolei $\delta\mathbf{w}$ to poprawki dla wag jakie należy nanieść po usunięciu wagi w_q.

Proces uczenia sieci przebiega podobnie jak dla sieci z OBD (patrz powyżej). Hassibi prezentuje też wpływ metod opartych na analizie wielkości wag z metodami OBD i OBS na błąd modelu:

$$E(wagi) \geq E(OBD) \geq E(OBS). \tag{4.16}$$

4.1.3. Statystyczne i inne metody zmniejszania struktury sieci neuronowych

Oba algorytmy OBD i OBS mogą być stosowane do sieci MLP, jak i do sieci RBF. Statystyczne podejście do usuwania zbędnych połączeń zostało przedstawione przez Finnoffa i Zimmermann [83, 82] i Cottrella (i. in.) [48], a później użyte też w pracy Weigend'a (i. in.) w [254]. Idea tej metody opiera się na kumulowaniu różnic poszczególnych wag podczas uczenia w ramach jednej epoki. Następnie definiuje się współczynniki s_i, oceniające przydatność dla każdej wagi i. Współczynnik s_i jest równy ilorazowi wartości wagi powiększonej o średnie wahanie wartości wagi podzielonemu przez standardowe odchylenie średniej różnicy tej wagi:

$$s_i = \frac{|w_i + mean(\Delta w_i^j)|}{std(\Delta w_i^j)}, \tag{4.17}$$

w_i jest wartością wagi przed rozpoczęciem wyliczania zmian w kolejnej epoce, Δw_i^j jest równe zmianie wartości wagi w_i pod wpływem prezentacji j-tego wzorca uczącego. Poza tym mamy

$$mean(\Delta w_i^j) = \frac{1}{N}\sum_{j=1}^{N} \Delta w_i^j, \tag{4.18}$$

a $std(\Delta w_i^j)$ jest zdefiniowane przez:

$$std(\Delta w_i^j) = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(\Delta w_i^j - mean(\Delta w_i^j))^2}. \tag{4.19}$$

Współczynnik s_i, testujący wagę i określony równaniem (4.17), jest duży, gdy waga jest duża, a przeciętne różnice zmian tej wagi są małe. W odwrotnym przypadku mamy do czynienia z wagą, która jest raczej mało istotna lub zupełnie nieprzydatna.

W pracy [82] została opisana też procedura *epsi-prune*, której zadaniem nie jest pełne usunięcie zbędnej wagi, lecz jej dezaktywacja, która może być tylko czasowa. Zanim dojdzie do usuwania wag sieć zostaje poddana procedurze uczenia, aż do przeuczenia się (np. aż do zaobserwowania pogorszenia błędu klasyfikacji bądź aproksymacji na podzbiorze zbioru treningowego wyodrębnionym do testowania). Następnie, po kilku epokach, wyznacza się współczynniki s_i zgodnie z równaniem (4.17). Wtedy następuje dezaktywacja tych wag, dla których współczynniki s_i są mniejsze od ϵ ($\epsilon > 0$). Natomiast wagi, które już zostały dezaktywowane poprzednio, a teraz ich współczynniki s_i są większe od ϵ, podlegają aktywacji z pewną małą losową wartością początkową. Po za tym w każdej epoce wartość ϵ jest powiększana o pewną wartość $\Delta\epsilon$ ($\Delta\epsilon > 0$), aż do osiągnięcia pewnej wartości ϵ_{max}.

Inna, statystyczna metoda kontroli usuwania neuronów dla sieci typu RBF zostanie przedstawiona w podrozdziale 4.3.

Wartym wspomnienia jest również podejście Orr'a [197] do sieci RBF, bazujące na macierzy projekcji P, która wspomaga dokonywanie różnych operacji. Macierz projekcji jest zdefiniowana jako

$$P = I - GA^{-1}G^T, \tag{4.20}$$

gdzie G, to macierz wartości funkcji bazowych dla poszczególnych wektorów treningowych:

$$G = \begin{bmatrix} g_1(\mathbf{x}_1) & \dots & g_M(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ g_1(\mathbf{x}_n) & \dots & g_M(\mathbf{x}_n) \end{bmatrix}, \tag{4.21}$$

z kolei A jest równe

$$A = G^T G + \Lambda, \tag{4.22}$$

gdzie Λ jest macierzą przekątniową opisującą parametry regularyzacyjne poszczególnych wymiarów.

Macierz projekcji P pojawia się przy rozpisaniu błędu jaki popełni sieć RBF dla wektorów treningowych

$$\begin{aligned} \hat{y} - F &= \hat{y} - Gw && (4.23)\\ &= (I - GA^{-1}G^T)\hat{y} && (4.24)\\ &= P\hat{y}. && (4.25) \end{aligned}$$

Macierz projekcji pozwala na szybkie wyznaczenie sumy błędu kwadratowego:

$$E = (\hat{y} - F)^T(\hat{y} - F), \tag{4.26}$$
$$= \hat{y}^T P^2 \hat{y}. \tag{4.27}$$

Także i obliczanie funkcji kosztu może być szybkie:

$$C = (HG - \hat{y})^T(Gw - \hat{y}) + w^T \Lambda w \tag{4.28}$$
$$= \hat{y}^T P \hat{y}. \tag{4.29}$$

Usunięcie zbędnej funkcji bazowej g_j pociąga za sobą następującą operację na macierzy projekcji (zakłada się iż funkcja g_j została przesunięta do ostatniej kolumny macierzy P_m):

$$P_{m-1} = P_m + \frac{P_m g_j g_j^T P_m}{\lambda_j - g_j^T P_m g_j}. \tag{4.30}$$

Takie usunięcie kosztuje n operacji arytmetycznych (n to liczba wektorów uczących), natomiast pełne przetrenowanie sieci wymagałoby $M^3 + nM^2 + p^2 m$ operacji (M liczba funkcji bazowych).

W podobny sposób można dodać nową funkcję bazową. Zostało to opisane w podrozdziale 4.2.

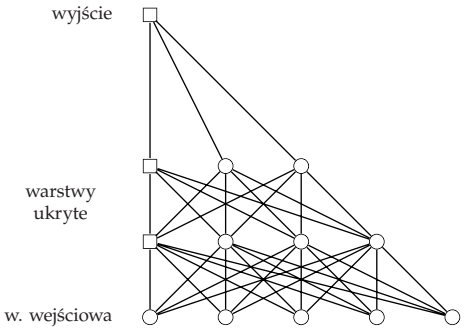
4.2. Modele o strukturach rozrastających się

4.2.1. Algorytm kafelkowania

Jednym z pierwszych modeli rozrastających się był algorytm kafelkowania (*ang. tiling algorithm*) do klasyfikacji wzorców binarnych (choć, jak zobaczymy poniżej, może być użyty nie tylko do wzorców binarnych) Mezarda i Nadala [183]. Sieć powstaje poprzez dokładanie kolejnych, nowych warstw o coraz mniejszej liczbie neuronów w kolejnych warstwach. Połączenia między neuronami są tylko pomiędzy sąsiednimi warstwami neuronów.

Każda z warstw musi spełniać warunek, który został nazwany warunkiem *wierności* zbiorowi treningowemu. Warunek ten wymaga, aby dwa wzorce uczące, należące do różnych klas nie były odwzorowane na taki sam wzorzec aktywacji danej warstwy. W przeciwnym razie dochodzi do sytuacji, w której dla wzorców z różnych klas powstaje wzorzec aktywacji odpowiadający jednej grupie aktywacji danej warstwy, co uniemożliwiłoby separacje takich wzorców na poziomie następnej warstwy neuronów, a w konsekwencji błąd.

Dopóki algorytm nie może zbudować *wiernej* warstwy, następuje dodawanie i uczenie kolejnych neuronów (dopełniających), tak długo, aż dana warstwa spełni warunek wierności (por. rysunek 4.2).



Rysunek 4.2: Architektura sieci kafelkowej. Kwadraty symbolizują pierwsze, główne neurony warstwy, kółka – neurony dopełniające (warstwy ukrytej), które stopniowo pomagają spełnić *warunek wierności* warstwy.

Uczenie odbywa się przy pomocy algorytmu kieszonkowego, do którego powrócimy za chwilę. Do uczenia wykorzystuje się wektory wejściowe, dla których uzyskano takie same aktywacje w budowanej warstwie, podczas gdy one należały do różnych klas (czyli wektory, które sprawiają, że nie jest spełniony warunek wierności).

Tak zdefiniowany algorytm gwarantuje zbieżność procesu uczenia. Wynika to z faktu, że używa się wzorców binarnych. Czyli tak naprawdę każdy wzorzec jest pewnym wierzchołkiem N wymiarowego hipersześcianu. Jak wiadomo wtedy bardzo łatwo odseparować go od reszty zbioru. Zbiory wzorców, w których każdy wzorzec daje się odseparować nazywa się wypukłymi. Innym zbiorem wypukłym wzorców jest zbiór, w którym dla każdego wzorca $\mathbf{x}$ mamy spełnione:

$$\sum_{i=1}^{d} x_i^2 = 1, \tag{4.31}$$

wtedy punkty leżą na N wymiarowej hipersferze.

4.2.1.0.1. Algorytm kieszonkowy Algorytm kieszonkowy stanowi odmianę algorytmu perceptronowego. Standardowa formuła zmian wag perceptronu:

$$\mathbf{w}^t = \mathbf{w}^{t-1} + \eta(y_p - f(\mathbf{x}_p))\mathbf{x}_p, \tag{4.32}$$

w przypadku linowej nieseparowalności może to prowadzić do fluktuacji. Przerwanie procesu uczenia oznacza w praktyce brak kontroli nad możliwie dobrym doborem wag. Dlatego powstało parę algorytmów, które mniej lub bardziej radzą sobie z tym problemem (inne to *termal perceptron* [85], *loss minimization algorithm* [125] czy *barycentric correction procedure* [210]).

Główna idea algorytmu kieszonkowego polega na zapamiętywaniu bieżących wag sieci, gdy ostatnia sekwencja poprawnie sklasyfikowanych wzorców była dłuższa niż ta, dla której ostatnio zapamiętano wagi sieci. Mówiąc dokładniej, najpierw wybierany jest losowo wektor uczący, gdy jest niepoprawnie klasyfikowany następuje adaptacja wag i wybór kolejnego wektora uczącego. Natomiast gdy wektor jest poprawnie klasyfikowany sprawdzana jest długość sekwencji ostatnio poprawnie klasyfikowanych wzorców i, jak już pisałem, gdy jest dłuższa niż ostatnia, następuje zapamiętanie bieżących wag sieci neuronowej.

Algorytm został zaproponowany w pracy [101]. Pokazano także, że dla skończonej liczby wzorców z prawdopodobieństwem $P < 1$ algorytm w skończonym czasie jest w stanie znaleźć optymalny zestaw wag z prawdopodobieństwem większym od P [102, 101, 100].

Wprowadzono także zmianę do algorytmu kieszonkowego. Zmiana polegała na dodatkowym warunku zapamiętania nowych wag sieci: liczba błędów musi ulec zmniejszeniu. To jednak nierzadko wydłużało proces uczenia.

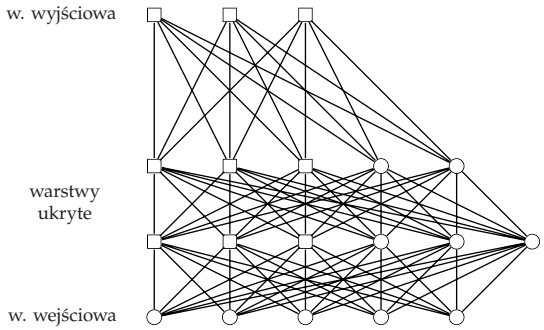
4.2.1.0.2. Sieć kafelkowa dla problemów wieloklasowych Bardzo łatwo rozbudować powyższą sieć tak, aby nadawała się dla problemów wieloklasowych. Wystarczy spojrzeć na rysunek 4.3 by zrozumieć ideę tego rozszerzenia. Każda warstwa ukryta, jak i wyjściowa, ma teraz K neuronów głównych, które są wspierane neuronami dopełniającymi. K jest równe liczbie klas problemu. Taki układ sprawia, że neurony główne z warstwy na warstwę coraz lepiej rozpoznają im odpowiadające klasy (i-ty neuron główny z warstwy na warstwę popełnia coraz mniej błędów w rozpoznawaniu swoich wzorców), dzięki specjalizacji neuronów dopełniających. Rozszerzenie tej metody jak i dowód zbieżności można znaleźć w [202]. W pracy [202] zaprezentowano wyniki, w których algorytm kafelkowania (dla wzorców o wartościach rzeczywistych) buduje mniejsze sieci (10%–250%) i dokładniejsze (0.3%–6%) w predykcji, niż algorytm piramidy opisany w następnym podrozdziale.

4.2.2. Algorytm wieża i piramida

Jeszcze prostszy, niż powyższy algorytm kafelkowania, jest algorytm wieża (ang. *tower*) [85]. Algorytm zakłada, że dane są wypukłe, a problem jest dwuklasowy. Tytułowa wieża jako podstawę ma warstwę z N wejściami. Kolejne warstwy ukryte składają się z pojedynczych neuronów. Każdy z nich połączony jest z każdym wejściem i neuronem bezpośrednio poniżej (por. rys. 4.4). Dokładanie kolejnych neuronów-warstw trwa, aż do osiągnięcia pełnej poprawności sieci.

Sieć uczona jest algorytmem kieszonkowym (4.2.1.0.1) ze wspomnianą poprawką. Jak można się domyślać, dzięki uczeniu zbioru wypukłego gwarantujemy, że każdy kolejny poziom zmniejsza liczbę błędów o nie mniej niż jeden.

Pewną generalizacją algorytmu wieża jest algorytm piramida [85]. Jest on chyba pierwszym przykładem architektury kaskadowej. W odróżnieniu od wie-



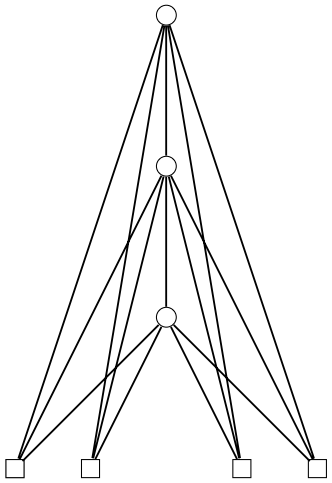
Rysunek 4.3: Architektura sieci kafelkowej dla problemów wieloklasowych. Kółka w warstwie ukrytej symbolizują neurony dopełniające, które stopniowo pomagają spełnić *warunek wierności* warstwy.

ży każdy neuron ukryty ma połączenia do wszystkich poniższych neuronów ukrytych.

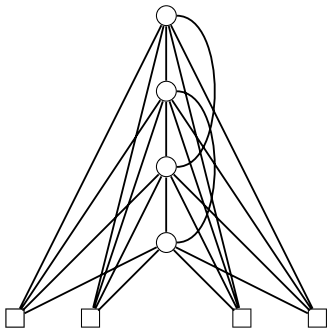
Oba algorytmy można, podobnie jak algorytm kafelkowy, uogólnić do problemów wieloklasowych. W tym celu w miejsce jednej kolumny (wieży, czy piramidy) umieszczamy K kolumn, po jednej dla każdej z klas. To uogólnienie podobnie jak uogólnienie algorytmu kafelkowego zostało zaprezentowane w [202]. Wadą jest, jak widać na rysunku 4.6, ogromny wzrost liczby połączeń.

4.2.3. Upstart

Algorytm (*upstart*) służy do klasyfikacji wzorców binarnych i został zaproponowany przez Freana [86]. Ten algorytm dokłada neurony pomiędzy już istniejącymi neuronami, a wejściami sieci. Algorytm upstart startuje ucząc jeden neuron algorytmem kieszonkowym. Następnie, gdy struktura sieci nie jest wystarczająca, aby uzyskać zakładany poziom dokładności, dokładane są dwa potomne neurony pomiędzy neuron, który *popełnia błędy* a jemu odpowiadające wejścia. Wagi neuronów ustala się na takie wartości, aby pierwszy neuron odpowiadał wzorcom niesklasyfikowanym przez neuron-ojca niepoprawnie, a drugi sklasyfikowanym ale błędnie. Po zamrożeniu wag dochodzących do neuronu-ojca następuje uczenie neuronów potomnych. Następnie i te wagi są zamrażane, i jeśli klasyfikacja nie jest satysfakcjonująca, dodawany jest kolejny neuron i proces jest powtarzany.



Rysunek 4.4: Sieć budowana za pomocą algorytmu wieża.



Rysunek 4.5: Sieć budowana za pomocą algorytmu piramida.

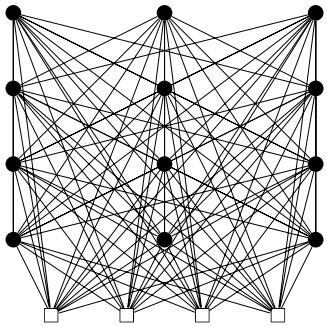
4.2.4. Algorytm budowania sieci kaskadowych przez analizę dychotomii

W kolejnym modelu zaproponowanym w [37] wstępnie zakłada się, że rozpatrujemy wzorce o binarnych wartościach, które mogą należeć do jednej z klas ($+1$ bądź -1).

Punktem wyjścia jest fakt, że istnieje neuron progowy, który *rozpoznaje* wszystkie wzorce klasy $+1$ i, ewentualnie, część reszty wzorców:

$$\exists_{\mathbf{w}} \forall_{\langle \mathbf{x}_i, y_i \rangle} \quad y_i = +1 \quad \Rightarrow \quad \Theta(\mathbf{w}, \mathbf{x}_i) = 1, \tag{4.33}$$

zakładam, że $\mathbf{w}$ w w_0 zawiera próg.



Rysunek 4.6: Sieć piramida dla problemów wieloklasowych.

Zdefiniujmy w takim razie neuron $N^{\oplus}$ jak poniżej

$$\forall_{\mathbf{x}_i \in \mathcal{S}_k} \quad N^{\oplus}_{\mathcal{S}_k}(\mathbf{x}_i) = \begin{cases} 1 & \Rightarrow \quad y_i = 1 \ \vee \ y_i = -1, \\ -1 & \Rightarrow \quad y_i = -1, \end{cases} \tag{4.34}$$

wtedy rozpoznaje on wszystkie swoje wektory (klasy $+1$) i część obcych (-1). W pierwszej iteracji ($k = 1$) $\mathcal{S}_k$ jest zbiorem wszystkich wzorców uczących. Oczywiście wyznaczenie takiego neuronu polega na ustaleniu wektora wag (do tego jeszcze powrócimy).

Teraz zdefiniujmy neuron $N^{\ominus}_{\mathcal{S}_k}$ jak poniżej

$$\forall_{\mathbf{x}_i \in \mathcal{S}_k} \quad N^{\ominus}_{\mathcal{S}_k}(\mathbf{x}_i) = \begin{cases} 1 & \Rightarrow \quad y_i = 1, \\ -1 & \Rightarrow \quad y_i = 1 \ \vee \ y_i = -1 \end{cases}. \tag{4.35}$$

Łatwo zauważyć, że gdy $N^{\oplus}_{\mathcal{S}_k}(\mathbf{x}_i) = 1$ i $N^{\ominus}_{\mathcal{S}_k}(\mathbf{x}_i) = 1$, to z pewnością $\mathbf{x}_i$ należy do klasy $+1$. Podobnie gdy $N^{\oplus}_{\mathcal{S}_k}(\mathbf{x}_i) = -1$ i $N^{\ominus}_{\mathcal{S}_k}(\mathbf{x}_i) = -1$, to $\mathbf{x}_i$ należy do klasy -1. Problem pozostaje nierozstrzygnięty, gdy dla danego $\mathbf{x}_i$ mamy $N^{\oplus}_{\mathcal{S}_k}(\mathbf{x}_i) = -N^{\ominus}_{\mathcal{S}_k}(\mathbf{x}_i)$. Czyli dokładniej — jedna para takich neuronów nie wystarczy…

Umieśćmy tak zdefiniowane dwa neurony $N^{\oplus}_{\mathcal{S}_k}$ i $N^{\ominus}_{\mathcal{S}_k}$ w pierwszej warstwie ukrytej. Każdy neuron pierwszej warstwy ukrytej jest połączony z każdym wejściem. Następnie, w kolejnej warstwie, umieszczamy neuron liniowy(!), który łączy dwa neurony dodane ostatnio i ustalamy wagi połączeń na $+1$. Wyjście takiego neuronu przyjmuje wartość sumy $N^{\oplus}_{\mathcal{S}_k}(\mathbf{x}_i)$ i $N^{\ominus}_{\mathcal{S}_k}(\mathbf{x}_i)$, mamy wtedy:

$$N^{\oplus}_{\mathcal{S}_k}(\mathbf{x}_i) + N^{\ominus}_{\mathcal{S}_k}(\mathbf{x}_i) = \begin{cases} 2 & \Rightarrow \quad y_i = 1, \\ 0 & \Rightarrow \quad y_i = 1 \ \vee \ y_i = -1, \\ -2 & \Rightarrow \quad y_i = -1. \end{cases} \tag{4.36}$$

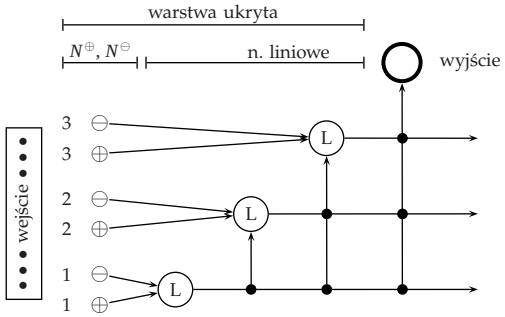
Jak widać obecnie sieć działa źle dla niektórych wektorów. Następny krok to zamrożenie obecnych wag i dodanie dwóch kolejnych neuronów do pierwszej warstwy ukrytej $N^{\oplus}_{\mathcal{S}_{k+1}}$ i $N^{\ominus}_{\mathcal{S}_{k+1}}$. Jednak wagi tych neuronów będą wyznaczane ze względu na zbiór $\mathcal{S}_{k+1}$, w którym znajdą się wszystkie wektory, dla których ostatnia wersja sieci z k-tej iteracji popełniała błędy. Ten etap całej metody budowania sieci został nazwany algorytmem zmiany celu (*ang. target switching algorytm*) [37].

Nowo dodane neurony stają się nowymi wejściami kolejnego liniowego neuronu (kaskadowego), w kolejnej warstwie ukrytej. Dodatkowo także wyjście poprzednio dodanego neuronu liniowego jest wejściem nowego neuronu kaskadowego (patrz rysunek 4.7).

Taki proces powtarzamy, aż osiągniemy pełne rozwiązanie problemu.

Zauważmy, że gdy pierwsze dwa neurony $N^{\oplus}_{\mathcal{S}_1}$ i $N^{\ominus}_{\mathcal{S}_1}$ dobrze rozpoznają dany wzorzec $\mathbf{x}$, to neuron liniowy, który je łączy ma wartość $+2$ lub -2 (porównaj 4.36). Tak silne wyjście hamuje wpływ decyzji pozostałych neuronów $N^{\oplus}_{\mathcal{S}_k}$ i $N^{\ominus}_{\mathcal{S}_k}$ i finalnie zostaje zwrócona poprawna odpowiedź.

Zauważmy z kolei, że gdy neurony $N^{\oplus}_{\mathcal{S}_1}$ i $N^{\ominus}_{\mathcal{S}_1}$ nie rozpoznają pewnego wzorca $\mathbf{x}$, to neuron liniowy łączący te dwa neurony ma na wyjściu wartość zero, która nie wpływa na działanie pozostałych neuronów. Przez rekurencję do głosu zostają dopuszczane kolejne neurony $N^{\oplus}_{\mathcal{S}_2}$ i $N^{\ominus}_{\mathcal{S}_2}$. Jeśli te dwa neurony rozpoznają wzorzec $\mathbf{x}$, to tak, jak poprzednio zdominują przepływ informacji przez sieć. Jeśli nie rozpoznają, to procedura powtarza się (por. rys 4.7).



Rysunek 4.7: Kaskadowa struktura sieci dychotomicznej.

Wyznaczanie wektora wag dla neuronów dychotomicznych $N^{\oplus}_{\mathcal{S}_k}$ i $N^{\ominus}_{\mathcal{S}_k}$ może w pierwszym etapie korzystać z algorytmu kieszonkowego, a gdy nie udało się w pełni rozseparować wektorów obcych klas, modyfikujemy progi tak, aby neurony $N^{\oplus}_{\mathcal{S}_k}$ i $N^{\ominus}_{\mathcal{S}_k}$ spełniały warunki 4.34 i 4.35.

Wykorzystując sposób konstruowania neuronów $N^{\oplus}$ i $N^{\ominus}$ Campbell i Perez zaproponowali nieco inne jeszcze rozwiązania [38, 36].

4.2.5. Algorytm korelacji kaskadowej

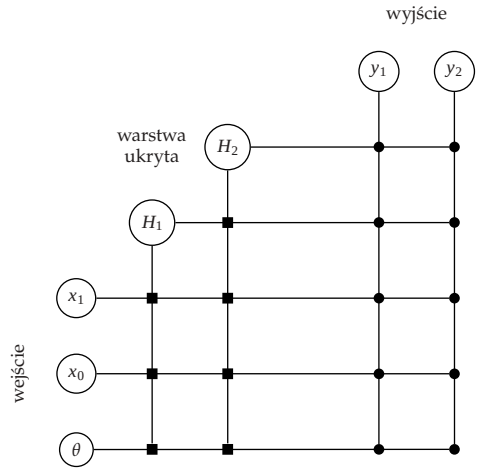
Cascade-correlation network (CC) Fahlmana i Lebiere'a [78, 79] jest jednym z najsprawniejszych algorytmów uczenia sieci MLP, który umożliwia rozrastanie się struktury aż do uzyskania sieci o wystarczającej pojemności. Fahlman i Lebiere używali do adaptacji wag algorytmu szybkiej wstecznej propagacji, choć można rozważać użycie i innych algorytmów adaptacji.

Sieć kaskadowej korelacji rozpoczyna z jedną warstwą wag pomiędzy wejściem i wyjściem sieci (por. rys. 4.8). Następnie sieć jest uczona przez pewien czas, po czym następuje dołożenie nowego neuronu do warstwy ukrytej. Proces ten jest powtarzany aż do uzyskania zakładanego poziomu błędu. Na wejścia kolejno dodawanych neuronów składają się wszystkie wejścia sieci i wyjścia poprzednio dodanych neuronów, skąd bierze się słowo kaskada w nazwie sieci.

Drugi człon nazwy sieci to maksymalizacja korelacji poniższego wyrażenia (4.37), którego celem jest ustalenie możliwie najlepszych wag dla nowego neuronu

$$S = \sum_{i=1}^{p} \left| \sum_{j=1}^{n} (o_j - \bar{o})(e_j^i - \bar{e}^i) \right|, \tag{4.37}$$

gdzie n oznacza liczbę wzorców uczących, p jest liczbą wyjść sieci, o_j jest wartością aktywacji neuronu, który ma być dodany dla j-tego wzorca uczącego, $\bar{o}$ jest średnią aktywacją dodawanego neuronu, e_j^i jest błędem i-tego wyjścia dla j-tego wzorca uczącego, a $\bar{e}^i$ jest średnim błędem i-tego wyjścia.



Rysunek 4.8: Architektura sieci kaskadowej korelacji. Kwadraty symbolizują zamrożone wartości wag z neuronami ukrytymi. Pozostałe wagi ulegają ciągłej adaptacji.

Łatwo można wyznaczyć pochodną równania (4.37)

$$\frac{\partial S}{\partial w_k} = z_i \sum_{i=1}^{p} \sum_{j=1}^{n} (e_j^i - \bar{e}^i) g_j' I_j^k, \tag{4.38}$$

z_i jest równe 1 lub -1 zależnie od tego, czy korelacja pomiędzy neuronem i i-tym wyjściem sieci jest dodatnia lub ujemna; g_j' jest wartością pochodnej funkcji transferu dodawanego neuronu dla j-tego wektora uczącego, z kolei I_j^k jest wartością k-tego wejścia dodawanego neuronu dla j-tego wektora uczącego.

Start uczenia neuronu, który ma zostać dodany, rozpoczyna się od wartości losowych, dlatego też często wybiera się kilka neuronów-kandydatów, z których następnie wybiera się lepszego po wstępnym procesie adaptacji, w którym maksymalizuje się korelacje.

Rozbudowując sieć do kilku neuronów wyjściowych można rozważać także problemy wieloklasowe.

Falhman opracował również rekurencyjną wersję wyżej opisanego algorytmu [77].

4.2.6. Kaskadowa sieć perceptronowa

Burgess w [31] zaproponował kaskadową sieć, w której użył neuronów progowych. Jeśli chodzi o architekturę i rozrost sieci, to są one takie same, jak w przypadku sieci korelacji kaskadowej z poprzedniego podrozdziału (4.2.5). Nowe neurony dokładane są tak długo, aż zostanie uzyskany wymagany stopień dokładności. Taka sieć może być uczona przez algorytm kieszonkowy (4.2.1.0.1).

4.2.7. Feature Space Mapping (FSM)

Innym, ciekawym modelem, umożliwiającym rozbudowywanie struktury podczas uczenia się przez dodawanie nowych neuronów, jest wspomniany już system FSM, rozwijany w naszym zespole [1, 64, 71]. Model ten używany głównie do klasyfikacji i wyciągania reguł logicznych z danych.

Architektura FSM jest praktycznie taka sama, jak sieci RBF. Składa się z warstwy wejściowej, wyjściowej i jednej warstwy ukrytej. Jako, iż typowym zastosowaniem systemu jest klasyfikacja, na wyjściu pojawia się informacja o tym, do której klasy został przypisany wektor pokazany warstwie wejściowej. Jednakże najważniejszą część stanowi warstwa ukryta, która jest odpowiedzialna za konstrukcje odpowiednich zbiorów odwzorowań. Mamy więc nie tylko podobieństwo architektur sieci RBF, ale i roli ich warstw ukrytych. W zastosowaniach klasyfikacyjnych, jak i w ekstrakcji reguł logicznych, neurony warstwy ukrytej odpowiadają pewnej (zależnej od funkcji transferu) klasteryzacji przestrzeni wejściowej. Typowymi funkcjami transferu neuronów warstwy ukrytej są funkcje gaussowskie wielu zmiennych 1.84, funkcje trójkątne, funkcje bicentralne 1.101, jak i funkcje prostokątne.

Inicjalizacja architektury jest dokonywana za pomocą algorytmów klasteryzacji: poprzez histogramy, drzewa decyzyjne lub dendrogramy. Algorytmy te zostały opisane w podrozdziałach 2.3.6 i 2.3.5, i [62].

Po procesie inicjalizacji następuje etap estymacji położeń centrów neuronów i ich rozmyć, jak i innych parametrów algorytmem optymalizacji dyskretnej opisanym w [1, 64]. Główna część procesu estymacji oparta jest o analizę poszczególnych wektorów uczących neuronu, dla którego uzyskano największą aktywację w wyniku prezentacji danego wektora uczącego i neuronu najbliższego neuronowi, dla którego uzyskano największą aktywację.

Algorytm adaptacji umożliwia dodanie nowego neuronu, gdy są spełnione trzy warunki:

1. dla danego wektora uczącego $\mathbf{x}$, neuron, dla którego została uzyskana największa aktywacja N_M i neuron jemu najbliższy N_N, należą do różnych klas,

2. odległość, pomiędzy wektorem uczącym, a neuronem maksymalnie pobudzonym N_M, spełnia poniższą nierówność

$$||\mathbf{x} - \mathbf{t}_{N_M}|| > \sigma_{N_M}\sqrt{\ln 10}. \tag{4.39}$$

3. maksymalna aktywacja, uzyskana dla neuronu N_M, jest mniejsza niż ustalona wartość Act_{min}

$$G_{N_M}(\mathbf{x}) < Act_{min}. \tag{4.40}$$

Wspomniana już sieć MLP2LR [60], służąca do wyciągania reguł logicznych, również umożliwia dodawanie nowych neuronów, a samo kryterium niewystarczalności struktury sieci jest zdefiniowane w bardzo prosty sposób: sieć przestała się uczyć (wartość funkcji błędu przestała maleć) i poprawność klasyfikacji jest wciąż zbyt mała.

W podrozdziale 4.1.3 opisano między innymi metodę usuwania funkcji bazowych sieci RBF, opisaną przez Orr'a [197] za pomocą operacji na macierzy projekcji P. Wykonując inne, choć podobne, operacje na macierzy projekcji, można dodać nowy neuron. Odpowiednie zmiany w macierzy projekcji P_m pokazane są poniżej.

$$P_{m+1} = P_m - \frac{P_m h_{m+1} h_{m+1}^T P_m}{\lambda_j + h_{m+1}^T P_m h_{m+1}}. \tag{4.41}$$

W pracach [94, 95] Fritzke prezentuje sieć RBF, w której co λ kroków uczenia (prezentacji pojedynczego wektora uczącego) następuje dodanie nowego neuronu. Jednakże zanim nastąpi dodanie nowego neuronu, podczas procesu adaptacyjnego, wyznaczane są skumulowane błędy, jakie sieć popełnia w poszczególnych podobszarach, które reprezentują neurony warstwy ukrytej. Przy każdej

prezentacji kolejnego wektora uczącego następuje kumulowanie błędu, doliczając sumaryczny błąd kwadratowy neuronowi s, który był najbliżej prezentowanego wektora uczącego:

$$\Delta err_s = \sum_{i=1}^{d} (F(\mathbf{x}^i) - y^i)^2, \tag{4.42}$$

$F(\mathbf{x})^i$ oznacza wartość i-tego wyjścia dla wektora $\mathbf{x}$, a y^i wartość oczekiwaną na i-tym wyjściu. Po upływie każdych λ kroków uczenia następuje dodanie nowego neuronu w pobliżu neuronu, dla którego aktualny skumulowany błąd jest największy. Dokładniej pozycja nowego neuronu jest wyznaczona jako średnia pozycji neuronu, dla którego skumulowany błąd był największy i jednego z jego najbliższych sąsiadów.

Informacje o jeszcze innych modelach sieci samoorganizujących się można znaleźć w [97, 96]. Tu na szczególną uwagę zasługuje pierwsza praca, w której rozkład topologii sieci może podążać za zmieniającym się w czasie rozkładem danych.

Fiesler w [81] dokonał porównania ponad 30 modeli ontogenicznych. Zestawienie zawiera informacje o liczbie warstw, czy sieć umożliwia rozrastanie się, czy sieć może się kurczyć, czy metoda jest lokalna, jakie są dodatkowe warunki nakładane na sieć (np. czy startuje z pustej struktury), czy istnieją jakieś matematyczne dowody na zbieżność metody. Jednakże zestawienie nie wskazuje na jakiekolwiek wady, czy zalety poszczególnych modeli.

Poniżej zostanie pokazany inny sposób rozrastania się sieci typu RBF, wraz z kryterium wystarczalności sieci neuronowej RAN. Natomiast w podrozdziale 4.3 pokazana zostanie sieć IncNet, korzystająca z jeszcze sprawniejszego kryterium wystarczalności modelu, wykorzystującego informację czysto statystyczną, wyznaczaną na podstawie stanu filtra Kalmana.

4.2.8. Sieć RAN z przydziałem zasobów

Sieć z przydziałem zasobów, której oryginalna angielska nazwa to *Resource Allocation Network* (RAN), została zaproponowana przez Platt'a w [204]. Nieco później Kadirkamanathan i Niranjan [149, 145] pokazali, iż, przy pewnych założeniach, użyty w sieci RAN sekwencyjny sposób uczenia jest poprawny matematycznie. Podobnie zostało pokazane, że kryteria rozrostu sieci (nazwane geometrycznym kryterium rozrostu) są również poprawne przy założeniach, które nie stoją w opozycji do opisanego modelu przez Platta.

4.2.8.1. Uczenie sekwencyjne.

Sieć RAN można rozpatrywać, jako sekwencyjną metodę estymacji pewnego nieznanego gładkiego odwzorowania F^*, co oznacza, że celem jest wyznaczenie

nowego *stanu modelu* [5]podstawie poprzedniego stanu modelu $F^{(n-1)}$ i nowej obserwacji $I^{(n)}$, która jest parą uczącą $\langle \mathbf{x}_n, y_n \rangle$ ($\mathbf{x} \in \mathcal{R}^N, y \in \mathcal{R}$).

Tak określone zadanie można zapisać w postaci funkcji celu $\mathcal{F}$-projekcji:

$$F^{(n)} = \arg\min_{f \in \mathcal{H}} ||f - F^{(n-1)}|| \qquad F^{(n)} \in \mathcal{H}_n, \tag{4.43}$$

gdzie $\mathcal{H}$ jest przestrzenią Hilberta funkcji o skończonej normie:

$$\mathcal{H} = \{f : ||f|| < \infty\}, \tag{4.44}$$

$||\cdot||$ jest normą L^2 zdefiniowaną przez

$$||f|| = \int_{\mathbf{x} \in \mathcal{D}} |f(\mathbf{x})|^2 d\mathbf{x} \qquad \mathcal{D} \subseteq \mathcal{R}^N, \tag{4.45}$$

a $\mathcal{H}_n$ jest zbiorem funkcji określonych poprzez:

$$\mathcal{H}_n = \{f : f \in \mathcal{H} \ \wedge \ f(\mathbf{x}_n) = y_n\}. \tag{4.46}$$

Warunek $F(\mathbf{x}_n) = y_n$ można zapisać jako iloczyn skalarny funkcji g_n należących do przestrzeni $\mathcal{H}$:

$$\langle F, g_n \rangle = \int_{\mathbf{x} \in \mathcal{D}} F(\mathbf{x}) g(\mathbf{x} - \mathbf{x}_n) = y_n, \tag{4.47}$$

gdzie $g(\cdot)$ jest funkcją impulsową (przybliżeniem do delty Diraca). To pozwala na zapisanie funkcji celu jako $\mathcal{F}$-projekcji w postaci modelu *a posteriori*:

$$F^{(n)} = F^{(n-1)} + e_n \frac{g_n}{||g_n||^2} = F^{(n-1)} + e_n h_n, \tag{4.48}$$

e_n jest błędem popełnionym przez a priori model $F^{(n-1)}$ w punkcie $\mathbf{x}_n$:

$$e_n = y_n - F^{(n-1)}(\mathbf{x}_n). \tag{4.49}$$

Takie rozwiązanie wprowadza bardzo ostrą funkcję impulsową w punkcie $\mathbf{x}_n$ do już istniejącego modelu $F^{(n-1)}$ tak, aby model w kolejnym stanie osiągał wartość y_n w punkcie $\mathbf{x}_n$. Kłóci się to z założeniem, iż szukane (estymowane) odwzorowanie jest funkcją gładką. To prowadzi do ograniczenia na gładkość funkcji $h_n(\cdot)$, która może być użyta w naszym rozwiązaniu (4.48)

$$h_n(\mathbf{x}_n) = 1 \qquad \wedge \qquad h_n(\mathbf{x}_n + a) = 0 \quad \text{dla } ||a|| > 0. \tag{4.50}$$

Dobrym przykładem takiej funkcji $h_n(\cdot)$ może być funkcja gaussowska z odpowiednio dobranym współczynnikiem rozmycia b

$$G(\mathbf{x}; \mathbf{t}, b) = e^{-||\mathbf{x}-\mathbf{t}||^2/b^2}, \tag{4.51}$$

[5]Przez stan modelu rozumie się wszystkie elementy, które określają jednoznacznie całą sieć neuronową. Czyli liczba neuronów, typ funkcji, parametry wszystkich neuronów i połączeń.

mamy wtedy:

$$G(\mathbf{x};\mathbf{x}_n,b)=1 \qquad \wedge \qquad G(\mathbf{x}+a;\mathbf{x}_n,b)\to 0 \quad \text{dla } ||a||\to\infty. \tag{4.52}$$

Podsumowując, korzystając z $\mathcal{F}$-projekcji i powyższego założenia o gładkości, możemy nasz model w stanie n przedstawić jako:

$$F^{(n)}=F^{(n-1)}+e_nG(\mathbf{x};\mathbf{x}_n,b_0), \tag{4.53}$$

b_0 jest pewnym rozmyciem początkowym.

Przyjmując, że poprzedni model $F^{(n-1)}$ składał się z M funkcji bazowych (neuronów) mamy:

$$F^{(n)} = \sum_{i=1}^{M} w_iG(\mathbf{x};\mathbf{t}_i,b_i)+e_nG(\mathbf{x};\mathbf{x}_n,b_0), \tag{4.54}$$

$$= \sum_{i=1}^{M+1} w_iG(\mathbf{x};\mathbf{t}_i,b_i), \tag{4.55}$$

z $\mathbf{t}_{m+1}=\mathbf{x}_n$ i $b_{M+1}=b_0$. Oznacza to rozbudowującą się sieć RBF (patrz rys. 4.9).

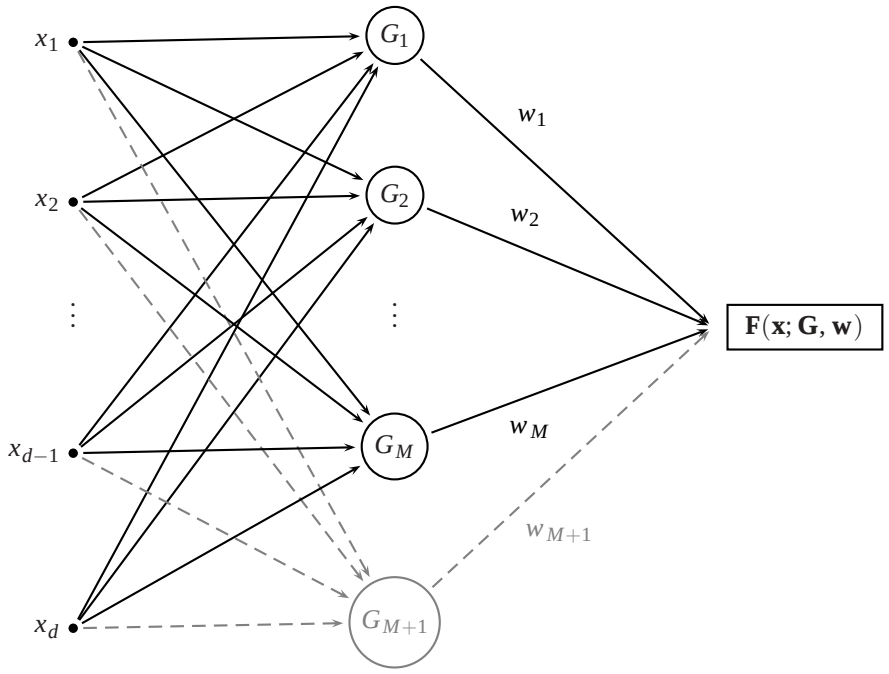
Z powyższych rozważań widać, iż sekwencyjny ciąg kolejnych modeli, wybierając odpowiednio małą wartość początkową b_0, może w rezultacie prowadzić do dowolnie małego ustalonego błędu interpolacji ϵ.

Oczywiście każdy kolejny stan k modelu $F^{(n+k)}$ nie powinien kosztować dodania nowej funkcji bazowej. Efekty takiej strategii były już opisywane w rozdziale 2, a szczególnie w podrozdziale 2.1 i 2.2. Dlatego też poniżej zostanie opisane geometryczne kryterium rozrostu sieci RAN, umożliwiające regulowanie złożoność sieci, w zależności od aktualnego jej stanu i napływających nowych obserwacji I_n.

4.2.8.2. Geometryczne Kryterium Rozrostu

Geometryczne Kryterium Rozrostu umożliwia kontrolę złożoności modelu podczas sekwencyjnego uczenia. Kryterium musi odpowiedzieć na pytanie: czy model $F^{(n)}$ wystarczy, aby dostatecznie dobrze estymować nową obserwację I_n, czy też jest niewystarczający i wymaga dodania nowej funkcji bazowej.

Aby odpowiedzieć na powyższe pytanie trzeba prześledzić sytuację, w której model nie zostaje powiększony o nowy neuron i sytuację, w której model zostaje rozbudowany o nową funkcję bazową (patrz rysunek 4.10). Modele *a priori* $F^{(n-1)}$ i *a posteriori* $F_*^{(n)}$ (nie powiększony o nową funkcję bazową) należą do przestrzeni $\mathcal{H}_M$. Natomiast model *a posteriori* $F^{(n)}$ powiększony o nową funkcję bazową należy już do przestrzeni $\mathcal{H}_{M+1}$. Model $F_*^{(n)}$ jest projekcją modelu $F^{(n)}$ do przestrzeni $\mathcal{H}_M$.



Rysunek 4.9: Sieć RAN z nową funkcją bazową G_{M+1}.

Tym samym odpowiedź na powyższe pytanie zawarta jest w wielkości odległości $||F^{(n)} - F_*^{(n)}||$. Jeśli jest ona większa od pewnego ϵ oznacza to, iż przestrzeń funkcji $\mathcal{H}_M$ jest niewystarczająca do uzyskania akceptowalnego poziomu estymacji:

$$||F^{(n)} - F_*^{(n)}|| > \epsilon. \tag{4.56}$$

Poza tym mamy też następującą własność (porównaj z rys. 4.10):

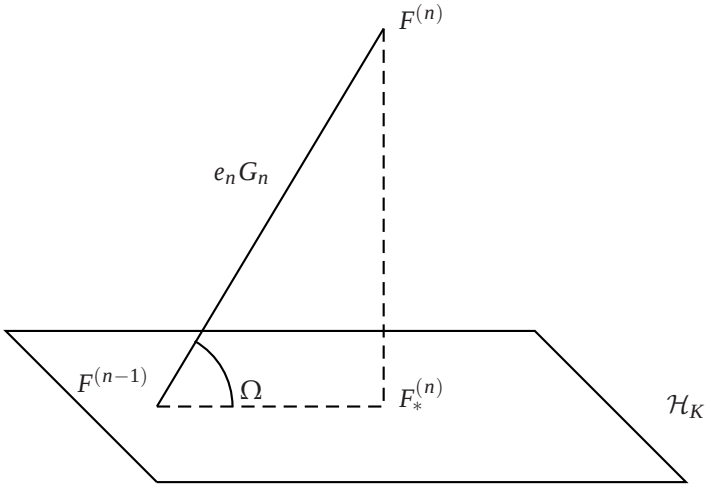
$$||F^{(n)} - F_*^{(n)}|| = |e_n| \cdot ||G_n|| \sin(\Omega). \tag{4.57}$$

Ponieważ $||G_n||$ zależy tylko od stałej początkowego rozmycia b_0, a z kolei kąt Ω może przyjmować wartości od 0 do $\pi/2$, można uprościć nierówność (4.56) do poniższych kryteriów geometrycznych:

$$e_n > e_{min}, \tag{4.58}$$

$$\Omega > \Omega_{min}. \tag{4.59}$$

Pierwsza z powyższych nierówność to *kryterium predykcji błędu*. Ocenia ono błąd interpolacyjny. Druga z powyższych nierówności to *kryterium kąta*, które ocenia na ile funkcja bazowa G_n jest ortogonalna (duży kąt) do funkcji bazowych z przestrzeni funkcji $\mathcal{H}_M$.



Rysunek 4.10: Zależności pomiędzy modelami a posteriori $F_*^{(n)}$ i $F^{(n)}$ (odpowiednio z przestrzeni $\mathcal{H}_M$ i $\mathcal{H}_{M+1}$) względem a priori modelu $F^{(n-1)}$.

W ogólnym przypadku wyznaczenie poszczególnych kątów jest trudne, ale można dokonać pewnych przybliżeń, na przykład przyjmując równe rozmycia funkcji G_n i funkcji G_i. Wtedy kąt pomiędzy funkcjami G_n i G_i jest równy:

$$\Omega_i = cos^{-1}\left(\exp\left(-\frac{1}{2b_0^2}||\mathbf{x}_n - \mathbf{t}_i||^2\right)\right). \tag{4.60}$$

Korzystając z powyższej własności można przepisać kryterium kąta (4.59) do postaci:

$$\sup_i G_i(\mathbf{x}_n) \leq \cos^2(\Omega_{min}), \tag{4.61}$$

co ze względu na monotoniczne nachodzenie się funkcji gaussowskiej przy o oddalaniu się punktu $\mathbf{x}_n$ od centrum funkcji G_i, można zastąpić przez:

$$\inf_i ||\mathbf{x}_n - \mathbf{t}_i|| \geq \epsilon. \tag{4.62}$$

Powyższe kryterium przeradza się w *kryterium odległości* i tak naprawdę pokazuje, że porównaniu podlega odległość pomiędzy punktem $\mathbf{x}_n$ i centrum najbliższej funkcji bazowej, a wartości ϵ

$$\epsilon = b_0\sqrt{2\log(1/\cos^2\Omega_{min})}. \tag{4.63}$$

Można też określić optymalne rozmycie dla nowej funkcji bazowej, które będzie maksymalnie duże, ale będzie też spełniać kryterium kąta (4.59):

$$b_n = \frac{||\mathbf{x}_n - \mathbf{t}_k||}{\sqrt{2\log(1/\cos^2\Omega_{min})}} \qquad k = \arg\min_k ||\mathbf{x}_n - \mathbf{t}_k||. \tag{4.64}$$

Podsumowując, stwierdzić można, że nowy neuron jest dodawany, gdy spełnione jest kryterium predykcji błędu (4.58) i kryterium odległości (4.62).

4.2.8.3. Adaptacja sieci RAN

Gdy nie jest spełnione kryterium predykcji błędu (4.58) lub kryterium odległości (4.62) następuje adaptacja wolnych parametrów sieci

$$F(\mathbf{x}) = \sum_{i=1}^{M} w_i G(\mathbf{x}; \mathbf{t}_i, b_i) + w_0. \tag{4.65}$$

Wagi i położenia centrów funkcji bazowych

$$\mathbf{p}^{(n)} = [w_0, w_1, \ldots, w_M, \mathbf{t}_1^T, \mathbf{t}_2^T, \ldots, \mathbf{t}_M^T]^T \tag{4.66}$$

są uczone algorytmem LMS:

$$\mathbf{p}^{(n)} = \mathbf{p}^{(n-1)} + \eta e_n \mathbf{d}_n, \tag{4.67}$$

gdzie $\mathbf{d}_n$ jest gradientem funkcji $F(\mathbf{x}_n)$ po parametrach $\mathbf{p}$:

$$\begin{aligned}\mathbf{d}_n = \frac{\partial f(\mathbf{x}_n)}{\partial \mathbf{p}_{n-1}} = \Bigg[1, G_1(\mathbf{x}_n), \ldots, G_M(\mathbf{x}_n), G_1(\mathbf{x}_n)\frac{2w_1}{b_1^2}(\mathbf{x}_n - \mathbf{t}_1)^T, \ldots, \\ G_M(\mathbf{x}_n)\frac{2w_M}{b_M^2}(\mathbf{x}_n - \mathbf{t}_M)^T\Bigg].\end{aligned} \tag{4.68}$$

Zastępując η przez $\frac{\eta}{||\mathbf{x}_n||^2}$, uzyskuje się znormalizowaną wersję algorytmu LMS.

W praktyce również odległość minimalna ϵ podlega zmianom podczas procesu uczenia. Początkowa wartość ϵ jest równa ϵ_{max}, a następnie podlega iteracyjnym zmianom

$$\epsilon = \epsilon_{max}\ \gamma^n \tag{4.69}$$

$(0 < \gamma < 1)$, aż do osiągnięcia wartości ϵ_{min}.

4.3. Sieć IncNet ze statystyczną kontrolą złożoności sieci

Praktycznie wszystkie przedstawione powyżej modele ontogeniczne zawsze nakładają dość drastyczne uproszczenia dla całości procesu uczenia. Wynikają one

z koncentracji uwagi nad częścią rozwiązywanego problemu, a nie na całości. Dla przykładu algorytmy OBD [51, 50], OBS [115, 116] czy FSM [1, 64, 71, 65], zezwalają na usuwanie neuronów praktycznie po zakończeniu właściwego procesu uczenia. Z kolei inne modele dodające człon regularyzacyjny, wymuszający niskie wartości wag, co często prowadzi do zbyt małych wartości niemal wszystkich wag, a z kolei inne metody regularyzacji wymagają dobrego dobrania parametru (lub parametrów) na podstawie wiedzy a priori, co nie zawsze jest proste w praktyce. Rozważania w poniższych podrozdziałach pokażą, iż można zdefiniować inne kryteria, które umożliwią usuwanie neuronów praktycznie z iteracji na iterację, czyli gdy tylko okaże się to korzystne dla procesu adaptacji, a nie co epokę lub po zakończeniu procesu uczenia tak jak jest to realizowane w innych algorytmach.

Algorytmy, które mogą modyfikować swoją architekturę podczas uczenia, nie czynią tego w zbyt optymalny sposób. Podobnie zresztą jak i modele, które są zdolne do eliminacji neuronów lub wag, nierzadko czeka się, aż sieć neuronowa kompletnie zakończy uczenie i dopiero wtedy próbuje się dokładać nowe neurony lub wagi. Czy też, tak, jak jest to w sieci RAN (por. podrozdział 4.2.8), niekoniecznie spełnienie kryteriów (kryterium predykcji błędu i kryterium odległości) opisanych równaniami (4.58 i 4.62), odpowiada sytuacjom, w których sieć nie jest zdolna do uwzględnienia nowej obserwacji. Może to odpowiadać zbyt szybkiej prezentacji danej w dość odległym rejonie, do którego i tak jakaś z funkcji bazowych zostałaby *przyciągnięta* w miarę postępowania procesu uczenia. Spotyka się również naiwną taktykę dokładania nowych funkcji bazowych do sieci RBF co stałą liczbę kroków uczenia. Dlatego też w jednym z poniższych podrozdziałach zaprezentowane zostanie alternatywne i znacznie efektywniejsze kryterium.

Jeszcze inną wadą już nie tylko powyżej opisanych systemów ontogenicznych lecz znacznej części sieci neuronowych jest lekceważenie istotności wyboru funkcji transferu, które silnie determinują możliwości generalizacji modeli adaptacyjnych.

Głównym celem systemu IncNet, zaprezentowanego w kolejnych podrozdziałach, jest stworzenie takiego modelu, który będzie korzystał z efektywnego algorytmu uczenia i jednocześnie będzie zdolny do auto-kontroli złożoności architektury sieci podczas procesu uczenia, tak aby złożoność architektury sieci odzwierciedlała złożoność zaprezentowanej dotychczas części zbioru wektorów uczących.

Algorytm uczenia + Kontrola złożoności architektury sieci + Funkcje transferu	=	Możliwości modelu

Proponowanym rozwiązaniem problemu może być system, który będzie łączył w sobie poniższe cechy:

Uczenie: wykorzystanie efektywnego filtra Kalmana do adaptacji swobodnych parametrów sieci,

Kontrola złożoności sieci: wykorzystanie kryteriów kontrolujących rozbudowywanie się sieci, jak i kurczenie się sieci, poprzez dodawanie, usuwanie i łączenie się neuronów,

Elastyczne funkcje transferu: użycie bicentralnych funkcji transferu umożliwiają znacznie efektywniejszą estymacje, a w efekcie umożliwia adaptacje bardziej złożonych problemów.

System, składający się z takich funkcji, jest zdolny do efektywnego uczenia sieci neuronowej przy jednoczesnej kontroli użyteczności każdego z podelementów (funkcji bazowych formujących przestrzeń modelowania) i wystarczalności całości systemu do estymacji nowych obserwacji.

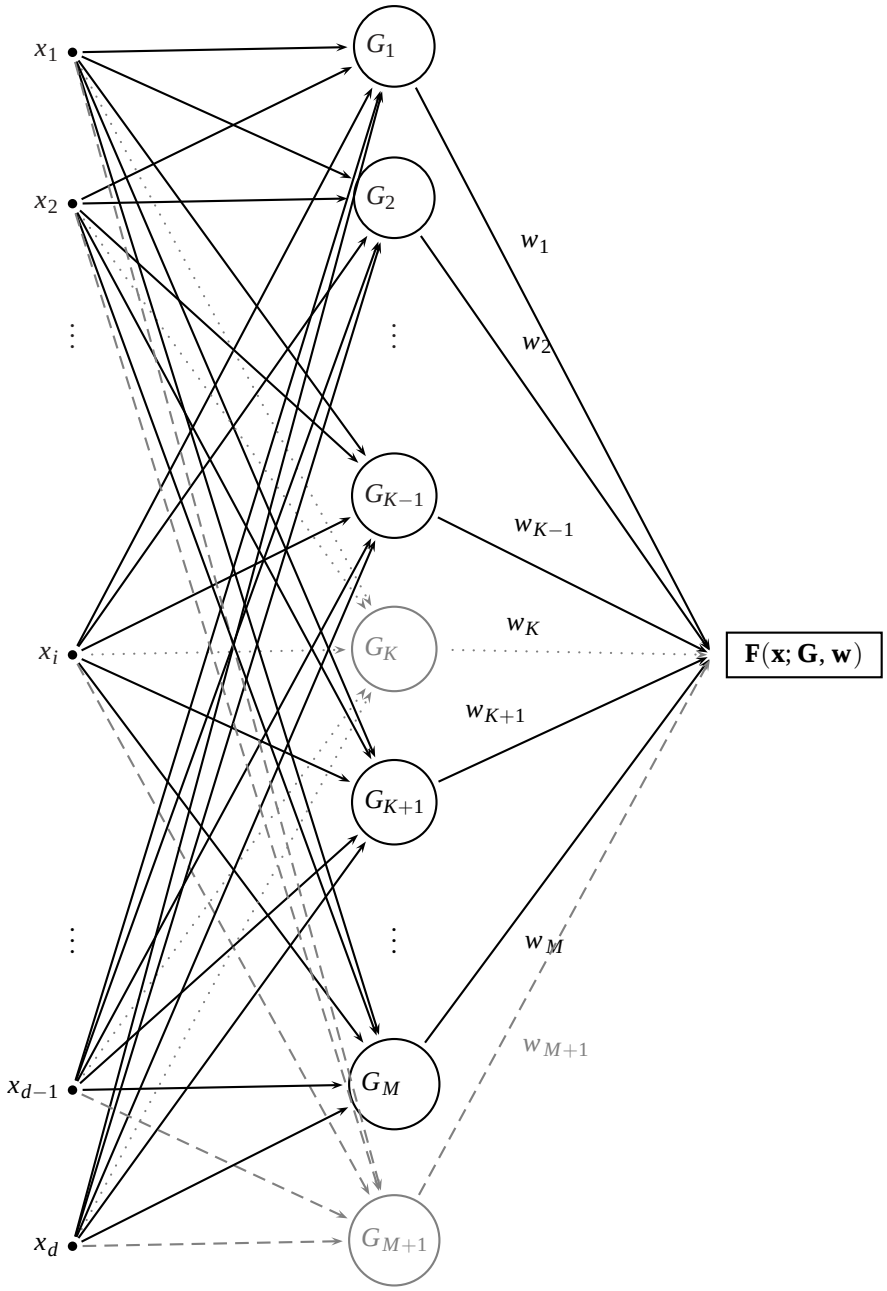
Poniżej zostaną omówione poszczególne elementy, składające się na sieć neuronową Incremental Network (IncNet).

4.3.1. Struktura sieci i funkcje transferu

Struktura sieci IncNet jest praktycznie taka sama jak sieci RBF (czy też RAN). Różni je jednakże fakt, iż struktura sieci RBF jest statyczna i nie podlega zmianom podczas uczenia, natomiast sieć IncNet jest dynamiczna i może się kurczyć i rozrastać. Widać to przy porównaniu praktycznie identycznych równań dla sieci RBF (2.2) i sieci IncNet (4.70), i częściowo różnych schematów tych sieci przedstawionych na rysunkach 2.1 i 4.11.

$$f(\mathbf{x}; \mathbf{w}, \mathbf{p}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x}, \mathbf{p}_i). \tag{4.70}$$

W standardowej sieci RBF, w sieci RAN i w pierwszej wersji sieci IncNet [146], jako funkcje bazowe używane były funkcje gaussowskie. Znaczny postęp uzyskano stosując w miejsce funkcji gaussowskich funkcje bicentralne, które powstają z produktu par funkcji sigmoidalnych. Pozwalają one na estymacje znacznie bardziej zróżnicowanych powierzchni przy użyciu mniejszej liczby funkcji bazowych. Standardowa funkcja bicentralna umożliwia niezależną kontrolę rozmycia jak i skosu w każdym wymiarze niezależnie. Zaproponowane różne rozszerzenia funkcji bicentralnych umożliwiają wzbogacenie możliwości o rotację gęstości w wielowymiarowej przestrzeni, czy uzyskiwanie semi-lokalnych gęstości, jak i desymetryzacji w podwymiarach. Szczegółowo funkcje bicentralne zostały już opisane i zilustrowane w podrozdziałach 1.4.6 i 1.4.7.



Rysunek 4.11: Struktura sieci IncNet. Sieć umożliwia dodawanie nowej funkcji bazowej G_{M+1} lub usuwanie pewnej funkcji bazowej G_K, jak również konkatenację neuronów.

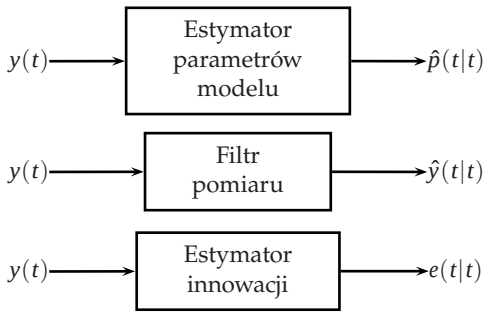
4.3.2. Rozszerzony filtr Kalmana

Jak przedstawiono w podrozdziale 4.2.8, na potrzeby uczenia sekwencyjnego funkcję błędu należy zdefiniować nieco inaczej niż robi się to dla typowych sieci neuronowych, czyli w postaci równania (2.14). Definicja błędu dla uczenia sekwencyjnego sieci RAN przez $\mathcal{F}$-projekcję, zdefiniowanej równaniem (4.43), może zostać uogólniona do minimalizacji funkcji:

$$E_{sq} = \int_{\mathcal{D}} |F^{(n)}(\mathbf{x}) - F^{(n-1)}(\mathbf{x})|^2 p^{(n-1)} d\mathbf{x} \; + \; \frac{1}{n-1} |y_n - F^{(n)}(\mathbf{x_n})|^2, \qquad (4.71)$$

gdzie $n-1$ i n odpowiadają określeniu kolejnych stanów modelu adaptacyjnego, $p^{(n-1)}$ jest rozkładem prawdopodobieństwa ostatnich $n-1$ obserwacji (wektorów wejściowych). Powyższa funkcja błędu została użyta do algorytmu RNLS (*ang. recursive nonlinear least squares*) w pracach Kadirkamanathana i Niranjana [145, 147]. Minimalizacja powyższej funkcji błędu może być aproksymowana przez użycie rozszerzonej wersji algorytmu filtru Kalmana (EKF) [39, 119, 220, 159]. Prowadzi to do zastąpienia algorytmu LMS algorytmem EKF do estymacji parametrów adaptacyjnych sieci. Pierwszy raz filtr EKF do estymacji parametrów sieci neuronowych został użyty przez Singhala i Wu w [233] dla sieci MLP; natomiast dla sieci typu RBF przez Kadirkamanathana w [145]. Z kolei z najnowszych i ciekawych zastosowań filtru EKF należy wspomnieć prace Freitas'a i Niranjana [87] do sekwencyjnej wersji modelu Support Vector Machines (por. podrozdział 3) i prace [88] prezentującą próbę modelowania giełdowego rynku finansowego.

Filtr Kalmana jest estymatorem trzech różnych wyjść dla danej obserwacji (wektora danych), co do której zakłada się, że jest obarczona pewnym szumem, o zerowej wartości oczekiwanej, z pewnym niezerowym odchyleniem standardowym. Pierwsze z wyjść stanowi estymator parametrów modelu. To właśnie główny element uczenia sieci. Kolejne dwa wyjścia to filtr pomiaru i estymator innowacji (lub błędu). Wyjścia te będą wykorzystywane do estymacji i do kontroli złożoności całości modelu.



Filtr EKF prowadzi estymacje parametrów *a posteriori* modelu $\mathbf{p}_n$, w oparciu

o ich poprzedni stan *a priori* $\mathbf{p}_{n-1}$ oraz błąd modelu e_n:

$$e_n = y_n - f(\mathbf{x}_n; \mathbf{p}_{n-1}) \tag{4.72}$$

i wartości wektora wzmocnienia Kalmana (*ang. Kalman gain*), które są pochodną macierzy kowariancji błędu a priori $\mathbf{P}_{n-1}$:

$$\mathbf{p}_n = \mathbf{p}_{n-1} + e_n \mathbf{k}_n. \tag{4.73}$$

Wektor $\mathbf{k}_n$ wzmocnienia Kalmana $\mathbf{k}_n$ jest wyznaczany przez:

$$\mathbf{k}_n = \mathbf{P}_{n-1}\mathbf{d}_n / R_y, \tag{4.74}$$

$\mathbf{d}_n$ jest wektorem gradientu funkcji realizowanej przez model względem adaptacyjnych parametrów modelu:

$$\mathbf{d}_n = \frac{\partial f(\mathbf{x}_n; \mathbf{p}_{n-1})}{\partial \mathbf{p}_{n-1}}, \tag{4.75}$$

natomiast R_y jest całkowitą wariancją modelu wyznaczaną poprzez:

$$R_y = R_n + \mathbf{d}_n^T \mathbf{P}_{n-1}\mathbf{d}_n, \tag{4.76}$$

R_n określa wariancję szumu pomiarów, kontroluje proces regularyzacji. Estymacja a priori macierzy kowariancji błędu przebiega zgodnie z równaniem:

$$\mathbf{P}_n = [\mathbf{I} - \mathbf{k}_n \mathbf{d}_n^T]\mathbf{P}_{n-1} + Q_0(n)\mathbf{I}, \tag{4.77}$$

$\mathbf{I}$ jest macierzą jednostkową. Człon $Q_0(n)\mathbf{I}$ spełnia rolę (drobnego) losowego skoku w kierunku gradientu, wprowadzając małą perturbację w procesie adaptacji parametrów modelu i zapobiegając zbyt szybkiej zbieżności, a czasami umożliwia *ucieczkę* z lokalnych minimów. $Q_0(n)$ może być bardzo małym dodatnim skalarem bądź funkcją monotonicznie malejącą, przyjmującą bardzo małe dodatnie wartości (stopniowe zastyganie). Dla przykładu:

$$Q_0(n) = \begin{cases} Q_0 & n = 1 \\ Q_0(n-1) \cdot Q_{des} & n > 1 \wedge Q_0(n-1) \cdot Q_{des} > Q_{min} \\ Q_{min} & n > 1 \wedge Q_0(n-1) \cdot Q_{des} \leq Q_{min} \end{cases}, \tag{4.78}$$

gdzie Q_0 jest wartością początkową dla $Q_0(n)$, Q_{des} definiuje szybkość zmniejszania pobudzania ($Q_{des} > 1$, zazwyczaj ok. 0.9988), a Q_{min} określa minimalną wartość $Q_0(n)$.

Wielką różnicę pomiędzy algorytmem EKF i LMS można zauważyć porównując ich równania (4.73) i (4.67), adaptacji parametrów p. W miejscu członu $\eta\mathbf{d}_n$ z algorytmu LMS, w filtrze EKF znajduje się wektor wzmocnienia Kalmana $\mathbf{k}_n$, który, jak widać z kolei w równaniu (4.74), wyznacza szybkość adaptacji każdego z parametrów nie tylko w zależności od wektora gradientu $\mathbf{d}_n$ (jak to

jest w algorytmie LMS czy stochastycznym spadku gradientu), lecz również w oparciu o macierz kowariancji $\mathbf{P}_{n-1}$. Właśnie to prowadzi do znacznie efektywniejszego procesu uczenia, radykalnie zmniejszając liczbę iteracji potrzebną do uzyskania zbieżności.

Słuszności wyboru filtra EKF jako algorytmu adaptacji parametrów modelu dowiodły już pierwsze jego zastosowania do adaptacji sieci RAN (RAN-EKF). W pracach [149, 150, 148] przedstawione zostały rezultaty użycia sieci RAN-EKF do aproksymacji funkcji Hermita (por. rozdział 7), przewidywania wartości szeregów czasowych, czy do klasyfikacji samogłosek. Rezultaty pokazują, iż użycie algorytmu EKF jest znacząco efektywniejsze i pozwala uzyskać większą generalizację często korzystając z mniejszej liczby funkcji bazowych.

4.3.3. Szybka wersja rozszerzonego filtru Kalmana

Macierz kowariancji $\mathbf{P}_n$, w miarę przybywania nowych neuronów, może urosnąć do sporych rozmiarów, gdy sieć uczy się klasyfikacji, bądź aproksymacji złożonych danych. Należy pamiętać, iż liczba elementów macierzy $\mathbf{P}_n$ to kwadrat liczby parametrów adaptacyjnych. To właśnie może okazać się zbyt obliczeniochłonnym procesem. Rozsądną decyzją okazało się zredukowanie sporej części macierzy kowariancji przyjmując, że korelacje pomiędzy parametrami różnych neuronów nie są tak istotne, jak korelacje pomiędzy parametrami tego samego neuronu. Takie uproszczenie prowadzi do sporych zmian macierzy $\mathbf{P}_n$ upraszczając ją do macierzy $\widetilde{\mathbf{P}}_n$, która tak naprawdę składa się z diagonalnego łańcucha podmacierzy $\widetilde{\mathbf{P}}_n^k$ (elementy poza diagonalnym łańcuchem są równe zeru):

$$\widetilde{\mathbf{P}}_n = \begin{bmatrix} \widetilde{\mathbf{P}}_n^1 & 0 & \dots & 0 & 0 \\ 0 & \widetilde{\mathbf{P}}_n^2 & \dots & 0 & 0 \\ & & \dots\dots\dots\dots & & \\ 0 & 0 & \dots & \widetilde{\mathbf{P}}_n^{M-1} & 0 \\ 0 & 0 & \dots & 0 & \widetilde{\mathbf{P}}_n^M \end{bmatrix}. \tag{4.79}$$

Podmacierze $\widetilde{\mathbf{P}}_n^k$ ($k = 1, 2, \dots, M$) są macierzami kowariancji związanymi z parametrami adaptacyjnymi k-tego neuronu.

Dzięki takiemu uproszczeniu, co prawda rezygnujemy z części informacji macierzy $\mathbf{P}_n$, ale za to znacznie zmniejszamy jej złożoność. Liczba parametrów macierzy $\mathbf{P}_n$ wynosi $n \cdot M \times n \cdot M$ (n to rozmiar przestrzeni wejściowej, a M liczba neuronów warstwy ukrytej), natomiast macierz $\widetilde{\mathbf{P}}_n$ ma $m^2 M$ parametrów. Tym samym dla danego problemu $\mathcal{P}$ złożoność macierzy $\mathbf{P}_n$ z $O(M^2)$ redukuje się do $O(M)$ dla macierzy $\widetilde{\mathbf{P}}_n$ (m jest stałe dla danego problemu $\mathcal{P}$).

Powyższa redukcja prowadzi również do przedefiniowania równań (4.72–

4.77) filtra EKF do postaci:

$$e_n = y_n - f(\mathbf{x}_n; \mathbf{p}_{n-1}) \qquad i = 1, \ldots, M \tag{4.80}$$

$$\mathbf{d}_n^i = \frac{\partial f(\mathbf{x}_n; \mathbf{p}_{n-1})}{\partial \mathbf{p}_{n-1}^i} \tag{4.81}$$

$$R_y = R_n + \mathbf{d}_n^{1\,T} \widetilde{\mathbf{P}}_{n-1}^1 \mathbf{d}_n^1 + \cdots + \mathbf{d}_n^{M\,T} \widetilde{\mathbf{P}}_{n-1}^M \mathbf{d}_n^M \tag{4.82}$$

$$\mathbf{k}_n^i = \widetilde{\mathbf{P}}_{n-1}^i \mathbf{d}_n^i / R_y \tag{4.83}$$

$$\mathbf{p}_n^i = \mathbf{p}_{n-1}^i + e_n \mathbf{k}_n^i \tag{4.84}$$

$$\widetilde{\mathbf{P}}_n^i = [\mathbf{I} - \mathbf{k}_n^i \mathbf{d}_n^{i\,T}] \widetilde{\mathbf{P}}_{n-1}^i + Q_0(n)\mathbf{I}. \tag{4.85}$$

4.3.4. Kryterium wystarczalności modelu

Kiedy aktualna architektura sieci nie jest już wystarczalna, aby akumulować nowe informacje?

Odpowiedź na to pytanie nie jest trywialna, czego dowodem mogą chyba być przedstawione poprzednio przykłady rozwiązania tego problemu. Powiększanie sieci o kolejne wagi czy neurony co pewien czas jest raczej naiwnym podejściem. Natomiast kryteria, których wyznaczenie wymaga przejrzenia (przeanalizowania) zachowania modelu dla wszystkich wektorów treningowych, można realizować jedynie od czasu do czasu w trakcie procesu adaptacyjnego (raczej nie częściej niż co epokę). Algorytmy bazujące (niemal) jedynie na wielkości popełnionego błędu dla danego wektora treningowego również nie są pozbawione wad, ponieważ podczas procesu uczenia nie możemy w pełni ufać modelowi, który przecież podlega uczeniu — sieć nie jest w pełni wiarygodna. Ta konkluzja zbliża do zdefiniowania ogólnego kryterium, które powinno na jednej szali położyć wielkość błędu, a na drugiej stopień naszego zaufania do aktualnego stanu sieci:

$$\frac{\text{błąd}}{\text{niepewność modelu}} < \alpha(M), \tag{4.86}$$

$\alpha(M)$ jest progiem zależnym od wielkości struktury modelu (liczby stopni swobody).

Statystyczną miarą niepewności dla *całości* rozpatrywanego modelu jest jego wariancja, która składa się z wariancji modelu (sieci neuronowej) i szumu danych (np. niedoskonałości pomiarów). Przyjmijmy iż wariancja szumu danych jest równa σ_{ns}^2, a wariancja samego modelu $Var(f(\mathbf{x}; \mathbf{p})) = \sigma_f^2(\mathbf{x})$.

Zakładając, iż błąd interpolacji pewnego nieznanego odwzorowania ma rozkład normalny, można oszacować, czy błąd leży w pewnym przedziale ufności, wyznaczonym przez niepewność modelu i szumu danych z pewnym, ustalonym stopniem zaufania. Wtedy hipotezę $\mathcal{H}_0$, iż aktualny model jest wystarczający, można zapisać jako:

$$\mathcal{H}_0 : \quad \frac{e^2}{Var[f(\mathbf{x}; \mathbf{p}) + \eta]} = \frac{e^2}{\sigma_f^2(\mathbf{x}) + \sigma_{ns}^2} < \chi_{M,\theta}^2, \tag{4.87}$$

gdzie $\chi^2_{M,\theta}$ jest rozkładem chi-kwadrat z progiem ufności θ% i M stopniami swobody (M – liczba funkcji bazowych). e oznacza błąd: $e = y - f(\mathbf{x};\mathbf{p})$ (por. równanie 4.72).

Gdy hipoteza $\mathcal{H}_0$ jest spełniona oznacza to, że bieżąca struktura sieci jest wystarczająca przy ustalonym stopniu zaufania. W przeciwnym przypadku bieżący model jest niewystarczający i należy rozszerzyć jego pojemność. W naszym przypadku oznacza to dodanie nowej funkcji bazowej do warstwy ukrytej.

Używając filtru EKF do estymacji parametrów modelu mamy automatycznie wyznaczoną całkowitą wariancję modelu:

$$R_y = Var[f(\mathbf{x};\mathbf{p}) + \sigma^2_{ns}], \tag{4.88}$$

R_y wyznaczane jest równaniem (4.76). Prowadzi to do ostatecznej definicji wystarczalności modelu:

$$\mathcal{H}_0 : \quad \frac{e_n^2}{R_y} < \chi^2_{n,\theta}. \tag{4.89}$$

Jeśli hipoteza $\mathcal{H}_0$ jest spełniona, sieć IncNet kontynuuje proces adaptacji parametrów, używając do tego filtru EKF (lub jego szybkiej wersji). W przeciwnym przypadku należy dodać nową funkcję bazową $G_{M+1}(\cdot)$ (neuron) do warstwy ukrytej z pewnymi wartościami początkowymi. Dla gaussowskich funkcji bazowych mamy:

$$\begin{aligned} w_{M+1} &= e_n = y_n - f(\mathbf{x}_n;\mathbf{p}_n) && (4.90)\\ \mathbf{t}_{M+1} &= \mathbf{x}_n && (4.91)\\ b_{M+1} &= b_0 && (4.92)\\ \mathbf{P}_n &= \begin{bmatrix} \mathbf{P}_n & 0 \\ 0 & P_0\mathbf{I} \end{bmatrix}, && (4.93) \end{aligned}$$

stała b_0 jest początkową wartością definiującą rozmycie, a P_0 określa nowe parametry diagonalne macierzy kowariancji (P_0 zazwyczaj jest równe 1).

Dla funkcji bicentralnych dodanie nowego neuronu wygląda podobnie:

$$\begin{aligned} w_{M+1} &= e_n = y_n - f(\mathbf{x}_n;\mathbf{p}_n) && (4.94)\\ \mathbf{t}_{M+1} &= \mathbf{x}_n && (4.95)\\ \mathbf{b}_{M+1} &= \mathbf{b}_0 && (4.96)\\ \mathbf{s}_{M+1} &= \mathbf{s}_0 && (4.97)\\ \mathbf{P}_n &= \begin{bmatrix} \mathbf{P}_n & 0 \\ 0 & \mathbf{P}_0\mathbf{I} \end{bmatrix}, && (4.98) \end{aligned}$$

wektory stałych $\mathbf{b}_0$ i $\mathbf{s}_0$ definiują początkowe wartości rozmyć i skosów w wielowymiarowej przestrzeni.

Tak zdefiniowany statystyczny test wystarczalności modelu jest bardzo efektywny, a dla jego zweryfikowania nie potrzeba wyznaczać żadnych dodatkowych elementów. Co najważniejsze, taki test można stosować w każdej iteracji algorytmu uczenia, tj. dla każdej prezentacji wektora treningowego. Jak się okazało w praktyce tak zdefiniowany test wystarczalności prowadzi do lepszego wykorzystania posiadanych już funkcji bazowych przez model, jak i do uzyskania lepszego poziomu generalizacji. Widać to porównując sieć IncNet z siecią RAN [204], czy RAN-EKF [149], co można zobaczyć w rozdziale 7 (a szczególnie w podrozdziale 7.3) jak i pracach [136, 141, 146].

4.3.5. Usuwanie neuronów

Podczas procesu uczenia często dochodzi do sytuacji, w której pewne wagi czy neurony przestają odgrywać istotną rolę. Takie okoliczności sprawiają, iż model i tak musi prowadzić adaptację takich neuronów pomimo, iż nie są one przydatne. Co więcej mogą one być przyczyną przeuczenia się naszego modelu.

W powyższej części rozdziału o ontogenicznych sieciach neuronowych można było zauważyć, że prawie żaden model nie był zdolny do powiększania i zmniejszania swojej struktury, a jeśli już mógł zmniejszać i zwiększać swoją strukturę, to nie mógł robić obu czynności podczas uczenia i zazwyczaj usuwanie neuronów dokonywane było po procesie uczenia. Umożliwienie modelowi zwiększania i zmniejszania swojej struktury **podczas** procesu uczenia sprawia, że model może (gdy potrzebuje) powiększyć swoją strukturę jak również może dokonać jej zmniejszenia poprzez usunięcie części struktury lub zastąpienie pewnej części struktury inną o mniejszej złożoności. Taki mechanizm regulacji złożoności modelu jest znacznie bardziej racjonalny, niż umożliwienie jedynie powiększania albo zmniejszania struktury modelu.

Jak pokaże poniższa cześć podrozdziału można tak zdefiniować algorytm sprawdzania przydatności poszczególnych neuronów, aby można było zwiększać, jak i zmniejszać architekturę sieci, dostosowując złożoność modelu uczącego do złożoności napływających do modelu danych.

Poniżej przedstawiony algorytm opiera się o analizę wartości współczynników istotności (przydatności) dla neuronów sieci. Tym samym, w pierwszym etapie należy ustalić sposób wyznaczania współczynników istotności dla poszczególnych neuronów warstwy ukrytej.

Zakładając, iż funkcje gęstości, opisywane przez poszczególne funkcje transferu neuronów warstwy ukrytej, są podobne (np. zlokalizowane i wypukłe), współczynniki istotności poszczególnych neuronów warstwy ukrytej można zdefiniować przez stosunek wielkości wagi do wariancji tej wagi. Taki stosunek faworyzuje silne wagi (tj. wagi o istotnym wpływie), których tendencje zmian wartości są małe (tj. wagi *nauczone*). W ten sposób zdefiniowane współczynniki można zapisać matematycznie:

$$s_i = \frac{w_i^2}{\sigma_{w_i}}, \tag{4.99}$$

gdzie σ_{w_i} oznacza wariancję i-tej wagi (związaną z i-tym neuronem warstwy ukrytej).

Oczywiście jeśli miałoby dojść do usunięcia jakiegokolwiek neuronu, to należy wybrać ten, dla którego współczynnik istotności będzie najmniejszy:

$$L_1 = \min_i s_i = \min_i \frac{w_i^2}{\sigma_{w_i}}. \tag{4.100}$$

Używając rozszerzonego filtra Kalmana jako estymatora parametrów modelu do wyznaczania wariancji σ_{w_i}, można użyć macierzy kowariancji $\mathbf{P}_n$.

W tym celu najpierw przyjmijmy, iż parametry sieci w wektorze parametrów p_n są uszeregowane w poniższy sposób:

$$\mathbf{p}_n = [w_1, \ldots, w_M, \ldots]^T. \tag{4.101}$$

Pierwsze parametry $\mathbf{p}_n$ to wagi pomiędzy warstwą ukrytą i wyjściową, po nich znajdują się pozostałe parametry wektora $\mathbf{p}_n$. Wtedy macierz kowariancji $\mathbf{P}_n$ wygląda tak:

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_w & \mathbf{P}_{wv} \\ \mathbf{P}_{wv}^T & \mathbf{P}_v \end{bmatrix}, \tag{4.102}$$

gdzie $\mathbf{P}_w$ jest macierzą kowariancji pomiędzy wagami, macierz $\mathbf{P}_{wv}$ jest macierzą kowariancji pomiędzy wagami i innymi parametrami, a macierz $\mathbf{P}_v$ jest macierzą kowariancji tylko pomiędzy innymi parametrami (tj. bez wag).

Korzystając z powyższego ułożenia parametrów w macierzy $\mathbf{P}$ wzór (4.100) można sprowadzić do postaci:

$$L_1 = \min_i s_i = \min_i \frac{w_i^2}{[\mathbf{P}_w]_{ii}}. \tag{4.103}$$

Oczywiście wartość L_1 można wyznaczać dla rozszerzonego filtra Kalmana, jak i dla jego szybkiej wersji, opisanej równaniami (4.80–4.85).

Pozostaje problem podjęcia decyzji, czy w ogóle należy usunąć jakiś neuron w danej k-tej iteracji algorytmu? W tym celu posłużmy się ogólną konkluzją, iż najczęściej nieistotne neurony występują w modelach niestabilnych, a z kolei wartościowe neurony mamy w dobrze nauczonych, stabilnych sieciach. Ta konkluzja wiedzie do ogólniejszego kryterium, które można zapisać jako:

$$\frac{L_1}{R_y} < \chi^2_{1,\vartheta}, \tag{4.104}$$

gdzie $\chi^2_{n,\vartheta}$ jest rozkładem chi-kwadrat z poziomem ufności ϑ% z jednym stopniem swobody, a R_y jest całkowitą wariancją modelu, która w przypadku użycia filtra Kalmana jest wyznaczana wzorem (4.76).

Jeśli powyższe kryterium jest spełnione oznacza to, że należy dokonać usunięcia neuronu, dla którego uzyskano najmniejszą wartość spośród współczynników istotności.

Tak zdefiniowane statystyczne kryterium usuwania neuronów warstwy ukrytej dla sieci typu RBF może być stosowane w praktyce z iteracji na iterację, szczególnie, gdy prowadzimy estymacje parametrów sieci za pomocą rozszerzonego filtra EKF. Z kolei, używając jednocześnie podczas uczenia sieci, kontroli wystarczalności i kontroli przydatności poszczególnych neuronów, model na bieżąco stara się estymować złożoność struktury sieci do złożoności napływających sekwencyjnie danych uczących. Takie postępowanie bardzo dobrze wpływa na końcowy poziom generalizacji.

Współpracujące statystyczne kryteria wystarczalności sieci i przydatności neuronów są używanie w sieci IncNet, a badania, wykonane dla różnych danych, wykazują dużą skuteczność ich działania [141, 140, 133, 136, 135, 134]. Rezultaty te zostały też umieszczone w rozdziale 7. Szczególnie widoczną kooperację kryteriów kontroli złożoności modelu widać w analizie danych psychometrycznych, podczas procesu uczenia, co można zobaczyć w podrozdziale 7.2.1.

4.3.6. Łączenie neuronów

Podczas rozwoju modeli sztucznych sieci neuronowych powstały różne metody usuwania neuronów. Metody usuwania neuronów opierają się o sposoby wyznaczania neuronów, które przestały być przydatne lub też ich wpływ na działanie sieci jest znikomy lub wręcz zły. Jednakże nierzadko mamy do czynienia z sytuacją, w której regiony decyzyjne, powstałe przez kooperację wielu neuronów, mogłyby zostać zastąpione przez mniejszą sieć, nie zmniejszając właściwości klasyfikacji, czy aproksymacji, a nawet dając możliwość polepszenia jakości generalizacji, dzięki zbliżeniu złożoności modelu adaptacyjnego do złożoności danych uczących (problemu). Dlatego też możliwość łączenia dwóch (lub większej liczby, choć to prowadzi do bardziej złożonego problemu) neuronów może wpłynąć pozytywnie, nie zmieniając przy tym istotnie powierzchni aproksymowanej funkcji, czy powierzchni regionów decyzji dla klasyfikacji.

Powyżej opisanego połączenia dla pewnych dwóch neuronów i, j oraz zastąpienia ich przez nowy neuron *new*, można dokonać przy założeniu, iż funkcje transferu neuronów są zlokalizowane, ciągłe i jest spełnione poniższe kryterium:

$$\frac{\int_{\mathbf{d}\subseteq\mathcal{D}^n} \left|\bar{\phi}_i(\mathbf{x}) + \bar{\phi}_j(\mathbf{x}) - \bar{\phi}_{new}(\mathbf{x})\right| \, \mathrm{d}\mathbf{x}}{\int_{\mathbf{d}\subseteq\mathcal{D}^n} \left|\bar{\phi}_i(\mathbf{x}) + \bar{\phi}_j(\mathbf{x})\right| \, \mathrm{d}\mathbf{x}} < \alpha, \tag{4.105}$$

d jest podprzestrzenią aktywności funkcji transferu $\phi_i(\mathbf{x})$ i $\phi_j(\mathbf{x})$, a $\bar{\phi}_i(\mathbf{x}) = w_i\phi_i(\mathbf{x})$ i $\phi_j(\mathbf{x}) = w_j\phi_j(\mathbf{x})$ są przeskalowanymi funkcjami przez wagi wyjściowe. Przeskalowana funkcja transferu nowego neuronu $\bar{\phi}_{new}(\mathbf{x})$ ($\bar{\phi}_{new}(\mathbf{x}) = w_{new} \cdot \phi_{new}(\mathbf{x})$) dla $\mathbf{x}$ spoza podprzestrzeni d musi być w przybliżeniu równa zeru. Współczynnik α określa błąd, jaki uznaje się za dopuszczalny. W przypadku użycia filtru Kalmana (i nie tylko) można współczynnik α uzależnić liniowo od szumu pomiarów R_n lub od całkowitej wariancji modelu R_y (patrz równanie 4.76).

Kryterium opisane równaniem (4.105) jest trudno zastosować w ogólnym

przypadku, lecz gdy funkcje transferu warstwy ukrytej są separowalne wymiarowo (na przykład dla funkcji gaussowskich lub bicentralnych), można dokonać uproszczenia tego kryterium do postaci:

$$\frac{\int_{d_1\subseteq\mathcal{D}_1}\cdots\int_{d_N\subseteq\mathcal{D}_N}\left[\bar{\phi}_i(\mathbf{x})+\bar{\phi}_j(\mathbf{x})-\bar{\phi}_{new}(\mathbf{x})\right]^2\ \mathrm{d}x_1\ldots\mathrm{d}x_N}{\int_{d_1\subseteq\mathcal{D}_1}\cdots\int_{d_N\subseteq\mathcal{D}_N}\left[\bar{\phi}_i(\mathbf{x})+\bar{\phi}_j(\mathbf{x})\right]^2\ \mathrm{d}x_1\ldots\mathrm{d}x_N}<\alpha. \tag{4.106}$$

Powyższe kryterium może zostać użyte w formie analitycznej lub numerycznej.

W innych przypadkach kryterium łączenia neuronów może zostać uproszczone do wyznaczania ważonego błędu średniokwadratowego w oparciu o chmurę punktów o gaussowskim rozkładzie rozmieszczonych na obszarze aktywności neuronów i, j:

$$\frac{\sum_{d\in\mathbf{d}}\left[\bar{\phi}_i(\mathbf{x})+\bar{\phi}_j(\mathbf{x})-\bar{\phi}_{new}(\mathbf{x})\right]^2}{\sum_{d\in\mathbf{d}}\left[\bar{\phi}_i(\mathbf{x})+\bar{\phi}_j(\mathbf{x})\right]^2}<\alpha. \tag{4.107}$$

W przypadku użycia funkcji bicentralnych jako funkcji transferu, poszczególne parametry funkcji bicentralnej mogą być wyznaczone jak poniżej:

$$\mathbf{w}_{new} = \frac{\bar{\phi}(\mathbf{t}_i,\mathbf{t}_i)\cdot\bar{\mathbf{P}}_i+\bar{\phi}(\mathbf{t}_j,\mathbf{t}_j)\cdot\bar{\mathbf{P}}_j}{\phi(\mathbf{t}_{new},\mathbf{t}_{new})} \tag{4.108}$$

$$\mathbf{t}_{new,k} = \frac{1}{\mathbf{M}}\int_{d\in\mathbf{D}} x_k\ [\bar{\phi}(\mathbf{x},\mathbf{t}_i)+\bar{\phi}(\mathbf{x},\mathbf{t}_j)]\ \mathrm{d}\mathbf{x} \tag{4.109}$$

lub (4.110)

$$\mathbf{t}_{new} = \mathbf{t}_i\cdot\bar{\mathbf{P}}_i+\mathbf{t}_j\cdot\bar{\mathbf{P}}_j \tag{4.111}$$

$$\mathbf{s}_{new} = \mathbf{s}_i\cdot\bar{\mathbf{P}}_i+\mathbf{s}_j\cdot\bar{\mathbf{P}}_j \tag{4.112}$$

$$\mathbf{b}_{new} = \begin{cases}\mathbf{b}_i & \text{neuron } j \text{ wewnątrz neuronu } i,\\ \mathbf{b}_j & \text{neuron } i \text{ wewnątrz } j,\\ \frac{\mathbf{b}_i+\mathbf{b}_j+|\mathbf{t}_i-\mathbf{t}_j|}{2} & \text{w p. p.}\end{cases}, \tag{4.113}$$

gdzie M jest zdefiniowane przez

$$\mathbf{M}=\int_{d\in\mathbf{D}}[\bar{\phi}(\mathbf{x},\mathbf{t}_i)+\bar{\phi}(\mathbf{x},\mathbf{t}_j)]\ \mathrm{d}\mathbf{x}=\mathbf{P}_i+\mathbf{P}_j, \tag{4.114}$$

natomiast $\bar{\mathbf{P}}_i$ i $\bar{\mathbf{P}}_j$ poprzez:

$$\bar{\mathbf{P}}_i = \frac{\mathbf{P}_i}{(\mathbf{P}_i+\mathbf{P}_j)}, \tag{4.115}$$

$$\bar{\mathbf{P}}_j = \frac{\mathbf{P}_j}{(\mathbf{P}_i+\mathbf{P}_j)}, \tag{4.116}$$

gdzie $\mathbf{P}_i$ i $\mathbf{P}_j$ są zdefiniowane jako

$$\mathbf{P}_i = \int_{d\in\mathbf{D}} \bar{\phi}(\mathbf{x}, \mathbf{t}_i)\ \mathrm{d}\mathbf{x} \tag{4.117}$$

$$\mathbf{P}_j = \int_{d\in\mathbf{D}} \bar{\phi}(\mathbf{x}, \mathbf{t}_j)\ \mathrm{d}\mathbf{x}. \tag{4.118}$$

Pozostaje pytanie, kiedy próbować, czy kryterium będzie spełnione, i dla jakich par neuronów sprawdzać, czy kryterium jest spełnione. Jednym ze sposobów jest sprawdzanie kryterium co epokę dla każdego neuronu i ($i = 1, \ldots, M$) i neuronu j, wybranego w następujący sposób:

$$j = \arg\max_k\ \phi(\mathbf{t}_k, \mathbf{t}_i). \tag{4.119}$$

Innym, kosztowniejszym sposobem jest próba łączenia jednej pary neuronów podczas każdej (p-tej) iteracji algorytmu adaptacji. W tym przypadku wybiera się pierwszy neuron i:

$$i = \arg\max_k\ \phi(\mathbf{x}_p, \mathbf{t}_k), \tag{4.120}$$

gdzie $\mathbf{x}_p$ jest wektorem wejściowym prezentowanym w p-tej iteracji algorytmu. Następnie wyznacza się drugi neuron j:

$$j = \arg\max_{k\neq i}\ \phi(\mathbf{x}_p, \mathbf{t}_k), \tag{4.121}$$

po czym bada się, czy jest spełnione kryterium łączenia neuronów dla neuronów i i j.

Gdy, dla pewnej pary neuronów kryterium łączenia będzie spełnione (niezależnie od tego czy sprawdzanie następuje co iterację, czy co epokę), następuje zastąpienie owej pary neuronów nowym neuronem o parametrach opisanych wzorami (4.108–4.113).

Powyżej zaproponowany sposób kontroli złożoności umożliwia zmniejszenie struktury sieci neuronowej poprzez unifikację jej fragmentów, w oparciu o analizę neuronów, które wcale nie muszą być nieprzydatne. Wręcz przeciwnie, nierzadko da się zastąpić dwa neurony, które odgrywają istotną rolę. Tak zdefiniowaną metodę łączenia neuronów można stosować praktycznie do każdej sieci typu RBF, jak do innych modeli opartych o ciągłe i zlokalizowane funkcje transferu.

4.3.7. Wykorzystanie sieci IncNet w klasyfikacji

Sieć IncNet, w której adaptacji podlegają nie tylko wagi wyjściowe, może mieć tylko jedno wyjście. Gdyby pozostać jedynie przy adaptacji wag pomiędzy warstwą ukrytą i wyjściową, wtedy łatwo można sformułować sposób uczenia sieci z wieloma wyjściami (zrobił to Kadirkamanathan w pracy [148]). Jednakże rezygnacja z adaptacji położeń funkcji bazowych, rozmyć czy skosów i ewentualnie

innych parametrów dla funkcji bicentralnych, znacząco zubaża całość modelu i zmniejsza jego potencjalne możliwości. Jeszcze gorszym rozwiązaniem (wręcz niedopuszczalnym) jest, aby sieć wykorzystywała jedno wyjście, a wartość tego wyjścia dla pewnej danej, po zaokrągleniu, byłaby interpretowana jako numer klasy, do której owa dana zostałaby przypisana. Liniowy układ wyjścia (klasa $1, 2, \ldots, K$) nie odzwierciedla w żaden sposób rzeczywistych zależności pomiędzy klasami, a raczej wprowadza wręcz arbitralne, nie istniejące i niestety niedopuszczalne zależności.

Z powyższych powodów najciekawszym rozwiązaniem wydaje się pomysł zbudowania klastra niezależnych sieci IncNet, a zadaniem każdej z podsieci byłoby wyspecjalizowanie się w rozpoznawaniu jednej (k-tej) klasy.

W tym celu ze zbioru par uczących:

$$\mathcal{S} = \{\langle \mathbf{x}_1, y_1 \rangle, \langle \mathbf{x}_2, y_2 \rangle, \ldots \langle \mathbf{x}_N, y_N \rangle\}, \tag{4.122}$$

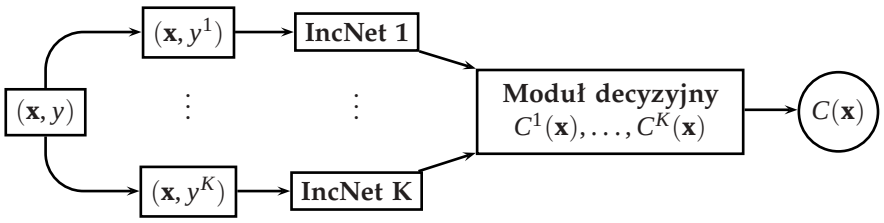
produkujemy K zbiorów, po jednym dla każdej klasy:

$$\mathcal{S}^k = \{\langle \mathbf{x}_1, y_1^k \rangle, \langle \mathbf{x}_2, y_2^k \rangle, \ldots \langle \mathbf{x}_N, y_N^k \rangle\} \quad k = 1, 2, \ldots, K, \tag{4.123}$$

gdzie y_i^k przyjmuje wartości 1 lub 0:

$$y_i^k = \begin{cases} 1 & y_i = k \\ 0 & y_i \neq k \end{cases} \quad i = 1, 2, \ldots, N. \tag{4.124}$$

Tak zdefiniowane zbiory $\mathcal{S}^k$ są zbiorami uczącymi dla poszczególnych sieci IncNet, które razem tworzą komitet. Schemat tak zdefiniowanego komitetu sieci IncNet przedstawiono na rysunku 4.12.



Rysunek 4.12: Komitet sieci IncNet w zastosowaniu do problemów klasyfikacyjnych.

Dla danej obserwacji $\mathbf{x}$ każda z sieci produkuje wartość wyjściową $C^i(\mathbf{x})$ (dla $i = 1, 2, \ldots, K$). Wartość $C^i(\mathbf{x})$ zbliżona do zera oznacza, iż obserwacja $\mathbf{x}$ nie odpowiada klasie i. Natomiast wartość $C^i(\mathbf{x})$ zbliżona do jedynki oznacza, iż obserwacja $\mathbf{x}$ odpowiada klasie i.

Moduł decyzyjny, widoczny na rysunku 4.12, podejmuje ostateczną decyzję i przypisuje pewien wektor $\mathbf{x}$ do jednej klasy $C(\mathbf{x})$. Podejmowanie takiej decyzji

może przebiegać zgodnie z poniższą definicją:

$$C(\mathbf{x}) = \arg\max_i \; C^i(\mathbf{x}). \tag{4.125}$$

Tym samym moduł decyzyjny wybiera sieć, której aktywacja była największa, tj. najbardziej odpowiadająca danej obserwacji.

Jednakże przydatność takiego klastra podsieci IncNet wcale nie musi sprowadzać się do obserwacji jedynie wartości $C(\mathbf{x})$. Bardzo przydatne jest wprowadzenie następującej renormalizacji:

$$p(C^i|\mathbf{x}) = \frac{\sigma(C^i(\mathbf{x}) - \frac{1}{2})}{\sum_{j=1}^{K} \sigma(C^j(\mathbf{x}) - \frac{1}{2})}, \tag{4.126}$$

gdzie $\sigma(x) = 1/(1 + \exp(-\gamma x))$, a γ jest stałą (np. około 10). Prowadzi to do aproksymacji prawdopodobieństwa przynależności wektora $\mathbf{x}$ do klasy i.

Można zdefiniować p_{max}:

$$p_{max}(\mathbf{x}) = \max_i \; p(C^i|\mathbf{x}), \tag{4.127}$$

jako prawdopodobieństwo najbardziej prawdopodobnej klasy.

Natomiast $C(\mathbf{x})$ zdefiniowane poprzez

$$C(\mathbf{x}) = \arg\max_i \; p(C^i|\mathbf{x}) \tag{4.128}$$

wyznacza po prostu najbardziej prawdopodobną klasę dla danej obserwacji $\mathbf{x}$.

Warto jednak pamiętać o fakcie, iż mamy do dyspozycji wszystkie wartości $p(C^i|\mathbf{x})$ dla $i = 1, 2, \ldots, K$, a nie tylko wartość prawdopodobieństwa najbardziej prawdopodobnej klasy. Choć z pewnością najważniejsze jest wyznaczenie klasy, dla której prawdopodobieństwo jest największe, jednak warto także przyjrzeć się klasie (może nawet więcej niż jednej), która może stanowić istotną alternatywę. Jeśli wartości tak wyselekcjonowanych prawdopodobieństw są zbliżone oznacza to, że klasyfikacja nie jest jednoznaczna. Obserwując wszystkie wartości prawdopodobieństw $p(C^i|\mathbf{x})$, z pewnością łatwo stwierdzić, które spośród klas można uznać za nieprawdopodobne, następnie można je wykluczyć z dalszej analizy (porównaj analizę rezultatów przedstawioną w podrozdziale 7.2.1).

4.3.8. Charakterystyka parametrów kontroli procesu adaptacji sieci IncNet

Powyżej opisane główne moduły sieci IncNet, na które składają się algorytm uczenia, funkcje transferu i metody kontroli złożoności sieci, posiadają różne parametry umożliwiające wpływanie na proces adaptacji. Do takich parametrów należą: współczynnik określający poziom szumu, funkcja $Q(n)$, która reguluje szybkość zbieżności filtra Kalmana, początkowe parametry funkcji transferu,

współczynniki określające stopnie ufności kryteriów wystarczalności i niewystarczalności struktury.

Większość z tych parametrów jest stała lub zdeterminowana danymi uczącymi lub charakterystykami procesu uczenia. Na przykład poziom wariancji szumu pomiarów R_n powinien być znany ze względu na sposób dokonanych pomiarów lub, jeśli nie jest znany można (czy też trzeba) dokonać jego oszacowania (filtr Kalmana nie jest zbyt wrażliwy na poziom wariancji; dzięki temu (ewentualne) oszacowanie wariancji szumu wcale nie ma krytycznego wpływu na proces estymacji).

Początkowe parametry funkcji transferu określają niejako *rozdzielczość* modelu, ale i początkową gładkość estymacji (poprzez dobór skosu funkcji transferu — jest to szczególnie widoczne dla funkcji bicentralnych). Rozdzielczość jest pochodną wielkości obszaru aktywnego dla zlokalizowanej funkcji transferu. Gdy obszar ten jest bardzo mały, mamy do czynienia z dużą rozdzielczością, lecz wtedy potrzeba wielu neuronów w warstwie ukrytej, by docelowy obszar przestrzeni wielowymiarowej został odpowiednio pokryty, lub też potrzeba by więcej czasu, aby funkcje mogły zostać odpowiednio *rozciągnięte*. I odwrotnie gdy obszar pokrywany przez funkcję jest bardzo duży, zbyt szybko może dojść do znacznego nakrywania się różnych funkcji transferu. Wtedy neurony zamiast kooperować zaczynają wręcz *walczyć* ze sobą. Dlatego też dobór parametrów, określających początkowe wartości parametrów funkcji transferu dobrze powiązać z analizą danych i wszelką dostępną wiedzą *a priori*, jak i próbą użycia różnych wartości dla sprawdzenia jakości działania procesu adaptacji.

Parametry kryteriów kontroli złożoności struktury mogą być niemal niezmienne, ze względu na różne aplikacje, albo wyznaczanie wartości tychże parametrów może być powiązane (liniowo) z poziomem wariancji danych.

Powyżej opisane fakty pokazują, iż przy zrozumieniu znaczenia odpowiednich współczynników, ich dobór nie jest trudny, a z pewnością korzystnie może wpłynąć na proces adaptacji — im lepsza wiedza *a priori* wszczepiona w początkowy model, tym większe prawdopodobieństwo uzyskania lepszego poziomu generalizacji.

Sieć IncNet rozbudowano o jeszcze kilka mechanizmów kontroli. Do ważniejszych z pewnością należy stopniowe *chłodzenie* dynamiki zmiany architektury. Polega to na wyostrzaniu kryteriów, przy jakich sieć może zmieniać swoją architekturę wraz z upływem procesu uczenia, zmuszając sieć do większej stabilizacji. Taki mechanizm może być przydatny, gdy dane, na których system jest poddawany uczeniu, są mocno zdyskretyzowane (np. realne wartości są ciągłe, natomiast w danych znajdują się wartości poddane dyskretyzacji, w wyniku której uzyskuje się jedynie kilka różnych wartości) lub mocno zaszumione.

Innym mechanizmem, przydatnym na przykład w wyżej opisanych sytuacjach, jest zapamiętywanie najefektywniejszych wersji sieci (architektura i wartości wszelkich parametrów adaptacyjnych) podczas prowadzenia procesu adaptacji. W tym celu, co pewną liczbę iteracji, dokonuje się testu na podstawie którego ocenia się bieżący model i porównuje z aktualnie najlepszym modelem,

po czym lepszy zostaje zapamiętany. Zapamiętywaniu podlegają wszelkie parametry modelu wraz ze strukturą sieci. Oczywiście przy ocenie wykorzystywane są jedynie dane treningowe.

Przy korzystaniu z dość dużych zbiorów danych uczących okazuje się, iż warto, w miarę prowadzenia procesu uczenia, zmniejszać początkowy obszar (dyspersje) zlokalizowanych funkcji transferu. Dzięki takiemu zabiegowi w pierwszej fazie adaptacji system dysponuje funkcjami o mniejszej rozdzielczości, natomiast w dalszej części ma do dyspozycji funkcje o większej rozdzielczości, które w już istniejącej strukturze mogą pokryć mniejsze obszary w bardziej precyzyjny sposób.

Poza opisanymi metodami kontroli zaimplementowano jeszcze parę mniej istotnych. Na koniec wspomnę o możliwości określenia maksymalnej liczby neuronów w warstwie ukrytej. Ten mechanizm, jak i inne nie wspomniane w pracy, są przydatne raczej na poziomie wstępnej oceny samych danych niż we właściwym procesie uczenia.

4.4. Sieć neuronowa optymalnych funkcji transferu

Sieci neuronowe używają niemal zawsze neuronów tego samego typu w każdej warstwie. Jednak wcale nie musi to oznaczać, że taka architektura ma optymalną złożoność i dokładność klasyfikacji, czy aproksymacji. Tak naprawdę już ściśle wybrany typ funkcji transferu dla danej warstwy narzuca pewne ograniczenia, które odzwierciedlą się w ostatecznym rozwiązaniu.

Dlatego też poniższy rozdział poświęcony został sieciom neuronowym o architekturach (w sensie liczby neuronów, połączeń i typów neuronów), które zostają zoptymalizowane dla danego problemu. W sieciach tych każdy neuron może implementować inną funkcję transferu, a z kolei całość struktury sieci jest nadzorowana przez statystyczne kryteria, bądź poprzez dodanie odpowiednich członów do funkcji błędu.

Sztuczne sieci neuronowe aproksymują nieznane odwzorowanie F^* pomiędzy parami $\langle \mathbf{x}_i, y_i \rangle$, dla $i = 1, \ldots, n$ ze zbioru $\mathcal{S}$. Naszym celem jest, jak zazwyczaj, aby $F(\mathbf{x}_i) = y_i$, gdzie $F(\cdot)$ reprezentuje funkcję, którą realizuje cała sieć neuronowa. Dokładność z jaką uda się zbudować sieć, która będzie stosunkowo bliska wyżej postawionemu celowi, zależy od algorytmu uczenia, liczby warstw, neuronów, połączeń między neuronami i od typu funkcji realizowanej przez każdy neuron. Aby uniknąć wspominanego już niedouczenia lub przeuczenia danych przez sieć, powinna ona złożonością architektury odpowiadać złożoności danych [140, 70].

Złożoność modelu może być kontrolowana na różne sposoby. Na przykład przy wykorzystaniu regularyzacji, czy statystycznych kryteriów kontroli złożoności, jak i przez odpowiedni wybór funkcji transferu — patrz rozdział 4.3. Wszystkie te metody będą wykorzystane w sieciach optymalnych funkcji transferu opisanych w tym rozdziale. Zamieszczone przykłady pokażą rozwiązania różnych problemów.

4.4.1. Sieć optymalnych funkcji transferu (OTFN)

Dokładność klasyfikacji sieci MLP i RBF może się znacznie różnić, raz na korzyść jednego modelu, innym razem na korzyść drugiego. [70]. Jest tak, ponieważ niektóre dane łatwiej dają się aproksymować za pomocą funkcji sigmoidalnych (1.5) z iloczynem skalarnym jako aktywacją, a inne za pomocą funkcji gaussowskich, bazujących na odległości jako aktywacji (1.65).

Dobrym rozwiązaniem, opisanym w rozdziale 4.3 i 1.4.6, jest użycie funkcji bicentralnych, których możliwości są znacznie większe. Używają one $3N$ parametrów, podczas gdy funkcja gaussowska używa $2N$ lub $N+1$, a funkcja sigmoidalna używa $N+1$. Dlatego też proponujemy zastosowanie optymalizacji funkcji transferu w procesie uczenia sieci neuronowej. Taka sieć może mieć ogólną definicję w postaci:

$$F(\mathbf{x}) = o\left(\sum_i w_i h_i[A_i(\mathbf{x};\mathbf{p}_i)]\right), \tag{4.129}$$

gdzie $h_i(A_i(\cdot)) \in \mathcal{H}$ ($\mathcal{H}$ jest pewnym zbiorem funkcji bazowych) jest i-tą funkcją transferu, $h_i(\cdot)$ jest funkcją wyjścia, a $A_i(\cdot)$ jest funkcją aktywacji. p_i jest wektorem parametrów adaptacyjnych dla neuronu i.

Funkcją $o(\cdot)$, która jest funkcją wyjścia całej sieci, może być funkcja tożsamościowa lub funkcja sigmoidalna. Warto także rozważyć użycie funkcji okienkującej, gdyż umożliwia ona aktywacje neuronu na pewnym przedziale (który może być dowolnie duży). Użycie funkcji tożsamościowej skraca czas obliczeń związany z procesem uczenia (niestety nie zawsze można pozostać przy funkcji tożsamościowej, np. gdy chcemy aby wyjście odpowiadało prawdopodobieństwu).

Sieć zdefiniowana równaniem 4.129 umożliwia użycie różnych funkcji transferu $h_i(\cdot)$. W rozważaniach poniżej i przykładowych problemach użyte będą typowe funkcje gaussowskie, funkcje sigmoidalne i funkcje okienkujące (1.73).

W celu uczenia sieci będzie stosowany algorytm gradientowy opisany w rozdziale 2.4. Wtedy sieć będzie mogła być kontrolowana przez przedstawione poniżej sposoby.

4.4.1.1. Usuwanie neuronów

W pierwszej wersji sieci OTFN użyta została metoda eliminacji wag zaproponowana przez Weigenda [253] opisana już w rozdziale 4.1.1, której głównym elementem jest dodanie regularyzacyjnego członu do funkcji błędu:

$$\lambda \sum_{i=1}^{M} \frac{w_i^2 / w_0^2}{1 + w_i^2 / w_0^2}, \tag{4.130}$$

(w_0, λ, M zdefiniowane są w rozdziale 4.1.1).

Nietrywialne zachowanie powyższego członu sprawia, iż rezultaty mogą być rzeczywiście ciekawe i godne uwagi.

4.4.1.2. Statystyczne kryterium usuwania neuronów

Można także skorzystać ze statystycznego kryterium, które określa czy usuwanie powinno nastąpić dla danego modelu $\mathcal{P}$:

$$\mathcal{P}: \ \frac{L}{Var[F(\mathbf{x};\mathbf{p}_n)]} < \chi^2_{1,\vartheta}, \tag{4.131}$$

gdzie $\chi^2_{n,\vartheta}$ jest ϑ% przedziałem ufności dla rozkładu χ^2 z jednym stopniem swobody. σ_{w_i} definiuje wariancję w_i modelu $\mathcal{P}$.

Natomiast L jest zdefiniowane przez:

$$L = \min_i \frac{w_i^2}{\sigma_{w_i}}. \tag{4.132}$$

Neuron o najmniejszej przydatności dla sieci zostanie usunięty gdy tylko przekroczy on próg zdefiniowany nierównością 4.131. Co oznacza, że albo współczynnik istotności L był za mały albo niepewność sieci $Var[F(\mathbf{x};\mathbf{p}_n)]$ była zbyt wielka.

Ten pomysł jest już znany z opisanej wcześniej sieci IncNet 4.3, jednak pozostaje problem wyznaczenia tych parametrów, które w przypadku sieci IncNet były wyznaczane za pośrednictwem filtru Kalmana.

Wariancję σ_{w_i} można naliczać iteracyjnie:

$$\sigma^n_{w_i} = \frac{N-1}{N}\sigma^{n-1}_{w_i} + \frac{1}{N}\left[\Delta {w_i}^n - \overline{\Delta {w_i}^n}\right]^2 \tag{4.133}$$

$$\overline{\Delta {w_i}^n} = \frac{N-1}{N}\overline{\Delta {w_i}^{n-1}} + \frac{1}{N}\Delta w_i^n, \tag{4.134}$$

gdzie n oznacza numer iteracji, a $\Delta w_i^n = w_i^n - w_i^{n-1}$. N definiuje długość *ogona* naliczania wariancji (może to dać bardzo ciekawe rezultaty w przypadku ciągłych danych o niestacjonarnych rozkładach).

4.4.1.3. Kryterium wystarczalności sieci

Można się tu również posłużyć ogólną ideą kryterium wystarczalności opisaną w rozdziale poświęconym sieci IncNet 4.3 (patrz też [140]).

Ogólne kryterium wystarczalności zdefiniowane było przez:

$$\mathcal{H}_0: \ \frac{e^2}{Var[F(\mathbf{x};\mathbf{p}) + \eta]} < \chi^2_{M,\theta}, \tag{4.135}$$

gdzie $\chi^2_{n,\theta}$ jest θ% przedziałem ufności dla rozkładu χ^2 z n stopniami swobody. $e = y - f(\mathbf{x};\mathbf{p})$ jest błędem jaki popełnia model w punkcie $\mathbf{x}$, a η jest wariancją danych.

Wariancja może być wyznaczana co epokę przez:

$$Var[F(\mathbf{x};\mathbf{p}_n)] = \frac{1}{N-1}\sum_i \left[\Delta F(\mathbf{x}_i;\mathbf{p}_n) - \overline{F(\mathbf{x}_j;\mathbf{p}_n)}\right]^2 \tag{4.136}$$

lub iteracyjnie, podobnie do kryterium usuwania neuronów:

$$Var[F(\mathbf{x};\mathbf{p}_n)] = \frac{N-1}{N} Var[F(\mathbf{x};\mathbf{p}_{n-1})] \; + \; \frac{1}{N}\left[\Delta F(\mathbf{x}_i;\mathbf{p}_n) - \overline{F(\mathbf{x}_j;\mathbf{p}_n)}\right]^2, \quad (4.137)$$

gdzie $\Delta F(\mathbf{x}_i;\mathbf{p}_n) = F(\mathbf{x}_i;\mathbf{p}_n) - F(\mathbf{x}_i;\mathbf{p}_{n-1})$. N również definiuje długość *ogona* zapominania.

4.4.2. Sieć optymalnych funkcji transferu typu II

Drugim sposobem uzyskania sieci optymalnych funkcji transferu jest użycie takich funkcji transferu, które mogą zmieniać swoją *naturę* w zależności od potrzeb, czyli przemieniać się (płynnie) z jednego typu funkcji w inny typ. Daje to nawet potencjalnie ciekawsze rozwiązanie od powyższego, ponieważ każdy z neuronów może niejako decydować o tym, jaki typ w danym przypadku jest bardziej korzystny.

Pierwszą z funkcji o takich możliwościach jest niewątpliwie funkcja stożkowa (1.95) opisana już w rozdziale 1.4.5. Ciekawsze rozwiązania może by uzyskać stosując rozszerzona wersję funkcji stożkowej (1.96). Tak zdefiniowana funkcja może zmieniać się od funkcji sigmoidalnej do funkcji gaussowskiej wielu zmiennych (por. rys. 1.29).

Inną, bardzo ciekawą alternatywę stanowi funkcja gaussowska uniwersalna (1.99). Funkcja ta może płynnie zmieniać się od funkcji okienkującej do funkcji gaussowskiej (por. rys. 1.31).

Spójrzmy jeszcze raz na aktywację funkcji (funkcją wyjścia jest funkcja sigmoidalna) uogólnionej stożkowej:

$$A_{GC}(\mathbf{x};\mathbf{w},\mathbf{t},\mathbf{b},\alpha,\beta) = -[\alpha I(\mathbf{x}-\mathbf{t};\mathbf{w}) + \beta D(\mathbf{x},\mathbf{t},\mathbf{b})]. \quad (4.138)$$

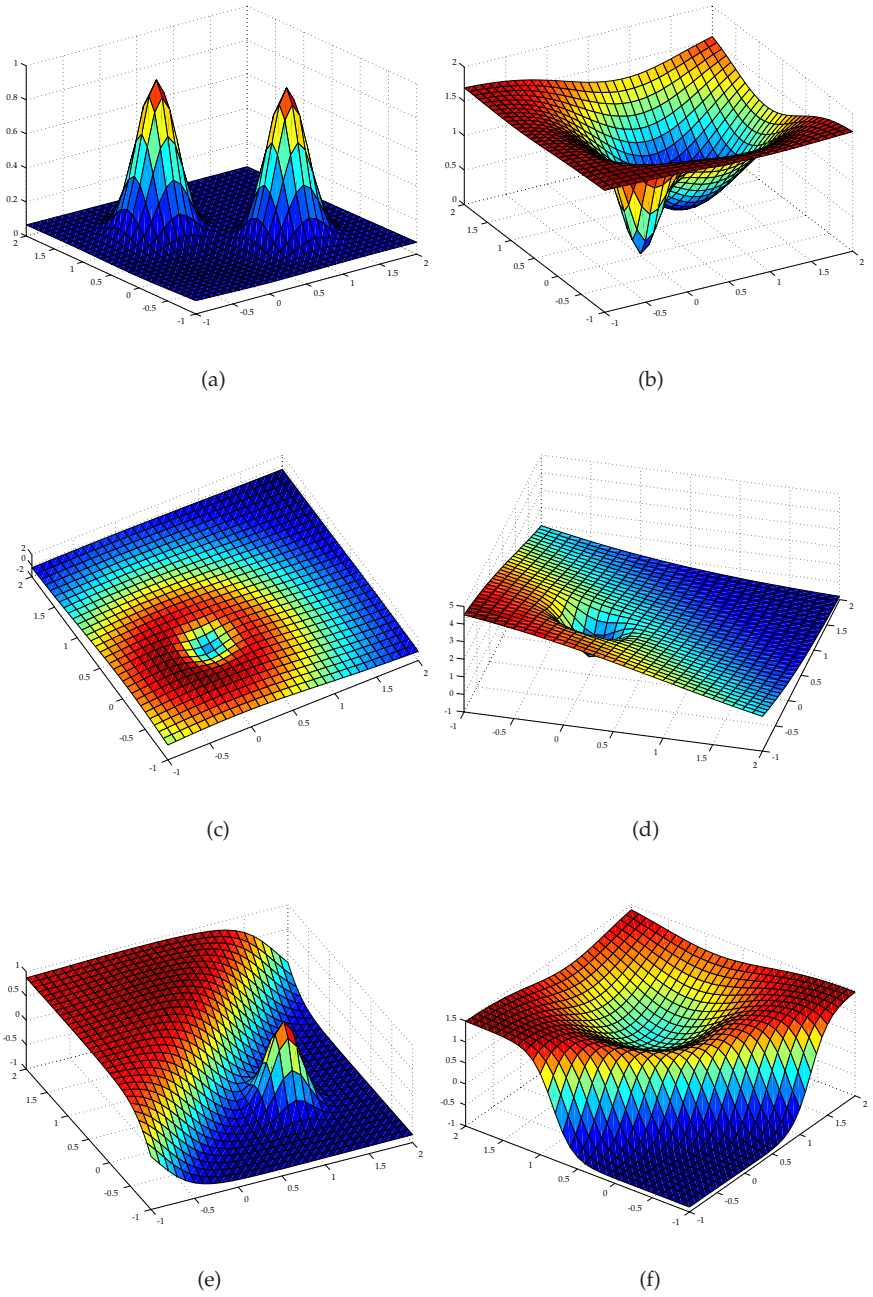
Łatwo zauważyć dwie części aktywacji, które odpowiadają za dwa różne typu podfunkcji. Pierwsza odpowiada za część sigmoidalną, druga za elipsoidalną.

Zmieniając odpowiednio funkcję błędu można sprawić, aby neuron opowiadał się za jedną z dwóch części funkcji uniwersalnej lub utrzymał obie części, ale za pewną karę. Taka funkcja błędu może wyglądać tak:

$$E_{we}(F,\mathbf{w}) = E_0(F) + \lambda \sum_{i=1}^{M} \left[\frac{\alpha_i^2/\alpha_0^2}{1+\alpha_i^2/\alpha_0^2} \cdot \frac{\beta_i^2/\beta_0^2}{1+\beta_i^2/\beta_0^2} \right]. \quad (4.139)$$

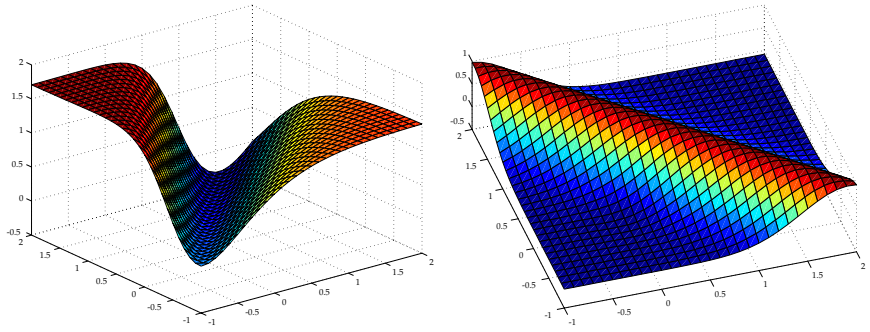
α_0 i β_0 są stałymi z wartościami zbliżonymi do 1.

Poza tym algorytm uczenia nie zmienia się względem algorytmu użytego do poprzedniej wersji sieci optymalnych funkcji transferu.



(a) (b) (c) (d) (e) (f)

Rysunek 4.13: Różnorodne rozwiązania problemu parzystości (XOR).



(a) (b)

Rysunek 4.14: Różnorodne rozwiązania problemu parzystości (XOR) cd.

4.4.3. Przykłady działania sieci optymalnych funkcji transferu

4.4.3.1. Problem parzystości

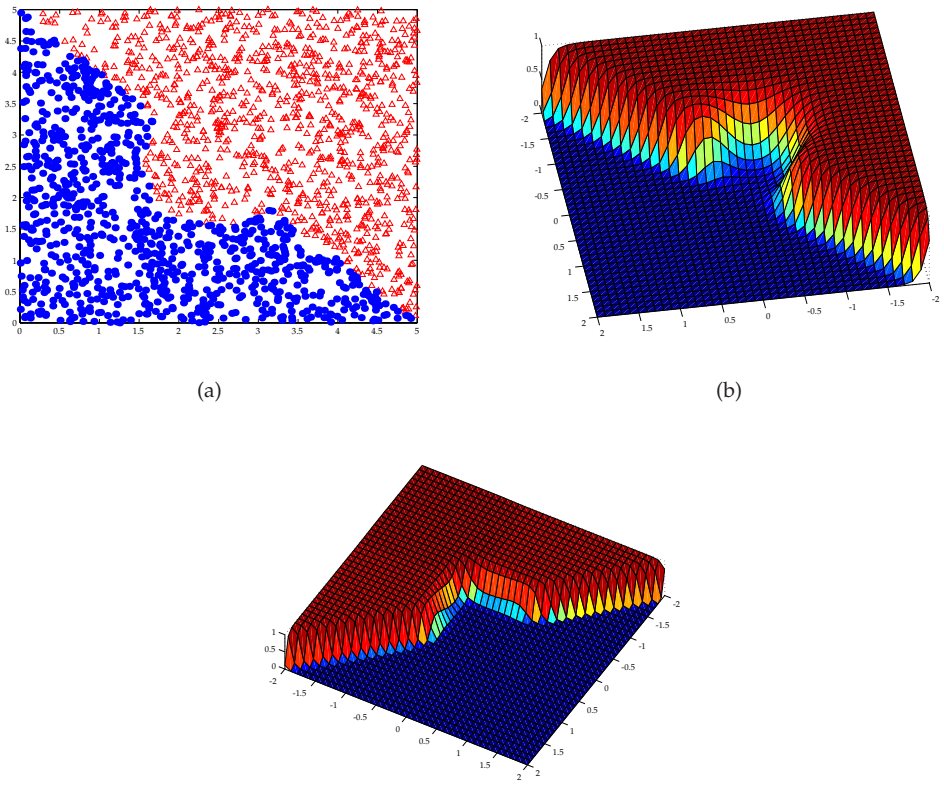
Problemy parzystości, których najprostszym przykładem jest XOR, są jednymi z bardziej eksploatowanych problemów do ilustracji działania różnych aspektów sieci neuronowych jak i innych metod inteligencji obliczeniowej.

Jak wiadomo sieć MLP dysponująca 2 neuronami sigmoidalnymi potrafi rozwiązać ten problem. Jeszcze łatwiej przychodzi to sieci RBF. Można się więc spodziewać nie tylko, że problem w przypadku sieci OFTN zostanie rozwiązany, ale i że dostarczy nam różnych rozwiązań. Tak też okazało się w rzeczywistości.

Sieć OTFN startowała z 4 neuronami: 2 neurony sigmoidalne i 2 neurony gaussowskie. Parametry neuronów były losowe z przedziału ± 0.5. Po pewnym czasie parametr λ równania 4.5 był zmniejszany tak, aby uzyskać możliwie proste rozwiązanie. Eliminacja wag była stosowana do wag pomiędzy warstwą ukrytą i wyjściową.

Proces uczenia takiej sieci neuronowej (trwający 2 000 – 10 000 iteracji) może zakończyć się bardzo różnymi poprawnymi rozwiązaniami. Niektóre z przedstawionych poniżej rozwiązań są rzadkie, inne znacznie częstsze. Rysunki 4.13 (a) do (h) ukazują różne rozwiązania. (a), (b) i (c) pokazują różne kombinacje funkcji gaussowskich, jakie zostały znalezione. Inne rozwiązania korzystają z funkcji sigmoidalnej i gaussowskiej (d), (e) i (f). Rozwiązanie, które korzysta z dwóch funkcji sigmoidalnych (g) jest bardzo rzadkie jeśli tylko sieć dysponuje funkcjami gaussowskimi one szybciej dostosowują się do funkcji błędu. (h) prezentuje rozwiązanie składające się z jednego neuronu okienkującego (1.73) – jest to najprostsze rozwiązanie tego problemu i bardzo łatwe do znalezienie siecią OTFN. Sieć bardzo szybko pozbywa się neuronów gaussowskich i sigmoidalnych zostawiając jeden neuron okienkujący.

4.4.3.2. Problem półsfery i półprzestrzeni.



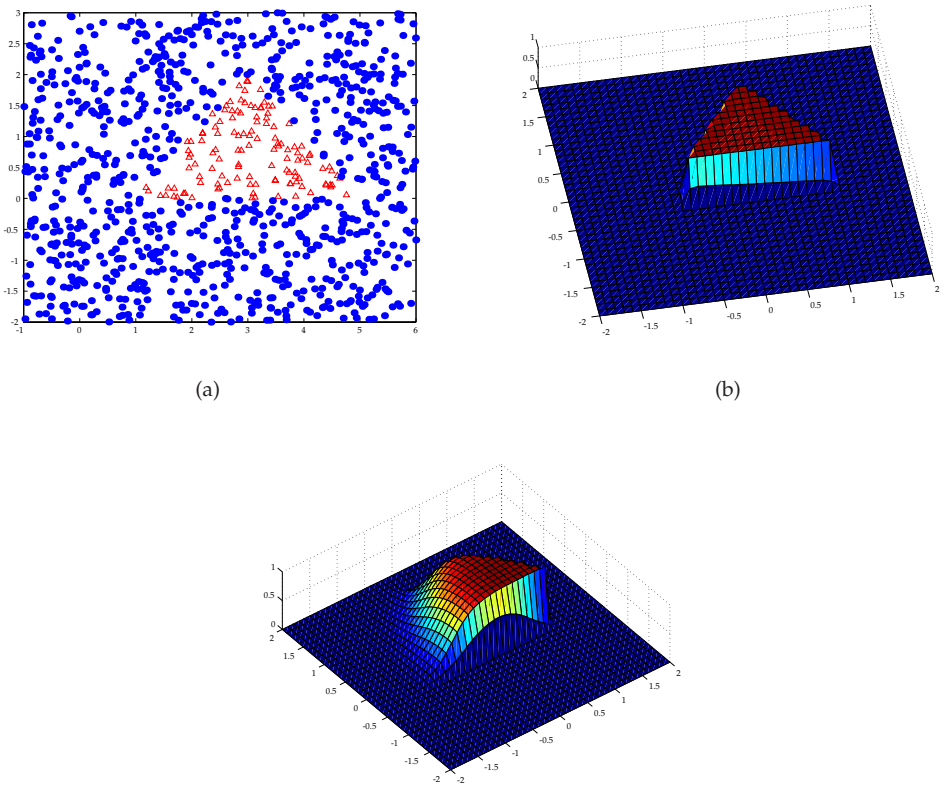
(a) (b)

(c)

Rysunek 4.15: Problem półsfera + półprzestrzeń i przykłady rozwiązań.

Problem półsfery i półprzestrzeni przedstawiony na rysunku 4.15 (a) jest nietrywialny. Idealne rozwiązanie problemu nie może być zbudowane ze stosunkowo małej liczby funkcji gaussowskich albo sigmoidalnych. Zbiór opisujący problem składa się z 2000 punktów. Na początku sieć OTFN składała się z 3 neuronów gaussowskich i 3 sigmoidalnych. Najprostsze rozwiązanie składa się z jednego węzła sigmoidalnego i jednego gaussowskiego – takie rozwiązanie zostało pokazane na rys. 4.15 (b). Choć 3 funkcje sigmoidalne dają również dobre rozwiązanie, tylko nieco gorsze – rys 4.16 (c). Liczba epok uczenia wynosiła 500 i końcowa poprawność wynosiła około 97.5–99%. Podobny test przeprowadzony został w 10-cio wymiarowej przestrzeni. Uzyskano poprawność rzędu 97.5–98%. Sieć końcowa miała 2 lub 3 neurony, zależnie od wartości współczynnika wymuszającego usuwanie neuronów.

4.4.3.3. Problem trójkąta.



(a) (b)

(c)

Rysunek 4.16: Rozwiązania problemu trójkąta.

Problem trójkąta to 1000 punktów, które składają się na dwie klasy — patrz rys. 4.16 (a). I w tym problemie na początku sieć OTFN miała po 3 neurony gaussowskie i sigmoidalne. Sieć OTFN znajdywała optymalne rozwiązanie składające się z 3 funkcji sigmoidalnych 4.16 (b) i nieco uproszczone wersje, składające się z jednego węzła gaussowskiego i 2 funkcji sigmoidalnych 4.16 (c). Znalezienie optymalnego rozwiązania jest naprawdę bardzo trudne, ponieważ funkcje gaussowskie bardzo szybko zawłaszczają spore części trójkąta, starając się rozwiązać problem. Sieć uczyła się 250 epok i poprawność końcowa wynosiła ok. 98–99%.

Rozdział 5

Komitety modeli

Komitety modeli to grupy modeli adaptacyjnych, które zostają razem użyte do rozwiązania jednego problemu. Na przestrzeni ostatnich lat powstało wiele różnych typów komitetów modeli i w poniższym rozdziale zostaną zaprezentowane te, które wydają się być najciekawsze i bardzo użyteczne.

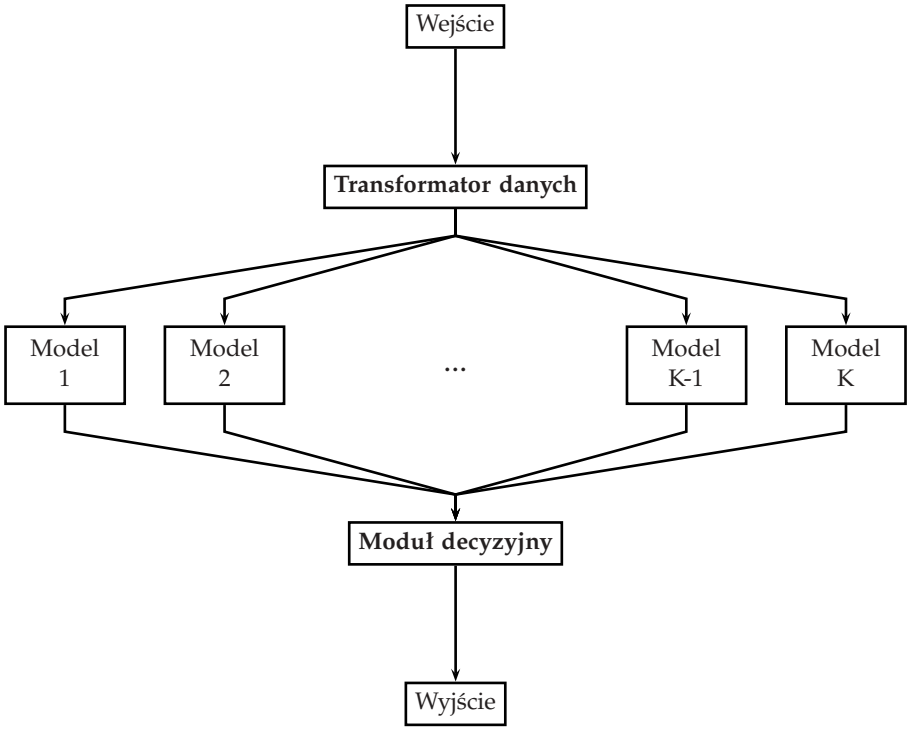
Rysunek 5.1 przedstawia ogólny schemat działania komitetu. Transformator danych przekazuje, często zmodyfikowane dane, do podmodeli. Moduł decyzyjny na podstawie odpowiedzi podmodeli dokonuje wyznaczenia ostatecznej wartości wyjściowej (klasyfikacji bądź aproksymacji).

Warto będzie zwrócić uwagę na to jak różne typy komitetów mogą i wpływają na finalną złożoność całościową modelu stworzonego do rozwiązania pewnego problemu. Będzie widać w tej kwestii duże zróżnicowanie, wynikające z odmienności natury różnych komitetów.

Dwa główne cele, jakie przyświecają stosowaniu komitetów to:

- Wspomaganie rozwiązywania problemów wieloklasowych przez rozbicie problemu na mniejsze i użycie podmodeli specjalizujących się w częściach całego problemu.
- Wyraźne polepszanie poziomu generalizacji i znaczne podniesienie poziomu stabilności modelu końcowego.

Pierwszy cel – wspomaganie problemów wieloklasowych, realizowany jest przez użycie podmodeli w komitecie. Podmodele te specjalizują się w pewnej części problemu. Jak pokażemy poniżej najczęściej konstruuje się podmodele specjalizujące się w dyskryminacji pomiędzy pewną klasą A a pozostałymi klasami, bądź tworzy się podmodele dyskryminujące pomiędzy parami klas lub podgrupami klas. Ma to ogromne znaczenie w systemach końcowych, gdzie naprawdę wymaga się, aby diagnoza była jak najbardziej wiarygodna. Właśnie w tym celu dobrze jest stosować nie tylko jeden model, który potrafi sobie radzić z problemami wieloklasowymi, ale także utworzyć podmodele (komitet),



Rysunek 5.1: Ogólny schemat komitetu.

które będą mógłby wnieść istotną dodatkową informację. W przypadku zaś, kiedy korzystamy z modeli adaptacyjnych, które nadają się tylko do problemów dwuklasowych, takie komitety stanowią jedne z lepszych sposobów obejścia problemów wieloklasowych.

Drugi cel – polepszanie poziomu generalizacji i stabilności modelu końcowego, jest pewnym fenomenem[1]. Ogólna idea polega na zastosowaniu szeregu modeli (od kilku do nawet kilkuset) w celu rozwiązania tego samego problemu. Jak łatwo się domyślić, nie ma sensu budowanie komitetu, który składałby się z identycznych modeli. To oczywiście do niczego nie prowadzi, z wyjątkiem straty czasu. Zróżnicowanie modeli uzyskiwane jest na różne sposoby (co zobaczymy w poniższych podrozdziałach), a dzięki temu powstaje szereg modeli *specjalistów*. Bywają one lepsze i gorsze, ale podejmując decyzję wspólnie, robią to trafniej (niwelacja wariancji błędu). Dzięki temu poziom generalizacji i stabilności rośnie. Dotyczy to praktycznie wszystkich modeli inteligencji obliczeniowej, nie tylko jakiejś ich podgrupy. Niewątpliwie można dopatrzyć się w takiej kooperacji pewnych elementów regularyzacji, niejako przez rozmycie

[1]Napisałem *fenomenem* ponieważ nierzadko w komitecie znajdują się podmodele o wątpliwej jakości generalizacji, ale *całość* komitetu radzi sobie bardzo dobrze.

pojedynczych decyzji.

Warto tu dodać, że bardzo pomocne może być stosowanie komitetów modeli ontogenicznych. Szczególnie gdy stosuje się zaostrzone kryteria co do złożoności architektury sieci. Wtedy pojedyncze modele ontogeniczne mogą nie dysponować oczekiwaną dokładnością, ale mogą doskonale nadawać się na podmodele komitetu.

5.1. *K*-klasyfikatorów

Gdy dany problem jest problemem wieloklasowym, a nasz model nie umie radzić sobie z więcej niż dwiema klasami, możemy zbudować komitet składający się *K* klasyfikatorów. Dalej taki komitet nazywany będzie *K*-klasyfikatorem. Każdy z podmodeli będzie specjalizował się w rozpoznawaniu jednej z klas, i będzie dokonywał dyskryminacji pomiędzy pewną *i*-tą klasą, a resztą klas. Takie rozwiązanie warte jest rozważenia nie tylko gdy dany model adaptacyjny nie umie radzić sobie z więcej niż dwiema klasami. Może okazać się, że dyskryminacja jednej klasy da proste, ciekawe rozwiązania (o prostej interpretacji), podczas gdy rozróżnienie naraz wszystkich klas może być znacznie bardziej skomplikowane. Nierzadko bywa, że złożoność modeli dyskryminujących poszczególne klasy będzie znacznie różna. Da nam to dodatkową wiedzę o problemie. Może też się okazać, że suma złożoności modeli dyskryminujących pojedyncze klasy będzie mniejsza, niż jednego modelu wieloklasowego.

W celu zbudowania *K*-klasyfikatora oryginalny zbiór par uczących:

$$\mathcal{S} = \{\langle \mathbf{x}_1, y_1\rangle, \langle \mathbf{x}_2, y_2\rangle, \ldots \langle \mathbf{x}_N, y_N\rangle\}, \tag{5.1}$$

zamieniamy na *K* zbiorów, po jednym dla każdej klasy:

$$\mathcal{S}^k = \{\langle \mathbf{x}_1, y_1^k\rangle, \langle \mathbf{x}_2, y_2^k\rangle, \ldots \langle \mathbf{x}_N, y_N^k\rangle\} \quad k = 1, 2, \ldots, K, \tag{5.2}$$

gdzie y_i^k przyjmuje wartości 1 lub 0:

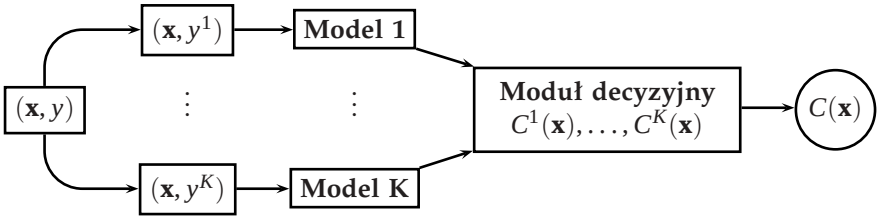
$$y_i^k = \begin{cases} 1 & y_i = k \\ 0 & y_i \neq k \end{cases} \quad i = 1, 2, \ldots, N. \tag{5.3}$$

Tak zdefiniowane zbiory $\mathcal{S}^k$ są zbiorami uczącymi dla poszczególnych podmodeli komitetu. Schemat *K*-klasyfikatora przedstawiono na rysunku 5.2.

Dla danej obserwacji $\mathbf{x}$ każdy z podmodeli generuje wartość wyjściową $C^i(\mathbf{x})$ (dla $i = 1, 2, \ldots, K$). Wartość $C^i(\mathbf{x})$ powinna być zawarta w przedziale $[0, 1]$. Wartość zbliżona do zera oznacza, że obserwacja $\mathbf{x}$ nie odpowiada klasie i, natomiast wartość $C^i(\mathbf{x})$ zbliżona do jedynki oznacza, że obserwacja $\mathbf{x}$ odpowiada klasie i.

Moduł decyzyjny, widoczny na rysunku 5.2, podejmuje ostateczną decyzję i przypisuje pewien wektor $\mathbf{x}$ do jednej klasy $C(\mathbf{x})$. Podejmowanie takiej decyzji może przebiegać zgodnie z poniższą definicją:

$$C(\mathbf{x}) = \arg\max_i \; C^i(\mathbf{x}). \tag{5.4}$$



Rysunek 5.2: Schemat komitetu K-klasyfikatora.

Tym samym moduł decyzyjny wybiera sieć, której aktywacja była największa, tj. najbardziej odpowiadająca danej obserwacji.

Normalizując wartości $C^i(\mathbf{x})$ możemy dostać prawdopodobieństwa przynależności do poszczególnych klas:

$$p(C^i|\mathbf{x}) = \frac{C^i(\mathbf{x})}{\sum_{j=1}^{K} C^j(\mathbf{x})}. \tag{5.5}$$

Przedstawiony komitet jest w 98% zgodny z tym, który został użyty dla sieci IncNet, opisanej w rozdziale 4.3.7. Różnica polega na wartościach wyjść modeli $C^i(\mathbf{x})$. Dla powyższego komitetu znajdują się one w przedziale $[0, 1]$. Natomiast dla sieci IncNet wyjście może przyjąć wartości nieco poniżej 0 lub nieco powyżej 1 (typowe wartości oczywiście znajdą się wewnątrz przedziału $[0, 1]$. Dlatego też inaczej trzeba postąpić w przypadku wyznaczania prawdopodobieństw. Proszę porównać wzory 4.126 i 5.5.

5.2. K^2-klasyfikatorów

Innym podejściem jest stworzenie komitetu, w którym znajdą się modele dyskryminujące dla każdej pary klas k_i, k_j. Wymusza to stworzenie $\binom{K}{2}$ modeli C_{ij} dla $i = 1, \ldots K, j = 1, \ldots, K, i \neq j$ i tyle samo zbiorów par uczących:

$$\mathcal{S}^{ij} = \{\langle \mathbf{x}_k, y_k^{ij} \rangle \ : \ y_k = i \ \vee \ y_k = j \rangle\}, \tag{5.6}$$

gdzie y_k^{ij} przyjmuje wartości 1 lub 0:

$$y_k^{ij} = \begin{cases} 1 & y_k = i, \\ 0 & y_k \neq i. \end{cases} \tag{5.7}$$

Proszę zauważyć, że dzięki definicji 5.6 zbiory $\mathcal{S}^{ij}$ składają się tylko z tych wektorów, które należą do klasy i lub j. Dzięki temu każdy model C_{ij} posiada po pozytywnym etapie uczenia wiedzę o rozróżnianiu klasy i od klasy j. Wyjście modelu C_{ij}, $C_{ij}(\mathbf{x})$, powinno dawać wartość z przedziału $[0, 1]$. Wartość bliska

zeru odpowiada wektorom **x** klasy j, a bliska 1 wektorom **x** klasy i. Gdy przyjąć: $C_{ij}(\mathbf{x}) = 1 - C_{ji}(\mathbf{x})$, wtedy wystarcza utworzyć tylko modele C_{ij} gdzie $i < j$.

Tym samym mamy macierz klasyfikatorów (i ich odpowiedzi dla wektorów wejściowych **x**:

–	$C^{1,2}$	$C^{1,3}$	$\dots$	$C^{1,k-1}$	$C^{1,k}$
$C^{2,1}$	–	$C^{2,3}$		$C^{2,k-1}$	$C^{2,k}$
$C^{3,1}$	$C^{3,2}$	–		$C^{3,k-1}$	$C^{3,k}$
$\vdots$			$\ddots$		$\vdots$
$C^{k-1,1}$	$C^{k-1,2}$	$C^{k-1,3}$		–	$C^{k-1,k}$
$C^{k,1}$	$C^{k,2}$	$C^{k,3}$	$\dots$	$C^{k,k-1}$	–

. (5.8)

Niestety, gdy zapytamy model C_{ij} o klasyfikacje wektora, który ewidentnie należy do klasy różnej od i i j, trudno spodziewać się racjonalnej odpowiedzi (model C_{ij} nigdy nie widział wektorów należących do tej klasy!). Jest to bardzo podobne do zapytania pewnej osoby co sądzi o jazzie podczas, gdy dana osoba nigdy w życiu nie słyszała ani jednego utworu jazzowego. Jak się domyślamy prowadzi to do sporego zagrożenia w wyciąganiu dalej idących wniosków, w oparciu jedynie o modele C_{ij}.

Przynależność danego wektora **x** do klasy i można zdefiniować przez poniższe równanie

$$p(C^i|\mathbf{x}) = \frac{\sum_{j,i\neq j} C^{ij}(\mathbf{x})}{\sum_{k,j,k\neq j} C^{kj}(\mathbf{x})}, \tag{5.9}$$

jednak jest to **niebezpieczne** z powodów wspomnianych powyżej. Bardziej zalecane jest dokonanie wstępnej diagnozy w oparciu o komitet K-klasyfikatorów. Następnie, gdy nie mamy pełnej jednoznaczności, tj. istnieją dwie (czasem więcej) klasy i i j, dla których wartości $C^i(\mathbf{x})$ i $C^j(\mathbf{x})$ (patrz równanie 5.4) są duże, można posiłkować się klasyfikatorem C^{ij}, który specjalizuje się w rozstrzyganiu przynależności do jednej z klas i i j. Takie postępowanie jest już poprawne i zalecane. Z pewnością pozytywnie powinno wpłynąć na wiarygodność ostatecznej decyzji.

Komitet K^2-klasyfikatorów może okazać się efektywniejszy od k-klasyfikatorów, gdy podmodelami komitetów będą liniowe modele dyskryminujące. W takich przypadkach może okazać się niemożliwe oddzielenie pojedynczą hiperpłaszczyzną wektorów pewnej klasy od pozostałych wektorów, natomiast całkiem realne będzie (może być) oddzielenie wszystkich par klas pomiędzy sobą.

5.3. Maszyna liniowa

W przypadku liniowej dyskryminacji z problemami wieloklasowymi radzono sobie wykorzystując *maszynę liniową*. Metoda polega na konstrukcji funkcji dys-

kryminujących dla każdej z klas:

$$g_i(\mathbf{x}) = \mathbf{w}_i^\top \mathbf{x} + w_{i0} \quad i = 1, \ldots, K. \tag{5.10}$$

Wtedy przynależność wektora $\mathbf{x}$ do jednej (lub kilku równoważnych) z klas wyznacza się poprzez:

$$C(\mathbf{x}) = \arg \max_{i=1,\ldots,K} g_i(\mathbf{x}). \tag{5.11}$$

Powyższe równanie można odczytywać jako poszukiwanie *najsilniejszej* dyskryminacji dla danego wektora $\mathbf{x}$. Nie trudno dopatrzyć się sporego podobieństwa pomiędzy K-klasyfikatorem, a maszyną liniową. Wystarczy porównać powyższe równanie z równaniem 5.4.

Warto zwrócić uwagę, że regiony przynależności poszczególnych punktów przestrzeni do klas nie muszą być ciągłe. W przypadku ciągłości dla pewnych dwóch regionów i i j, ich podział definiuje hiperpłaszczyzna

$$g_i(\mathbf{x}) = g_j(\mathbf{x}), \tag{5.12}$$

czyli

$$(\mathbf{w}_i - \mathbf{w}_j)^\top \mathbf{x} + (w_{i0} - w_{j0}) = 0. \tag{5.13}$$

Co daje w łatwy sposób wyznaczyć odległość punktu $\mathbf{x}$ do hiperpłaszczyzny zdefiniowanej powyższym wzorem: $(g_i - g_j) / ||\mathbf{w}_i - \mathbf{w}_j||$.

5.4. Sposoby podejmowania decyzji przez komitet: głosowanie i ważenie

Powyższe komitety składały się z modeli, które specjalizowały się w rozwiązywaniu części problemu, dyskryminacji pomiędzy dwiema klasami, bądź dyskryminacji pomiędzy pewną klasą i pozostałą częścią zbioru. Poniżej omówione zostaną komitety, których modele wspólnie rozwiązują ten sam problem. Wiele publikacji wskazuje jednoznacznie, że stosowanie komitetów z ważeniem czy głosowaniem ewidentnie podnosi poziom generalizacji [89, 245, 21, 55, 178, 214, 8].

Komitety mogą być wykorzystywane równie dobrze do problemów aproksymacyjnych, jak i klasyfikacyjnych. Ogólna postać komitetu do **problemów aproksymacyjnych** może mieć postać:

$$\bar{F}(\mathbf{x}) = \sum_{i=1}^{T} w_i F_i(\mathbf{x}), \tag{5.14}$$

jak widać, efektem komitetu jest po prostu **ważona** kombinacja podmodeli. Najczęściej suma wag w_i powinna być równa 1, a poszczególne $w_i >= 0$ (w najprostszym przypadku $w_i = 1/T$).

W przypadku klasyfikacji można przeprowadzić **głosowanie** w oparciu o klasyfikacje poszczególnych modeli komitetu:

$$\bar{F}(\mathbf{x}) = \arg\max_i \sum_{j:F_i(\mathbf{x})=i} 1. \tag{5.15}$$

Można także użyć **ważenia głosowania**:

$$\bar{F}(\mathbf{x}) = \arg\max_i \sum_{j:F_j(\mathbf{x})=i} w_i. \tag{5.16}$$

Sposoby wyznaczania wag w_i zostaną zaprezentowane w kolejnych podrozdziałach.

Zakładając natomiast, że każdy model daje nie tylko etykietę zwycięskiej klasy, ale także wektor prawdopodobieństw poszczególnych klas:

$$\mathbf{p}^i = F_i(\mathbf{x}), \tag{5.17}$$

gdzie $\mathbf{p}^i = [p_1^i, \ldots, p_K^i]$, a K jest liczbą klas, możemy dokonać **ważenia prawdopodobieństw decyzji**:

$$\bar{\mathbf{p}} = \frac{\sum_{i=1}^T w_i \mathbf{p}^i}{\sum w_i} \tag{5.18}$$

i następnie wybrać najbardziej prawdopodobną klasę:

$$\arg\max_i \bar{p}_i. \tag{5.19}$$

Wszystkie powyższe sposoby sprawiają, że docelowo można uzyskać lepszy poziom generalizacji. Wariancja oczekiwanego błędu zmniejsza się aż T razy [245].

5.5. Bootstrap Aggregating (Bagging)

Jak napisał Breiman w [24], gdyby *przyjacielski duszek* obdarzył nas niezależnymi kopiami $\mathcal{S}_1, \mathcal{S}_2, \ldots$ zbioru par $\mathcal{S} = \{\langle \mathbf{x}_i, y_i \rangle : 1 \leq i \leq N\}$ i każdy zbiór $\mathcal{S}_i$ składałby się z N wylosowanych próbek z tym samym, nieznanym prawdopodobieństwem, można by skonstruować modele predykcyjne $F(\mathbf{x}, \mathcal{S}_i)$, a następnie utworzyć model finalny:

$$\bar{F}(\mathbf{x}) = \frac{1}{T} \sum_{i=1}^T F(\mathbf{x}, \mathcal{S}_i). \tag{5.20}$$

Wtedy model $\bar{F}(\mathbf{x})$ będzie miał mniejszy błąd predykcji. W przypadku klasyfikacji trzeba uśrednić głosowanie podmodeli klasyfikujących:

$$\bar{F}(\mathbf{x}) = \arg\max_i \sum_{j:F(\mathbf{x},\mathcal{S}_j)=i} 1. \tag{5.21}$$

Niestety zbiorów $\mathcal{S}_i$ zdefiniowanych powyżej stworzyć nie można. Można jednak stworzyć ich przybliżenie. Nowe zbiory $\mathcal{S}_i^B$ można utworzyć, losując niezależnie N razy próbki ze zbioru $\mathcal{S}$ (z możliwością powtarzania). Breiman próbki $\mathcal{S}_i^B$ nazywa próbkami rozkładu *bootstrap*. Każdemu punktowi ze zbioru $\mathcal{S}$ przydziela się prawdopodobieństwo $1/N$.

Można wtedy stworzyć nowy model przez agregację (uśrednienie):

$$\bar{F}(\mathbf{x}) = \frac{1}{T}\sum_{i=1}^{T} F(\mathbf{x}, \mathcal{S}_i^B) \tag{5.22}$$

dla regresji. Natomiast dla klasyfikacji należy przeprowadzić głosowanie:

$$\bar{F}(\mathbf{x}) = \arg\max_i \sum_{j\,:\,F(\mathbf{x},\mathcal{S}_j^B)=i} 1. \tag{5.23}$$

Dlatego też tytułowa nazwa *bagging* pochodzi od bootstrap aggregating.

5.6. Boosting i AdaBoost

Boosting jest komitetem, którego końcowy model charakteryzuje się dużym marginesem ufności (porównaj rozdział 3). Przykłada on szczególną wagę do wektorów, które mają mały, bądź ujemny margines ufności, czyli są umiejscowione na granicach decyzyjnych lub nawet po błędnych stronach granic decyzyjnych [89, 90, 224].

Zasadnicza różnica pomiędzy Baggingiem i Boostingiem polega na tym, że Bagging uczy wszystkie podmodele niezależnie, natomiast boosting używa sekwencyjnie bazowego algorytmu uczenia (wybranego modelu adaptacyjnego, z którego będzie budowany komitet), za każdym razem zmieniając jednak rozkład danych treningowych. Załóżmy, że algorytm będzie składał się docelowo z T modeli. Wtedy w krokach $t = 1, \ldots, T$ rozkład $D_t(\cdot)$ będzie pewną funkcją przydzielającą prawdopodobieństwa z rozkładem zależnym od t (każdy model komitetu ma próbki losowane zgodnie z $D_T(\cdot)$) zbioru uczącego $\mathcal{S}$, który składa się z par $\mathcal{S} = \{\langle \mathbf{x}_i, y_i\rangle : 1 \le i \le N\}$.

Adaptive Boosting (AdaBoost) jest jedną z realizacji koncepcji boostingu. Celem AdaBoost jest konstruowanie kolejnych modeli h_t tak, aby w miarę możliwości malał błąd ϵ_t:

$$\epsilon_t = \sum_{i\,:\,y_i \neq h_t(\mathbf{x}_i)} D_t(i). \tag{5.24}$$

Początkowo (D_1) ma rozkład jednostajny: $D_1(i) = 1/N$. Następnie w każdej kolejnej iteracji będzie zmieniany:

$$D_{t+1}(i) = \frac{D_t(i)\exp(-y_i\alpha_t h_t(x_i))}{Z_t}, \tag{5.25}$$

gdzie $\alpha_t = \frac{1}{2}\ln((1-\epsilon_t)/\epsilon_t)$. Zakłada się, że $y_i \in \{-1,1\}$ i $h_t(\mathbf{x}_i) \in \{-1,1\}$ (czyli rozważany jest przypadek dwuklasowy, dla problemów wieloklasowych można użyć zmodyfikowanej wersji opisanej w dalszej części). Z_t jest czynnikiem normalizującym D_{t+1} tak, aby D_{t+1} było prawdopodobieństwem. Może to być:

$$Z_t = \sum_{i=1}^{N} D_t(i)\exp(-y_i\alpha_t h_t(\mathbf{x}_i)) \tag{5.26}$$

$$= \sum_{i:y_i=h_t(\mathbf{x}_i)} D_t(i)e^{-\alpha_t} + \sum_{i:y_i\neq h_t(\mathbf{x}_i)} D_t(i)e^{\alpha_t} \tag{5.27}$$

$$= 2\sqrt{\epsilon_t(1-\epsilon_t)}. \tag{5.28}$$

Finalny komitet jest wtedy ważonym większościowym klasyfikatorem:

$$F(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T}\alpha_t h_t(\mathbf{x}_t)\right). \tag{5.29}$$

Dzięki powyżej przedstawionej ewolucji rozkładu D_t AdaBoost nie tylko zmniejsza wariancję modelu, ale zmniejsza błędy w regionach o małym marginesie zaufania. Prowadzi to do jeszcze lepszych wyników, niż wyniki uzyskane przy stosowaniu Baggingu [89, 90, 224].

Zaskakująco ciekawy rezultat uzyskano dla komitetów jednowęzłowych drzew decyzyjnych (*ang. decision „stump"*, drzewo o jednym możliwie najlepszym binarnym węźle podziału). Użycie AdaBoost dawało nieporównywalnie lepsze rezultaty (dzięki adaptacji D_t). Na zbiorach *letter* (z repozytorium UCI [182]) AdaBoost uzyskał 20% błędu, a Bagging 100% (oba komitety budowane były z drzew jednowęzłowych), dla zbioru *satimage* AdaBoost dał 18%, a Bagging 40%, i dla zbioru *vehicle* AdaBoost dał 20%, a Bagging 60%. Należy jednak zwrócić uwagę, że drzewa decyzyjne z jednym węzłem są bardzo słabym klasyfikatorem i informacja, jaką dają jest bardzo uboga. Choć, jak widać, nawet w takich warunkach AdaBoost daje sobie radę – końcowy komitet składał się z aż 1000 podmodeli.

Z drugiej strony rezultaty uzyskane przez Freund and Schapire [89][213, 214] dla AdaBoostingu i Baggingu z użyciem C4.5 [213, 214] jako modelu bazowego używały, dla kilku baz danych z repozytorium UCI [182] także pokazują wyższość AdaBoostingu (tabela 5.1).

Jeśli chcielibyśmy użyć AdaBoost do problemów **wieloklasowych**, to należałoby skorzystać z wersji AdaBoost.M2, opisanej w [89]. W tej wersji również sekwencyjnie buduje się modele h_t z tą różnicą, że $h_t(\mathbf{x}_i, y_i)$ zwraca wartość 1 lub 0 w zależności czy h_t poprawnie przewidział klasę y_i (1 – poprawnie, 0 – nie), a $h_t(\mathbf{x}_i, y)$ informuje czy h_t przewidział jakąś niepoprawną klasę dla $\mathbf{x}_1$ (1 – przewidział niepoprawną klasę, 0 – nie przewidział żadnej niepoprawnej klasy dla $\mathbf{x}_i$). Pamiętajmy, że h_t może wskazać nie więcej niż jedną klasę.

Błąd (%)	AdaBoost+C4.5	Bagging+C4.5	C4.5
Letter	3.3	6.85	6.8
Satimage	8.9	10.6	14.8
Vehicle	22.6	26.1	29.9
Iris	5	5	5.9
Breast C. Wisconsin	3.3	3.2	5
Promoters	5	12.7	22
Glass	22.7	25.7	31.7

Tabela 5.1: Porównanie rezultatów dla kilku baz danych z UCI repository [182] przy użyciu algorytmu C4.5 i AdaBoost z C4.5 i Bagging z C4.5 [89].

Wtedy trzeba przedefinować ϵ_t:

$$\epsilon_t = \frac{1}{2} \sum_{(i,y)\in\mathcal{S}} D_t(i,y)[1 - h_t(\mathbf{x}_i, y_i) + h_t(\mathbf{x}_i, y)], \tag{5.30}$$

jak również D_t:

$$D_{t+1}(i,y) = \frac{D_t(i,y)}{Z_t} \cdot \beta_t^{\frac{1}{2}[1-h_t(\mathbf{x}_i,y_i)+h_t(\mathbf{x}_i,y)]}, \tag{5.31}$$

gdzie $\beta_t = \epsilon_t(1-\epsilon_t)$, Z_t jak i poprzednio pełni rolę czynnika normalizującego, tak aby D_t było rozkładem.

Natomiast końcowy model przyjmuje postać:

$$F(\mathbf{x}) = \arg\max_y \sum_{t=1}^{T} \left(\log \frac{1}{\beta_t}\right) h_t(\mathbf{x}, y). \tag{5.32}$$

Innym sposobem użycia AdaBoostingu w przypadku problemu wieloklasowego jest skorzystanie z *k*-Klasyfikatora (opisanego powyżej).

Jeszcze inne zmiany AdaBoostingu (gradient boosting, stochastic gradient boosting) zaproponował Friedman w [93, 93].

W celu głębszego prześledzenia własności boostingu (i modeli pokrewnych) należy polecić [181].

5.7. Inne komitety: Arcing, RegionBoost, Stacking, Grading, Mixture of experts

Powyższe rozdziały o komitetach bynajmniej nie wyczerpują tematyki konstrukcji komitetów. Z drugiej strony, najistotniejsze informacje zostały już przedstawione. Trudno jednak nie wspomnieć o kilku pomysłach.

5.7.1. Arcing

Breiman zaproponował w [20, 23, 8] metodę **Arcing**. Komitet jest bardzo zbliżony do AdaBoost, z tym, że inaczej wyznacza się rozkład D_t:

$$D_{t+1}(i) = \frac{1}{Z} \cdot \left(1 + \sum_{1 \le j \le t \,:\, h_j(\mathbf{x}_i) \neq y_i} 1 \right). \tag{5.33}$$

Zadaniem Z jest, jak zawsze, sprawić aby D_t było rozkładem prawdopodobieństwa.

Końcowy komitet to już standardowe głosowanie:

$$\bar{F}(\mathbf{x}) = \arg\max_i \sum_{j \,:\, h(\mathbf{x})=i} 1. \tag{5.34}$$

5.7.2. RegionBoost

Inna metodą, uwzględniającą lokalny charakter błędów, jest **RegionBoost** [177]. Działa ona niemal tak samo, jak opisany komitet AdaBoost z tym, że lokalny (ważony) wpływ każdego z podmodeli jest ważony na podstawie poprawności, jaką uzyskują podmodele w klasyfikacji k najbliższych sąsiadów na odpowiednich podmodelach.

Wagi oparte o analizę poprawności najbliższych sąsiadów mogą być zdefiniowane jak poniżej:

$$w_i(\mathbf{x}_i) = \frac{1}{T} \sum_{j=1}^{T} kNN(K, F_j, \mathbf{x}_i, y_i), \tag{5.35}$$

$$kNN(K, F_j, \mathbf{x}_i, y_i) = \frac{1}{K} \left[\sum_{s \in \mathcal{NS}(K, \mathbf{x}_i) \;\wedge\; F_j(\mathbf{x}_s) = y_s} 1 \right], \tag{5.36}$$

gdzie $\mathcal{NS}(K, \mathbf{x})$ jest zbiorem K najbliższych sąsiadów wektora $\mathbf{x}$, T oznacza liczbę modeli komitetu.

5.7.3. Stacking

Kolejną znaną i ciekawą metodą wyznaczania wag w_i wpływu poszczególnych, już nauczonych podmodeli komitetu jest **Stacking** [264, 243, 244, 22]. W tej metodzie miejsce modułu decyzyjnego zajmuje model adaptacyjny, przez co uczenie w stackingu przebiega dwuetapowo. Pierwszy etap to nauczenie podmodeli komitetu. Drugi to uczenie modelu z modułu decyzyjnego podejmowania decyzji na podstawie decyzji podejmowanych przez podmodele komitetu.

Omówiona zostanie tu wersja MLR stackingu [243] ze względu na jej skuteczność. Jednak zanim przejdziemy do sedna, czyli drugiego etapu uczenia,

należy jeszcze wskazać różnice pierwszego etapu uczenia wspólnego wszystkim komitetom. W przeciwieństwie do różnych metod opartych o boosting, w stackingu podmodele komitetu mają przygotowywane dane do uczenia i testu, tak jak to się robi dla walidacji skośnej. Czyli cały zbiór danych $\mathcal{S}$ jest dzielony na T równych części $\mathcal{S}_i$. Następnie każdy podmodel i uczony będzie na podstawie zbioru $\mathcal{S} - \mathcal{S}_i$, a testowany będzie na części testowej $\mathcal{S}_i$. W ten sposób uzyskujemy T modeli, z których żaden nie był uczony na wszystkich wektorach pierwotnego zbioru uczącego, lecz każdy z wektorów brał udział w uczeniu $T-1$ podmodeli.

Aby przejść do uczenia na drugim etapie musimy poznać jak konstruowane są dane do uczenia modułu adaptacyjnego, który już będzie spełniał funkcję modułu decyzyjnego komitetu. Tworzony jest nowy zbiór uczący $\mathcal{S}^{CV}$:

$$\mathcal{S}^{CV} = \{\langle \mathbf{z}_i, y_i \rangle, i = 1, \ldots, N\}, \tag{5.37}$$

gdzie $\mathbf{z}_i$ jest wektorem:

$$[z_{1i}, z_{2i}, \ldots, z_{Ti}] = [F_1(\mathbf{x}_i), F_2(\mathbf{x}_i), \ldots, F_T(\mathbf{x}_i)]. \tag{5.38}$$

Jak teraz widać moduł decyzyjny będzie uczony w oparciu o wyjścia podmodeli komitetu.

W przypadku Stackingu-MLR za metodę uczenia modułu decyzyjnego przyjęto metodę *multi-response linear regression*, czyli wielwymiarową metodę regresji. W przypadku problemów klasyfikacji K klasowej tworzy się K niezależnych modeli regresji liniowej. Każda regresja liniowa jest uczona rozpoznawania obiektów swojej klasy i dyskryminowania obiektów obcych klas (tak jak ma to miejsce w k-klasowym komitecie). Liniowa regresja dla i-tej klasy została zdefiniowana przez:

$$LR_i(\mathbf{x}) = \sum_{t}^{T} \sum_{k}^{K} \alpha_{tki} P_{tk}(\mathbf{x}), \tag{5.39}$$

gdzie $P_{tk}(\mathbf{x})$ to prawdopodobieństwo przynależności wektora $\mathbf{x}$ do klasy k uzyskane przez podmodel t.

Współczynniki α_{tki} wyznaczone zostają w procesie minimalizacji funkcji błędu:

$$\sum_{j} \sum_{\langle \mathbf{x}_j, y_j \rangle \in \mathbf{S}_j} (y_n - LR_j(\mathbf{x}))^2. \tag{5.40}$$

Po procesie uczenia klasą zwycięską zostaje klasa, dla której $LR_i(\mathbf{x})$ było największe.

Po uśrednieniu rezultatów [273] dla 20 różnych zbiorów danych z repozytorium UCI [182] najlepszy średni rezultat dał właśnie stacking-MLR (por. z tabelą 5.2).

Model	Średni błąd (%)
C 4.5	14.57
IBk	12.52
Naive Bayes	15.53
Bagging+C4.5	12.49
Boosting+C4.5	12.97
Voting	11.85
Grading	11.90
Multi-scheme	11.22
Stacking-MLR	10.92
Stacking-MDT	11.17

Tabela 5.2: Porównanie efektywności stackingu do innych modeli komitetowych (wyniki zaczerpnięte z pracy [273]). Średnie błędy zostały uzyskane z wyników dla 20 różnych baz danych z UCI [182]. Grading jest opisany poniżej (5.7.4). Multi-scheme jest prostym mechanizmem selekcji modelu przez walidację skośną, wektory są klasyfikowane przez model bazowy, który uzyskał najmniejszy błąd na zbiorze treningowym.

5.7.4. Grading

Komitety grading zaproponowane w [231] dla każdego modelu bazowego budują jeden meta-model, którego zadaniem będzie przewidzieć, kiedy dany model bazowy popełnia błędy. Tym samym etykiety–klasy zbioru uczącego meta-model zawierają informacje o poprawności lub niepoprawności decyzji modelu bazowego. Ostateczna decyzja to głosowanie, ale z ewentualnymi korekcjami uzyskanymi z meta-modeli.

5.7.5. Mixture of local experts

Ciekawy komitet *kombinacja lokalnych ekspertów (ang.* mixture of local experts*)* zaproponowali Jacobs i in. w [131]. Komitet ten waży wpływ poszczególnych podmodeli. Wagi jednak nie są skalarne, lecz przyjmują postać funkcyjną:

$$\bar{F}(\mathbf{x}) = \sum_{i=1}^{T} g_i(\mathbf{x}) F_i(\mathbf{x}). \tag{5.41}$$

Porównaj ze wzorem 5.14. Funkcje $F_i(\mathbf{x})$ spełniają rolę ekspertów (sieci neuronowe, ale i inne modele adaptacyjne), natomiast $g_i(\mathbf{x})$ są funkcjami dodatnimi ($g_i(\mathbf{x}) > 0$) i ich suma po modelach daje 1:

$$\sum_{i=1}^{T} g_i(\mathbf{x}) = 1. \tag{5.42}$$

Do uczenia użyto algorytmu ME (*ang. maximum likelihood*). Zakłada on dodanie gaussowskiego szumu do wyjścia sieci $F_i(\mathbf{x})$ z wariancją $\sigma_i(\mathbf{x})^2$. Funkcja $g_i(\mathbf{x})$ określa prawdopodobieństwo wyboru eksperta i. Wtedy prawdopodobieństwo wyjścia jest mieszanką prawdopodobieństw:

$$P(y|\mathbf{x}) = \sum_{i=1}^{T} g_i(\mathbf{x}) G\left(y; F_i(\mathbf{x}), \sigma_i(\mathbf{x})^2\right), \tag{5.43}$$

$G(\cdot)$ jest normalnym rozkładem gaussowskim z centrum w $F_i(\mathbf{x})$ i wariancją $\sigma_i(\mathbf{x})^2$.

Co daje w efekcie:

$$E(y|\mathbf{x}) = \sum_{i=1}^{T} g_i(\mathbf{x}) F_i(\mathbf{x}). \tag{5.44}$$

Jako funkcje $g(\cdot)$ wybrać można znormalizowane węzły gaussowskie:

$$g_i(\mathbf{x}) = \frac{\exp h_i(\mathbf{x})}{\sum_{j=1}^{T} h_j(\mathbf{x})} \tag{5.45}$$

i

$$\sigma_i(\mathbf{x}) = \exp s_i(\mathbf{x}). \tag{5.46}$$

Natomiast $h_i(\mathbf{x})$ i $s_i(\mathbf{x})$ są realizowane przez sieci neuronowe.

Do uczenia komitetu ME używa się funkcji błędu maksymalnej wiarygodności (dla danego wektora $\mathbf{x}_j$):

$$L_j(\mathbf{x}_j) = \log \sum_{i=1}^{T} g_i(\mathbf{x}_j) G\left(y; F_i(\mathbf{x}_j), \sigma_i(\mathbf{x}_j)^2\right). \tag{5.47}$$

Wtedy można powyznaczać gradienty:

$$\frac{\partial L_j}{F_i(\mathbf{x}_j)} = P(i|\mathbf{x}_j, y_i) \frac{y_j - F_i(\mathbf{x}_j)}{\sigma_i(\mathbf{x}_j)^2}, \tag{5.48}$$

$$\frac{\partial L_j}{h_i(\mathbf{x}_j)} = P(i|\mathbf{x}_i, y_i) - g_i(\mathbf{x}_j), \tag{5.49}$$

$$\frac{\partial L_j}{s_i(\mathbf{x}_j)} = P(i|\mathbf{x}_i, y_i) \left(\frac{(y_j - F_i(\mathbf{x}_j))^2}{\sigma_i(\mathbf{x}_j)^2} - 1 \right). \tag{5.50}$$

Prawdopodobieństwo $P(i|\mathbf{x}_i, y_i)$ dla i-tego eksperta jest wyznaczane poprzez:

$$P(i|\mathbf{x}_i, y_i) = \frac{g_i(\mathbf{x}_j) G\left(y; F_i(\mathbf{x}_j), \sigma_i(\mathbf{x}_j)^2\right)}{\sum_{i=1}^{T} g_i(\mathbf{x}_j) G\left(y; F_i(\mathbf{x}_j), \sigma_i(\mathbf{x}_j)^2\right)}. \tag{5.51}$$

5.8. Komitety heterogeniczne

Z pewnością warto polecić także stosowanie komitetów modeli różnych typów. Wybór modeli nie musi ograniczać się do różnych typów sieci neuronowych, ale można rozpatrywać różne typy modeli adaptacyjnych. Jak wiadomo już z poprzednich rozważań często bywa tak, że niektóre modele działają lepiej dla jednych danych, a inne dla drugich. Użycie różnych typów modeli może w znacznym stopniu zniwelować problemy, na które natykamy się stosując jeden typ modeli.

Poza tym, stosując modele różnych typów w komitecie do danego problemu, uzyskujemy dodatkowo różne typy *wyjaśnień* dlaczego uzyskana została taka, a nie inna klasyfikacja. Używając na przykład w jednym komitecie sieci neuronowych, drzew decyzyjnych, SVM, modeli opartych na podobieństwie, K klasyfikatorów możemy naprawdę dogłębnie wyeksplorować wiedzę drzemiącą w danym zbiorze danych. Mamy wtedy ogólną decyzję, która powinna być stabilna, a dodatkowo *wspierające* decyzje podmodeli bazowych, które mogą wskazywać stopień ufności, regułę klasyfikacyjną, etc.

Takie komitety zostały zaimplementowane w systemie GhostMiner [139].

Komitety heterogeniczne, dzięki swej różnorodności, lepiej dopasowują się do różnych danych. Dzięki temu mogą bazować na mniejszej liczbie modeli bazowych (mniejsza złożoność), niż komitety Boosting czy AdaBooting uzyskując jednocześnie porównywalną jakość klasyfikacji.

5.9. Komitety z lokalną kompetencją

Powyższy komitet, jak i inne już wspomniane, można rozbudować o mechanizm lokalnej kompetencji. Celem tego mechanizmu jest odbieranie prawa głosu w sytuacji, w której wiemy, że dany model bazowy nie radzi sobie najlepiej. W odróżnieniu od komitetu grading (5.7.4), lokalna kompetencja nie jest wynikiem uczenia dodatkowego modelu (meta-model), lecz jest wyznaczana analitycznie. Dodatkowo decyzja o lokalnej kompetencji nie jest zero-jedynkowa, lecz ciągła.

Decyzje poszczególnych podmodeli F_i będą ważone przez $c_{F_i}(\mathbf{x})$. Wtedy końcowa decyzja przynależności danego wektora $\mathbf{x}$ do klasy C jest zdefiniowana poprzez:

$$p(C|\mathbf{x}) = \sum_{i=1}^{T} \bar{c}_{F_i}(\mathbf{x}) p(C|\mathbf{x}, F_i). \tag{5.52}$$

Mechanizm ważenia modeli jest pewną pochodną metody regularyzacji danych, opisanej w rozdziale 6.4. W tym wypadku skupiamy uwagę wokół punktu $\mathbf{x}$ i tam modelujemy rozkład poprawności klasyfikacji nauczonego modelu F_i:

$$\bar{c}_{F_i}(\mathbf{x}) = \frac{c_{F_i}(\mathbf{x})}{\sum_{j=1}^{T} c_{F_j}(\mathbf{x})}, \tag{5.53}$$

gdzie

$$c_{F_i}(\mathbf{x}) = \frac{\sum_{\mathbf{v}\in V^+} G(\mathbf{x};\mathbf{v})}{\sum_{\mathbf{v}'\in V^+\cup V^-} G(\mathbf{x};\mathbf{v}')}, \tag{5.54}$$

$G(\mathbf{x};\mathbf{v})$ jest funkcją Gaussa z centrum w $\mathbf{v}$. Natomiast V^+ i V^- są zdefiniowane jak poniżej

$$V^+ = \{\mathbf{v} : \mathbf{v} \in N_\mathbf{x} \wedge F_i(\mathbf{v}) = C_\mathbf{v}\}, \tag{5.55}$$

$$V^- = \{\mathbf{v} : \mathbf{v} \in N_\mathbf{x} \wedge F_i(\mathbf{v}) \neq C_\mathbf{v}\}, \tag{5.56}$$

gdzie $N_\mathbf{x}$ jest zbiorem punktów wokół $\mathbf{x}$ (wszystkie punkty $\mathbf{x}'$ takie, że $||\mathbf{x}-\mathbf{x}'|| < \epsilon k$. $N_\mathbf{x}$ może być także zbiorem k najbliższych sąsiadów punktu $\mathbf{x}$ ze zbioru treningowego). C_v jest klasą wektora $\mathbf{v}$. Jeśli $N_\mathbf{x}$ definiujemy przez zbiór sąsiadów można rozważać przedefiniowanie $c_{F_i}(\mathbf{x})$ z 5.54 do postaci:

$$c_{F_i}(\mathbf{x}) = \frac{1}{|N_\mathbf{x}|} \sum_{\mathbf{v}\in N_\mathbf{x} \ \wedge \ F_i(\mathbf{v})=C_\mathbf{v}} 1. \tag{5.57}$$

Warto porównać tę metodę z komitetem RegionBoost 5.7.2 i mixture of local experts 5.7.5. RegionBoost jest oparty o model kNN. Metoda powyższa jest niezależna od modelu bazowego, ale na nim opiera swój stopień kompetencji. Mixture of local experts wymaga procesu uczenia dla wyznaczenia systemu ważenia. Inny ciekawy sposób analizy lokalnej kompetencji przedstawił Duch (i. in.) w [68].

Taki mechanizm niewątpliwie przypomina mechanizm pobudzania odpowiednich ośrodków mózgu, w zależności od ich wstępnie ocenionej kompetencji i przydatności do przetwarzania informacji danego typu (informacji wzrokowej, słuchowej, etc.).

Rozdział 6

Wstępne i końcowe przetwarzanie danych

Jak już stwierdzono w poprzednim podrozdziale, na ostateczny całkowity błąd ma wpływ nie tylko niedoskonałość modelu, ale i niereprezentatywność danych. Dlatego też niezwykle ważne jest, aby dane były jak najlepsze. Na jakość danych wpływa wiele czynników. Do głównych należą: jakość dokonanych pomiarów, jakość zbierania i przechowywania danych (nierzadko mamy do czynienia z brakującymi informacjami), zawartość informacyjna poszczególnych cech (atrybutów) i wstępne przetwarzanie danych. Uwag na tematy wstępnego przetwarzania danych można doszukać się w różnych publikacjach, a szczególnie warto wymienić [28, 241, 13, 132].

6.1. Transformacje danych

Nierzadko wartości niektórych cech nie mają rozkładów liniowych czy normalnych, obserwuje się czasami eksponencjalny czy logarytmiczny rozrzut danych w pewnym wymiarze. Takie cechy najczęściej mogą wnieść więcej informacji po dokonaniu transformacji odwrotnej od tej, która została zaobserwowana w danym wymiarze. Z kolei gdy pewne cechy przyjmują wartości symboliczne należy rozważyć użycie miar heterogenicznych. Więcej informacji na ten temat znajduje się w podrozdziale 1.2.1.2.

W problemach klasyfikacyjnych większość metod zazwyczaj zakłada podobny rozrzut danych w poszczególnych wymiarach przestrzeni wejściowej. Stąd też najczęściej dokonuje się pewnej transformacji danych, która zniweluje zbyt duże, początkowe dysproporcje pomiędzy wartościami w poszczególnych wymiarach.

Najprostszą stosowaną transformacją jest **normalizacja** danych tak, aby po transformacji wartości mieściły się w przedziale $[0, 1]$ (w podobny sposób można

dokonać transformacji tak aby wartości mieściły się w przedziale$[-1,1]$). W tym celu dla każdej cechy dokonuje się poniższego przekształcenia:

$$x_i' = \frac{x_i - x_{min}}{x_{max} - x_{min}}, \tag{6.1}$$

przy czym

$$x_{min} = \min_i x_i, \tag{6.2}$$

$$x_{max} = \max_i x_i, \tag{6.3}$$

a i zmienia się od 1 do n.

Powyższa transformacja może być czasami wręcz niebezpieczna. Gdy mamy do czynienia z błędnymi wartościami w pewnych cechach, może się okazać iż leżą one daleko poza normalnym zakresem wartości danej cechy. W takim przypadku dochodzi do nadmiernego *ściśnięcia* znormalizowanych wartości danej cechy, niosących najwięcej istotnych informacji. Z tego też powodu często stosuje się powyższą normalizację danych, ale wartości x_{min} i x_{max} wybiera się nie spośród całego zbioru $\mathcal{S} = \{x_1, x_2, \ldots, x_n\}$, lecz po odrzuceniu ze zbioru $\mathcal{S}$ k% najmniejszych i największych wartości (za k przyjmuje się najczęściej 5 lub 10). Taką normalizację najczęściej nazywa się **normalizacją z obcięciem**.

Innym sposobem jest **standaryzacja** danych:

$$x_i' = \frac{x_i - \bar{x}}{\sigma_x}, \tag{6.4}$$

gdzie $\bar{x}$ jest wartością średnią, natomiast σ_x jest standardowym odchyleniem:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \tag{6.5}$$

$$\sigma_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}. \tag{6.6}$$

Stosowanie podstawowej normalizacja może prowadzić do złych konsekwencji w wyniku zaburzeń wartości danej cechy. Stosowanie normalizacji z obcięciem, czy standaryzacji, jest bezpieczniejsze i zazwyczaj nie prowadzi do istotnych różnic w późniejszym procesie adaptacji.

Czasem, gdy rozkład danych dla pewnej cechy (czy grupy cech) nie jest normalny lub liniowy i bardziej zależy nam na zachowaniu relacji pomiędzy poszczególnymi elementami, niż odległości, jakie one wyznaczają, można skorzystać z poniższego przekształcenia (niezależnie dla wszystkich cech):

$$x_i' = 2|\{x : x < x_i\}| + |\{x : x = x_i\}|, \tag{6.7}$$

$|\mathcal{Z}|$ oznacza tu moc zbioru $\mathcal{Z}$.

Powyższe przekształcenie oparte o podobieństwo, a nie o realne odległości, może być zastosowane do części lub wszystkich wymiarów. Ważną własnością tego przekształcenia jest niwelacja nieliniowości rozkładu danych w wymiarze, który poddaje się transformacji.

Wadą może okazać się łączenie się (niemal) uprzednio odległych skupisk danych o ciągłych wartościach, pomiędzy którymi nie było żadnych innych danych. Jednak można temu zaradzić, wprowadzając do powyższego przekształcenia informacje o punktach, które są istotnymi punktami podziału wartości cech w wymiarze, który aktualnie podlega transformacji. Punkty takie można wyznaczyć korzystając na przykład z kryterium używanego w drzewach CART [25], kryterium dipolowego [17, 15, 16], czy SSV [110] lub metod opartych na histogramach i dendrogramach. Wyznaczanie punktów podziałów powinno przebiegać równocześnie dla wszystkich wymiarów podlegających transformacji. Prowadzi to do optymalnego wyboru tych punktów i przypisania im wartości, które odzwierciedlają ich wpływ na polepszenie podziału.

Tak zmodyfikowaną transformację można zdefiniować poprzez:

$$x'_i = 2|\{x : x < x_i\}| + |\{x : x = x_i\}| + \sum_{p \in P^k \wedge p < x_i\}} \kappa_p, \tag{6.8}$$

gdzie P^k jest zbiorem punktów podziału k-tego wymiaru. Natomiast κ_p jest współczynnikiem określającym istotność podziału w punkcie p względem innych wyznaczonych punktów.

6.2. Wartości nietypowe i brakujące

6.2.1. Wartości nietypowe

Problemów mogą dostarczyć również wartości nietypowe zaburzając proces uczenia i w końcowym efekcie powodując istotne obniżenie poziomu generalizacji. Dobrymi przykładami takich wartości są wszelkie irracjonalne wartości w danym zagadnieniu, np. temperatura ciała człowieka wynosząca 70 stopni Celsjusza. Może też dojść do powstania wartości nietypowych w parach cech lub grupach cech. Na przykład: *wiek = 5 lat*, *wzrost = 166 cm*. W tym przypadku wartości poszczególnych cech są jak najbardziej dopuszczalne, ale ich kombinacja jest bardzo mało prawdopodobna. Pochodzenie wartości nietypowych jest zazwyczaj związane z procesem dokonywania pomiarów, lub ich kolekcjonowania i przechowywania.

W miarę możliwości należy dokonać poprawek takich wartości, bądź ich usunięcia, lub usunięcia całych wektorów, w których występują takie wartości.

Warto także w takich przypadkach wziąć pod uwagę możliwości jakie daje regularyzacja danych opisana w rozdziale 6.4. Jej zadaniem miedzy innymi jest automatyczne wyznaczanie wartości nietypowych.

6.2.2. Wartości brakujące

Inny problem stanowią wartości brakujące w wektorach, opisujących różne przypadki. Niewątpliwie najlepszym lekarstwem jest właściwe uzupełnienie takich braków. Jednak w praktyce jest to niemal zawsze niewykonalne. Jeśli nie można uzupełnić, to należy wziąć pod uwagę możliwość redukcji zbioru danych o wektory, które posiadają wartości brakujące, lecz gdy zbiór danych nie jest wystarczająco duży, to i takie rozwiązanie nie możne być zastosowane. Ważna jest przyczyna powstania wartości brakujących. Możemy mieć do czynienia z dwoma istotnie różnymi przyczynami. Pierwsza najczęściej jest spowodowana decyzją, której konkluzją było niewykonanie jakiejś analizy czy pomiaru. Druga grupa to braki wynikające z zagubienia informacji lub na przykład utraty kontaktu z pacjentem.

Jedną z ciekawszych możliwości jest pominięcie wymiaru, w którym mamy do czynienia z wartością brakującą. Jednak nie każdy model umożliwia opuszczenie takiego wymiaru bez znacznych skutków ubocznych. Możliwość opuszczenia pewnego wymiaru podczas klasyfikacji mamy, gdy model korzysta z separowalnych funkcji transferu (por. podrozdział 1.4.6 i 1.4.7), na przykład z funkcji Gaussa, czy funkcji bicentralnych.

Inną możliwością postąpienia z wartościami brakującymi, jest przypisanie im specjalnych wartości spoza przedziału, w którym występują wartości w danym wymiarze. Powoduje to uwzględnienie informacji o, na przykład, niedokonaniu pewnego badania, czy niewykonaniu jakiegoś pomiaru.

Jeszcze innym, naprawdę wartym uwagi, podejściem jest zastąpienie wartości brakującej wartością, która jest najbardziej prawdopodobna (np. najczęściej występującą w danym wymiarze i dla danej klasy bądź zakresu wartości) lub najoptymalniejszą wartością, zakładając wykorzystanie pewnego klasyfikatora $\mathcal{X}$. Wtedy w oparciu o dany model $\mathcal{X}$ staramy się wyznaczyć najbardziej prawdopodobną wartość jaką mogłaby przyjąć cecha o nieokreślonej wartości. Dla przykładu łatwo można to zrobić w oparciu o model k najbliższych sąsiadów (kNN). Po znalezieniu k najbliższych sąsiadów, opuszczając uprzednio wymiar z wartością brakującą, można poprzez interpolację ustalić wartość brakującą.

Taka decyzja jest obarczona stosunkowo niskim ryzykiem. Problem jednak narasta gdy mamy więcej wartości brakujących w ramach jednego wektora.

Stosowanie wartości średnich danego wymiaru, jako substytutu wartości brakującej jest znacznie gorszym rozwiązaniem i może powodować przyjmowanie wartości dla danej cechy, które dla pewnych klas mogą być mało prawdopodobne. Dla przykładu, gdy założyć, iż wartości brakujące zastąpi się najczęściej występującą wartością danej cechy, można spodziewać się, że brakująca informacja o wzroście niemowlęcia wyniesie około 165 cm. W takich przypadkach wykorzystując na przykład model kNN w wyznaczaniu wartości zastępczych ryzyko powinno być znacznie mniejsze. Niestety metoda zastępowania wartości brakujących średnimi dla wartościami danej cechy jest często stosowana.

Nieco lepszym sposobem jest zastępowanie wartości brakujących wartościami średnich wyznaczonych w ramach wektorów o tej samej klasie.

Wszystkie powyżej zaprezentowane podejścia do problemu wartości brakujących rozwiązują problem jedynie częściowo i mogą powodować różne konsekwencje, niekoniecznie oczekiwane. Lecz gdy wrócić do źródeł powstawania brakujących wartości, to można dojść do konstruktywnych konkluzji.

Może tu także wspomnieć o innych sposobach postępowania z wartościami brakującymi, takimi jak *multiple imputation* [217, 3]. To ogólny schemat postępowania, który doczekał się rożnych rozwiązań finalnych (por. [124]). Metoda polega na wygenerowaniu (z losowością) pewnej liczby wektorów w oparciu o wektor z wartościami brakującymi, następnie stosuje się analizę używając metod danymi bez wartości brakujących. Na koniec trzeba dokonać oceny uzyskanych wyników i połączenia ich w jeden wektor–wynik już bez wartości brakujących. Można także stosować algorytm EM (*ang. expectation maximization*) [103].

Źródła wartości brakujących to:

- wartość nie była wyznaczana — np. lekarz nie zlecił wykonania danego testu
- wartość nie została wyznaczona — np. lekarz zlecił wykonanie danego testu, lecz wyniku nie ma

W pierwszym przypadku zastępowanie wartością średnią bądź niby najbardziej prawdopodobną może nieść za sobą duże ryzyko, ponieważ *nie zlecenie* wyznaczenia wartości na pewno było świadomą decyzją, popartą pewną wiedzą *a priori*, o której trudno założyć, że istnieje o niej informacja w samych danych. Dlatego też raczej należy przypisać pewną ustaloną stałą wartość spoza zakresu występowania danych w tym wymiarze, która będzie symbolizowała owo *nie zlecenie* wyznaczenia wartości.

Natomiast w drugim przypadku wartość w ewidentny sposób została zagubiona w procesie jej wyznaczania bądź przechowywania. Nie ma w tym żadnej świadomej działalności podobnej do poprzedniego przypadku. W takim przypadku bardziej wskazanym wydaje się uzupełnienie wartości możliwie najbardziej prawdopodobną wartością, przy wykorzystaniu do tego wszelkich informacji dostępnych w danych uczących lub korzystaniu z pewnego klasyfikatora[1], wspomagającego sam proces uprawdopodobniania owej wartości.

Takie postępowanie jest połączeniem dwóch, już wcześniej opisywanych metod, ale z uwzględnieniem źródła wartości brakującej. W rezultacie otrzymuje się bardziej racjonalną metodę postępowania.

6.3. Metody selekcji i ważenia cech

Bardzo pozytywnie na przebieg procesu uczenia, jak i ostateczny poziom generalizacji, może wpłynąć wybór istotnych cech, spośród wszystkich wymiarów danych wejściowych. Powstało bardzo wiele i bardzo różnych metod wyboru

[1]Jest to niedopuszczalne w przypadku aproksymacji.

istotnych cech. W pracach [53, 80, 80, 13, 4] można znaleźć porównania metod selekcji cech.

Metody różnią się sposobami doboru ilości i kolejności cech w analizie. Niektóre z metod prowadzą analizę wykorzystując całą przestrzeń możliwości, która czasem może być duża ($O(2^N)$). Inne metody dobierają lub odrzucają cechy heurystycznie (wtedy złożoność zazwyczaj nie przekracza $O(N^2)$) lub analizują różne losowe kombinacje cech (wtedy maksymalna złożoność znów wnosi $O(2^N)$, lecz liczbę losowań ogranicza się z góry).

Metody doboru cech mogą też różnić się metodami oceny cech. Można prowadzić analizę w oparciu o odległość (separowalność czy dyskryminację — np. drzewa decyzyjne). Korzysta się również z różnych miar informacji przy ocenie danego wymiaru. Innym sposobem jest też badanie zależności czy korelacji pomiędzy wymiarami. Bada się także wymiary pod kątem utrzymywania pewnego warunku zgodności, jednocześnie wymuszając korzystanie z jak najmniejszej liczby cech. Jeszcze inną rodzinę stanowią metody korzystające z pewnego klasyfikatora (czy aproksymatora), który służy do oceny dokonanego doboru cech (np. [67]).

6.3.1. Ważenie i selekcja cech dyskretną metodą quasi-gradientową

Ważenie cech polega na wyznaczeniu współczynników, które będą skalowały poszczególne wymiary (cechy), np. podczas wyznaczania funkcji odległości. Współczynniki ważenia są ogólniejszą formą niż metody ekstrakcji (eliminacji) cech. Ekstrakcja cech odpowiada użyciu współczynników ważenia o wartościach $0, 1$. Ważenie wyznacza współczynniki skalowania, które przyjmują wartości rzeczywiste dodatnie. Nie tracąc na ogólności można dokonać przeskalowania wag do przedziału $[0, 1]$. Współczynniki ważenia mogą być także używa w rankingu ważności cech co może być przydatne dla wielu algorytmów

Poniższa metoda ważenia cech może być równie dobrze stosowana do eliminacji cech. Umożliwia ona start z pustej bazy cech, bądź z pełnej bazy cech, co czasem może mieć ogromne znaczenie (szczególnie dla przestrzeni o wielu cechach). W ogólności dowolny algorytm uczenia może być użyty z tą metodą ważenia cech. W przykładach zastosowań tego algorytmu ważenia będzie on używany wraz z modelem k najbliższych sąsiadów.

Będzie można zobaczyć, że rezultaty otrzymane tą metodą są naprawdę bardzo ciekawe. Metoda umożliwia także uzyskanie alternatywnych zestawów wektorów wag. Może to mieć duży wpływ na zrozumienia wpływu poszczególnych cech danych.

Wiadomo, że oryginalny zestaw cech pewnej bazy danych może być daleki od takiego, który umożliwiłby uzyskanie jak najlepszej generalizacji klasyfikacji, czy aproksymacji [13, 74, 75]. Może się okazać, że pewne cechy nie niosą ze sobą żadnej informacji, bądź mogą być redundantne ze względu na inne cechy. Ważenie i redukcja cech upraszczają wtedy problem estymacji optymalnego modelu.

Różne metody ważenia mogą stosować odmienne strategie w wydobywaniu informacji o istotności poszczególnych cech [53, 80, 13, 4]. Część z metod może oceniać cechy, czy grupy cech przez probabilistyczne bądź teorioinformacyjne zależności. Inne metody z kolei mogą posiłkować się informacją, jaką może dać pewien model uczący jako reakcję na próby zmian współczynników ważenia cech przez algorytm ważenia [259, 257, 80]. Takie informacje mogą być wykorzystane w następnym kroku procedury ważenia. Proces ten prezentuje pewnego rodzaju symbiozę pomiędzy dwoma algorytmami, które kooperując zwiększają szanse ostatecznego powodzenia w klasyfikacji, czy aproksymacji danego problemu.

Algorytm zaprezentowany poniżej należy właśnie do drugiej z powyższych grup metod ważenia cech i będzie on używany w przykładach wraz z dobrze znanym modelem k najbliższych sąsiadów (kNN) [49], choć w ogólności można użyć dowolnego algorytmu.

6.3.1.1. Algorytm ważenia cech

Ogólna idea algorytmu dyskretnego quasi-gradientowego ważenia cech polega na poszukiwaniu wektora zmian współczynników ważenia, po czym wprowadza się te zmiany. Proces powtarzany jest jak długo, aż daje się uzyskać poprawę klasyfikacji. Równolegle następują zmiany pewnych parametrów, które kontrolują cały proces zmian wag. Dzięki tym parametrom i ich roli algorytm łatwo radzi sobie z ucieczką z lokalnych minimów (choć globalne minimum nie jest zagwarantowane).

Drugim głównym celem algorytmu jest uzyskanie takich wag cech, które nie pogorszą klasyfikacji czy aproksymacji po ich zastosowaniu do finalnego modelu. Należy pamiętać, że oryginalna baza cech może być bliska optymalnej i wtedy trudno liczyć na radykalną poprawę.

Powtarzając procedurę ważenia wielokrotnie można uzyskać kilka różnych rozwiązań, czyli różnych kompletów wektorów wag. Nie jest to obrazem niestabilności, tylko faktu, iż często istnieje kilka alternatywnych zestawów wag, które dają zbliżone dobre rezultaty. Może to być bardzo przydatne. Dzięki temu, na przykład, może się okazać, że stosowanie pewnej inwazyjnej metody w medycynie, celem wyznaczenia jakiegoś wskaźnika nie jest konieczne i można będzie zastąpić go czymś innym.

6.3.1.1.1. Uczenie z wykorzystaniem walidacji skośnej Na główną pętlę algorytmu składa się użycie mechanizmu walidacji skośnej (ang. crossvalidation, CV) (porównaj rozdział 7.1) do wyznaczania optymalnych współczynników wag. Głównym celem użycia walidacji skośnej do uczenia jest możliwość trenowania modeli na częściach treningowych walidacji skośnej i następnie walidacji tych modeli na odpowiadających im częściach testowych (walidacyjnych). Daje to możliwość obserwacji efektów uczenia modeli na danych, które nie były wykorzystane w procesie uczenia, a to daje nam estymację możliwości potencjalnego modelu i tym samym możliwość weryfikacji zastosowanych parametrów

uczenia.

Tak jak w przypadku walidacji skośnej do testowania jakości klasyfikacji czy aproksymacji cały zbiór danych dzieli się na n (możliwie) równych części $\mathcal{S}_i$, $i = 1, \ldots, n$. W każdej iteracji walidacji skośnej używa się procedury *FindWeights* do wyznaczenia wag. *FindWeights* używa dwóch zbiorów $\hat{\mathcal{S}}_i$

$$\hat{\mathcal{S}}_i = \bigcup_{k=1,\ldots,n,\ k\neq i} \mathcal{S}_k, \tag{6.9}$$

jako zbioru uczącego i $\mathcal{S}_i$ jako zbioru walidacyjnego. Na zbiorze walidacyjnym dokonuje się walidacji procesu uczenia na części treningowej, dzięki czemu można oceniać jakość zmian wag.

Jako rezultat procedura *FindWeights* zwraca wektor współczynników $\mathbf{w}_i$. Mamy więc:

$$\mathbf{w}_i = FindWeights(\hat{\mathcal{S}}_i, \mathcal{S}_i). \tag{6.10}$$

Każda z wag $\mathbf{w}_i$ znajduje się w przedziale $[0, 1]$, co oczywiście nie wpływa ograniczająco na końcowe możliwości algorytmu.

Szczegóły procedury *FindWeights* zostaną przedstawione poniżej.

6.3.1.1.2. Estymacja wag końcowych Wektor $\mathbf{w}_i$ składa się z wag dla każdej z cech $k = 1, \ldots, d$, gdzie d oznacza rozmiar zbioru danych $\mathcal{S}$. Teraz wyznaczmy sumę wszystkich zestawów wag z walidacji skośnej:

$$\mathbf{w} = \sum_{i=1}^{n} \tilde{\mathbf{w}}_i \tag{6.11}$$

i zastosujmy normalizację, aby końcowe wagi znalazły się w przedziale $[0, 1]$:

$$\tilde{\mathbf{w}} = \mathbf{w} \,/\, \max_{k=1,\ldots,d} w_k, \tag{6.12}$$

gdzie $\mathbf{w} = [w_1, \ldots, w_d]$.

Powyższa część algorytmu uczenia przez walidację skośną jest naprawdę prosta. Po zakończeniu uczenia wszystkich modeli przez walidację skośną, finalny model stanowi uśrednienie modeli z walidacji skośnej.

Teraz można już przejść do opisu procedury *FindWeights*.

6.3.1.1.3. Procedura *FindWeights* Pierwszy krok to przypisanie początkowych wartości wag $w_k = 1, \quad k = 1, \ldots, d$. W tym przypadku algorytm startuje z pełnej bazy cech. Jak już wspomniano, czasem ważne jest, by móc wystartować z pustej bazy, aby zaoszczędzić na złożoności problemu. Jest to szczególnie przydatne dla wielowymiarowych baz danych. Wtedy przypisujemy wstępnie wagi zerowe $w_k = 0$, co oznacza, że algorytm najpierw będzie próbował dołączać cechy do pustej bazy cech (dalsza część jest identyczna). W przypadku startu z pustej bazy należy rozważyć ustalenie maksymalnej liczby cech dokładanych jednocześnie i ich para-losowy wybór.

Główna pętla procedury *FindWeights* (jedna z trzech) jest powtarzana tak długo, jak długo daje się ulepszać poprawność klasyfikacji na zbiorze walidacyjnym (pętla ta będzie zwana główną).

Cel głównej pętli to obserwacja zmian jakie niosą zmiany wag poszczególnych cech i zapamiętywanie tych, które mogą zwiększyć poprawność klasyfikacji. Wielkości zmiany wag są definiowane przez Δ. Zmiany mogą być dodatnie bądź ujemne.

Wewnętrzna pętla wyznacza wielkości v_i^+ i v_i^- dla każdej z cech:

$$v_i^+ = validate(\mathbf{w}^+, i), \tag{6.13}$$

$$v_i^- = validate(\mathbf{w}^-, i), \tag{6.14}$$

gdzie $\mathbf{w}^\pm = [w_1, \ldots, w_{i-1}, w_i \pm \Delta, w_{i+1}, \ldots, w_d]$. $validate(\mathbf{w})$ jest funkcją, która wylicza poprawność na zbiorze walidacyjnym $\mathcal{S}$ (patrz równanie 6.10) dla modelu uczącego $\mathcal{M}$ który jest trenowany na zbiorze danych $\hat{\mathcal{S}}$. Do uzyskania rezultatów przedstawionych w rozdziale 6.3.1.3 użyto modelu k najbliższych sąsiadów (kNN) jako modelu $\mathcal{M}$.

Wektory $\mathbf{v}^+$ i $\mathbf{v}^-$ zawierają informacje o efektywności zmian wag o czynnik Δ. Korzystając z informacji w wektorach $v_i^\pm$ i pamiętając jaka poprawność była osiągnięta poprzednio, można określić nowy zestaw wag:

$$w_i' = w_i - \theta\Delta \quad \text{if} \quad v_i^- > v, \tag{6.15}$$

$$w_i' = w_i + \theta\Delta \quad \text{if} \quad v_i^+ > v \wedge v_i^- <= v, \tag{6.16}$$

gdzie θ definiuje szybkość zmian (rezultaty przedstawione w poniższej części rozdziału korzystały z $\theta = 1$), a $v = validate(\mathbf{w})$.

Powyższe równania sprawiają, że wszystkie cechy, których zmiany mogą pomóc będą zmienione. Jeśli zdarzyło by się, że dla danej cechy można i zmniejszyć i powiększyć wagę cechy, i obie zmiany powiększają jakość modelu, wybrane zostanie zmniejszenie wagi. Jeśli dla nowych wag dokładność $v_{new} = validate(\mathbf{w}')$ będzie mniejsza niż $v = validate(\mathbf{w})$, procedura powraca do poprzednich wag $\mathbf{w}$. W przeciwnym przypadku wektor wag $\mathbf{w}'$ staje się nowym wektorem wag.

Tak długo, jak tylko dla bieżącego parametru Δ daje się uzyskać lepszą dokładność, pętla wewnętrzna będzie kontynuowana. Najczęściej Δ na początku przyjmuje wartość 1. Skok Δ jest zmniejszany, jeśli tylko nie można już zwiększyć poprawności klasyfikacji na zbiorze walidacyjnym:

$$\Delta = \Delta / 2. \tag{6.17}$$

Następnie jeśli tylko Δ nie jest mniejsza niż Δ_{min}, pętla środkowa jest kontynuowana kolejny raz.

Jeśli Δ jest mniejsza niż Δ_{min} (Δ_{min} może dla przykładu przyjąć wartość 0.01), pętla środkowa jest przerywana, a wagi poddawane są normalizacji:

$$\tilde{\mathbf{w}} = \mathbf{w} / \max_{k=1,\ldots,d} w_k. \tag{6.18}$$

Główna pętla jest powtarzana (z początkowym $\Delta = 1$) tak długo, jak długo można poprawić dokładność klasyfikacji na zbiorze walidacyjnym.

Podczas wykonywania głównej pętli algorytm jest zdolny do *wyskakiwania* z lokalnych minimów i tym samym do dalszego poszukiwania lepszych rozwiązań, często prowadząc do zmniejszenia liczby istotnych wag. Unikanie lokalnych minimów jest możliwe tylko dzięki głównej pętli.

Na stronie 211 można znaleźć schemat przedstawionego algorytmu ważenia cech.

6.3.1.2. Eliminacja cech

Powyższy algorytm może być stosowany także do eliminacji cech. Można to zrobić na dwa sposoby. Pierwszy sposób polega na wyznaczeniu wektora wag zgodnie z powyższym algorytmem; następnie wszystkie wagi poniżej pewnego progu α (np. $\alpha = .02$) można uznać za zbyt mało istotne i ustawić na zera. Współczynniki niezerowych wag definiują cechy, które nie zostały wyeliminowane.

Drugi sposób polega na modyfikacji równań 6.13 i 6.14 tak, aby umożliwiały one zmiany wartości wag z 0 tylko na 1, a z 1 tylko na 0. Uzyskujemy w ten sposób algorytm przeszukiwania wartości binarnych. Pozostałe części algorytmu pozostają bez zmian.

Można także rozważyć testowanie pośrednich rozwiązań, jakie uzyskuje się w poszczególnych iteracjach walidacji skośnej. Często mogą one dawać również bardzo ciekawą generalizację.

6.3.1.3. Przykłady rezultatów dla ważenia cech

Poniżej przedstawione zostaną rezultaty sprawdzania algorytmu ważenia dla różnych baz danych. Sprawdzano, na ile algorytm może wpłynąć na poprawę klasyfikacji po zastosowaniu wyznaczonego wektora wag dla zestawu cech. Wszystkie testy zostały przeprowadzone dla baz danych dostępnych przez Machine Learning Database Repository z uniwersytetu Kalifornijskiego w Irvine [182]. Każdy test był przeprowadzany przy użyciu dziesięciokrotnej walidacji skośnej z wyjątkiem danych, które posiadają swój własny zbiór testowy i treningowy. Walidacja skośna była powtarzana dziesięć razy i zaprezentowane zostały uśrednione rezultaty poprawności na zbiorach testowym i treningowym. Najczęściej należało także dokonać standaryzacji przed procesem uczenia danych, aby cechy miały zbliżony rozkład danych.

Każda tabela z rezultatami składa się z czterech części. Pierwsza prezentuje rezultaty uzyskane z pomocą algorytmu ważenia dla modelu k najbliższych sąsiadów, druga część prezentuje rezultaty modelu kNN bez udziału ważenia. Trzecia informuje o części, jaką stanowi najliczniejsza spośród klas w zbiorze danych. Ostatnia część prezentuje rezultaty innych modeli, które osiągały najlepsze wyniki dla odpowiedniej bazy danych. Ta część zawiera rezultaty tylko

Algorithm 1: Schemat dyskretnego quasi-gradientowego algorytmu ważenia cech

Function $FeatureWeighting(\mathcal{S})$
  $\mathcal{S}_1, \ldots, \mathcal{S}_1$ — n równe części $\mathcal{S}$:
  $\hat{\mathcal{S}}_i := \bigcup_{k=1,\ldots,n,\ k\neq i} \mathcal{S}_k$
  for $i = 1$ to n **do**
    $\mathbf{w}_i := FindWeights(\hat{\mathcal{S}}_i, \mathcal{S}_i)$
  end for
  $\mathbf{w} := \mathbf{w}_1 + \ldots + \mathbf{w}_n$
  $w_{max} := \max_{k=1}^{d} w_k$
  $\mathbf{w} := \mathbf{w} / w_{max}$
  return $\mathbf{w}$
Function $FindWeights(\hat{\mathcal{S}}, \mathcal{S})$
  $lastAcc := -1; back := false$
  repeat
    $\Delta := 1$
    $lastPhaseAcc := lastAcc$
    $lastAcc := newAcc := validate(\mathbf{w})$
    while $\Delta >= \Delta_{min}$ **do**
      repeat
        if back **then**
          $lastAcc := newAcc$
        else
          $back := false$
        end if
        for $k = 1$ to d **do**
          $v_i^+ := validate(\mathbf{w}^+, i)$
          $v_i^- := validate(\mathbf{w}^-, i)$
          if $v_i^{\pm} > lastAcc$ **then**
            $w_i' := w_i \pm \theta\Delta$ if $w_i' < 0$ then $w_i' := 0$
          end if
        end for
        $newAcc := validate(\mathbf{w}')$
        if $newAcc > lastAcc$ **then**
          $\mathbf{w} := \mathbf{w}'$
        else
          $back := true$
        end if
      until $newAcc <= lastAcc$
      $\Delta := \Delta / 2$
    end while
    $w_{max} := \max_{k=1}^{d} w_k$
    $\mathbf{w} := \mathbf{w} / w_{max}$
  until $lastPhaseAcc >= lastAcc$
  return $\mathbf{w}$

najlepszych modeli dla danych baz danych. Dla porównania, proszę zwrócić uwagę, iż *k-weight* [66] i GIBL [257] są algorytmami, które także uwzględniają wpływ poszczególnych cech na model (tj. są metodami ważenia cech). Warto też zwrócić uwagę, że wiele modeli z którymi porównany jest algorytm ważenia można tylko czasem spotkać w czołówce tabel, często też wogóle nie ma ich dla niektórych baz danych, z powodu ich niskiej poprawności.

W tabelach z rezultatami zastosowano następującą notację: kNN [CVx] [M]. k oznacza liczbę sąsiadów dla modelu kNN. *CVx* jeśli istnieje oznacza, że był użyty algorytm ważenia cech z *x*-krotną walidacją skośną. Jeśli występuje *M* oznacza to, że użyto innej metryki, niż metryka euklidesowa (może to być miara odległości Manhattan, Minkowski 1.13 lub Canberra 1.19.

Można zauważyć, że dla niektórych zbiorów można uzyskać znaczne polepszenie poziomu klasyfikacji, natomiast dla innych zbiorów poziom klasyfikacji podnosi się nieznacznie bądź pozostaje na tym samym poziomie. Nigdy nie zaobserwowano istotnego pogorszenia poziomu klasyfikacji.

Tabela 6.1 porównuje rezultaty uzyskane dla modeli: 1 najbliższego sąsiada, k najbliższych sąsiadów (k dla którego uzyskano najlepszy rezultat i najlepszej miary, jednej z euklidesowych, Manhattan, Canberra, Minkowskiego), ważonego kNN (WkNN) i najlepszego znanego model (kolumna *Najlepszy*) dla danego zbioru danych.

Kolumna *N-kNN* opisuje różnicę pomiędzy najlepszym znanym modelem, a modelem kNN; kolumna *N-WkNN* pokazuje różnice pomiędzy ważonym kNN'em i najlepszym znanym modelem. Wiersz *śr. różnica* pokazuje średnie różnice pomiędzy modelem kNN lub WkNN i najlepszym znanym modelem. Pokazuje to, jaki uzyskano średni postęp dzięki używaniu algorytmu ważenia.

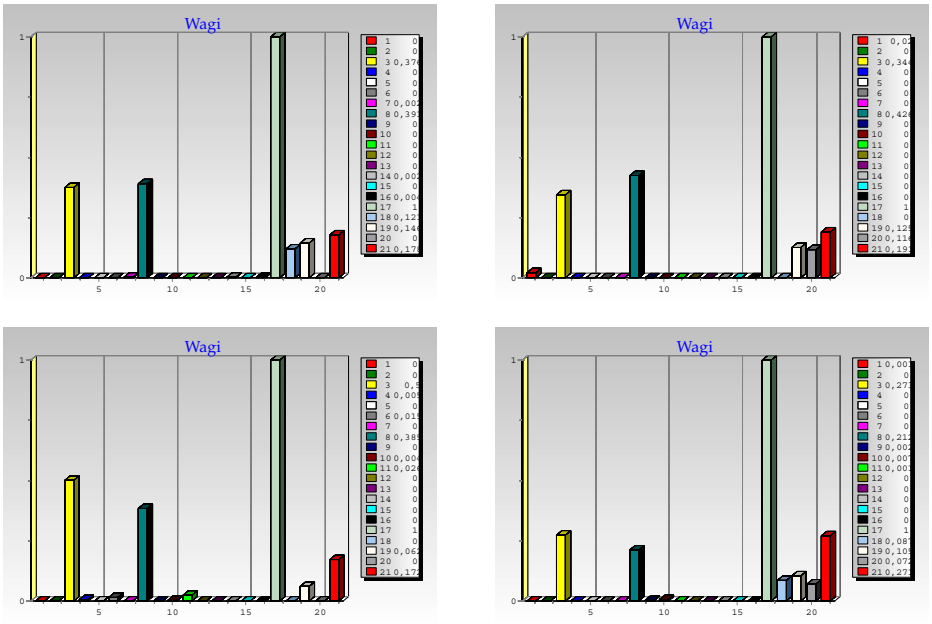
%	1NN	kNN	**WkNN**	Najleszy	N−kNN	N-WkNN	ref.
Thyroid	93,14	96,3	**98,56**	99,36	3,06	0,8	C-MLP2LN +ASA [60]
Appendicitis	81,2	88,14	**88,72**	90,9	2,76	2,18	IncNet [137]
Australian	80,04	84,7	**86,49**	86,9	2,2	0,41	Cal5 [185]
Flag	48,52	50,91	**62,04**	62,4	11,49	0,36	CART [272]
Heart	76,54	83,72	**84,21**	85,1	1,38	0,89	28NN f. sel. [66]
Breast	95,18	96,98	**97,05**	97,2	0,22	0,15	SVM (5xCV?)
Glass	70,45	73,5	**80,81**	78,55	5,05	-2,26	GIBL [257]
Wine	95,38	97,53	**97,35**	98,3	0,77	0,95	Per [272]
śr. różnica	7,28	3,37	**0,43**	0			

Tabela 6.1: Dokładności dla 1NN, kNN, ważonego kNN, najlepszego znanego modelu i różnice pomiędzy 1NN, kNN, WkNN a najlepszym modelem.

6.3.1.3.1. Baza danych tarczycy Ta baza (*Thyroid*) jest jedną z lepiej znanych pośród baz powszechnie używanych w publikacjach dotyczących inteligencji

obliczeniowej. Baza jest z pewnością nietrywialna i tylko część z metod daje satysfakcjonujące rezultaty. Dokładniejszy opis danych został umieszczony w rozdziale 7.2.2.

Poprawność na zbiorze testowym dla modelu 1NN (kNN z 1 sąsiadem) jest bardzo niska i wynosi tylko 93.14% (patrz tab. 6.2). Dla 3NN z miarą Manhattan, poprawność wyniosła 94.4%, lecz dla ważonej wersji kNN uzyskano poprawność 98.36%. Najlepszy rezultat był otrzymany dla 3NN z miarą Canberra – 98.56%. Taki rezultat udało się uzyskać, dzięki usunięciu cech, które wprowadzały zbędny szum, bądź były redundantne. Na rysunku 6.1 przedstawiono kilka wektorów wag, jakie uzyskano podczas uczenia.



Rysunek 6.1: Kilka zestawów wag uzyskanych dla zbioru tarczycy.

6.3.1.3.2. Dane wyrostka robaczkowego (*Appendicitis*) Dokładniejszy opis danych został umieszczony w rozdziale 7.2.2.

To jest przykład danych, na których ważenie cech nie przynosi dużej poprawy, ale i nie powoduje pogorszenia wyników na testowej części. Tabela 6.3 opisuje rezultaty ważenia.

6.3.1.3.3. Dane *Australian credit* Ten zbiór danych składa się z 690 wektorów, 14 cech i jest problemem dwuklasowym. Najlepszy rezultat został osiągnięty dla 7NN CV3 z miarą Manhattan. Wyniósł 86.49% na części testowej, natomiast bez

Model	Test	Train
3NN CV3 Camberra	98.56	99.37
3NN CV2 Manh	98.49	99.33
3NN CV2 Manh	98.36	99.28
1NN CV2 Manh	98.26	100.00
3NN CV2	98.28	99.24
1NN CV2	98.26	100.00
2NN CV2	98.36	99.28
3NN Camberra	96.30	98.49
3NN Manh	94.40	96.58
1NN Manh	93.76	100.00
1NN	93.14	100.00
większość	92.71	-
SSV [111]	99.36	99.90
C-MLP2LN/ASA [60]	99.36	99.90
MLP a,b opt. [60]	99.36	99.90
CART [255]	99.36	99.80
PVM [255]	99.36	99.80
SSV [111]	99.36	99.76
IncNet [137]	99.24	99.68
MLP2LN [60]	99.00	99.70
Cascade corr. [225]	98.50	100.00
BP+genetic [225]	98.40	99.40
Quickprop [225]	98.30	99.60
kNN+weights [66]	98.22	98.89
RProp [225]	98.00	99.60
kNN+weights [66]	97.96	98.89…

Tabela 6.2: Dokładność dla zbioru danych tarczycy.

Model	Test	Train
9NN CV3 Manh	88.72	88.75
19NN CV2 Manh	88.38	88.66
8NN CV2 Manh	88.26	88.87
11NN CV2 Manh	88.25	88.62
3NN CV3 Manh	85.65	91.21
9NN Manh	88.14	89.20
19NN Manh	87.95	88.01
8NN Manh	87.03	88.52
11NN Manh	87.65	88.33
1NN Manh	84.52	100.00
1NN	81.20	100.00
większość	80.19	-
IncNet [137]	90.90	90.11
kNN+weights [66]	87.38	88.20
kNN+weights [66]	87.13	88.55
FSM [1]	84.90	-
MLP+BP	83.90	-
RBF	80.20	- …

Tabela 6.3: Dokładności dla zbioru wyrostka robaczkowego.

Model	Test	Train
7NN CV3 Manh	86.49	89.28
17NN CV2 Manh	86.40	87.83
3NN CV3 Manh	85.61	91.42
1NN CV3 Manh	82.76	100.00
17NN Manh	86.32	87.51
17NN	85.57	86.61
7NN Manh	84.70	88.42
1NN	80.04	100.00
1NN Manh	79.50	100.00
większość	55.51	-
Cal5 [185]	86.90	86.80
ITrule [185]	86.30	-
SSV [111]	86.00	82.30
L. Disrim. [185]	85.90	86.10
RBF [185]	85.50	89.30
CART [185]	85.50	85.50
Naive Bayes [185]	84.90	86.40 …
IBL [257]	81.88	-
GIBL [257]	80.72	- …

Tabela 6.4: Dokładności dla zbioru *australian credit*.

Model	Test	Train
1NN CV3 Manh	62.04	100.0
3NN CV3 Manh	61.51	76.07
7NN CV3 Manh	58.30	68.28
27NN CV3 Manh	56.19	59.78
27NN Manh	52.56	56.53
7NN Manh	52.03	63.57
1NN Manh	50.91	100.00
17NN Manh	50.25	57.97
3NN Manh	48.07	72.53
27NN	46.76	51.75
1NN	48.52	100.00
większość	31.09	-
CART [272]	62.40	-
IB1 [272]	56.60	-
C 4.5 [272]	56.20	-
GIBL [257]	55.11	-
IBL [257]	48.29	- …

Tabela 6.5: Dokładności dla zbioru flagi.

Model	Test	Train
28NN CV5	84.21	84.29
17NN CV20	84.20	84.13
5NN CV3 Manh	83.27	87.83
3NN CV3 Manh	81.67	90.10
1NN CV3 Manh	79.30	100
17NN	83.92	85.39
28NN	83.72	84.31
3NN Manh	83.15	91.09
5NN Manh	82.51	87.48
1NN	76.54	100
1NN Manh	75.45	100
większość	54.13	-
28NN f. sel. [66]	85.1±0.5	-
LDA [236]	84.50	-
Fisher DA [236]	84.20	-
FSM [1]	82.4-84	84.00
Naive Bayes[236]	82.5-83.4	-
SNB [236]	83.10	-
LVQ [236]	82.90	-
SVM (5xCV) [10]	81.50	-
MLP+BP [236]	81.30	-
CART [236]	80.80	-
RBF	79.10	-
kNN+weights [66]	78.56	83.46
ASR [236]	78.40	-
C4.5 (5xCV) [10]	77.80	-
IB1c (WEKA)	77.60	-
kNN+weights [66]	76.88	82.32
QDA [236]	75.40	-

Tabela 6.6: Dokładności dla zbioru chorób serca.

Model	Test	Train
1NN CV3 Manh	80.81	100.00
3NN CV3 Manh	78.04	87.64
3NN CV3 Mink–0.7	77.97	88.78
3NN CV2 Manh	77.49	87.53
3NN CV2	75.00	85.11
1NN Manh	73.50	100.00
3NN Manh	72.06	84.98
3NN	70.85	83.63
1NN	70.45	100.00
większość	35.51	-
GIBL [257]	78.55	-
Bayes [272]	71.80	-
CART	71.40	-
IB1 [272]	71.10	-
ID3 [272]	69.10	-
IBL [257]	70.52	-

Tabela 6.7: Dokładności dla zbioru *glass*.

Model	Test	Train
3NN CV2 Manh	97.05	98.14
3NN CV20 Manh	96.98	98.04
5NN CV3 Manh	96.90	97.56
3NN CV3 Manh	96.79	98.26
1NN CV3 Manh	96.74	100.00
3NN Manh	96.98	98.23
3NN Mink–0.8	96.95	97.65
3NN	96.60	97.97
1NN Manh	96.20	100.00
1NN	95,18	100.00
większość	65.52	-
SVM (5xCV?)	97.20	-
IncNet [137]	97.10	-
kNN DVDM	97.10	-
SVM [10]	96.90	-
Fisher DA [236]	96.80	-
MLP+BP [236]	96.70	-
LVQ [236]	96.60	-
SNB [236]	96.60	-
FSM [1]	96.50	-
Naive Bayes [236]	96.40	- …

Tabela 6.8: Dokładności dla zbioru raka piersi.

Model	Test	Train
7NN CV3 Manh	97.35	98.25
1NN CV3 Manh	97.19	100.00
3NN CV3 Manh	96.79	98.67
1NN Manh	97.53	100.00
7NN Manh	97.31	97.79
AutoK CV3 Manh	97.23	98.43
3NN Manh	96.95	98.45
1NN	95.38	100.00
większość	39.89	-
Per [272]	98.30	-
BP [272]	98.30	-
GIBL [257]	96.60	-
IBL [257]	95.46	-
MML [272]	95.00	-
IB1 [272]	94.90	-
Bayes [272]	94.90	-
C 4.5 [272]	94.40	-
SMML [272]	94.40	-
ID3 [272]	93.80	-
IB2 [272]	93.20	-
IB3 [272]	91.50	-
CART [272]	87.60	-

Tabela 6.9: Dokładności dla zbioru *wine*.

użycia ważenia uzyskano poprawność 84.7% (porównaj tabela 6.5). Dla 17NN poprawa była nieco mniejsza: 86.4% z ważeniem cech i 86.32% bez ważenia cech. Lecz dla 1NN z miarą Manhattan postęp był większy: 82.76% z ważeniem i 79.5% bez ważenia.

6.3.1.3.4. Dane opisujące flagi narodowe (*Flags*) Zbiór *Flags* jest także problemem klasyfikacyjnym i zawiera 193 wektory, 28 cech, a każdy wektor może być przypisany do jednej z 8 klas. Algorytm ważenia cech na tym zbiorze zwiększył znacznie jakość poprawności. Rezultaty zostały przedstawione w tabeli 6.5. Dla 1NN CV3 z miarą Manhattan poprawność na części testowej wyniosła 62.04%, natomiast bez ważenia uzyskano tylko 50.91%. Dla 3NN z ważeniem uzyskano 61.51% i 48.07% bez ważenia. Najlepszy nie ważony model kNN osiągnął dokładność 52.56%, czyli około 10% mniej, niż najlepszy model kNN z ważeniem.

6.3.1.3.5. Dane raka piersi Dokładniejszy opis danych został umieszczony w rozdziale 7.2.2 (*Breast cancer Wisconsin*).

Na tym zbiorze danych użycie ważenia cech nie zwiększa poprawności klasyfikacji. Najlepszy ważony model uzyskał poprawność 97.05%, a najlepszy nie ważony model 96.98%. Rezultaty zamieszczono w tabeli 6.8.

6.3.1.3.6. Zbiór danych *Glass* Ten zbiór danych składa się z 214 wektorów, a każdy wektor ma 9 atrybutów. Problem jest 6 klasowy. W przypadku tego zbioru udało się znacznie podnieść poziom klasyfikacji po zastosowaniu algorytmu ważenia. Rezultaty zostały porównane w tabeli 6.7. Najlepszy model został uzyskany dla 1NN CV3 z miarą Manhattan i dał poprawność 80.81%. Natomiast bez ważenia uzyskano końcowo tylko 73.50%. Dla 3 sąsiadów (3NN CV3 z miarą Manhattan) ważony kNN uzyskał dokładność 78.04%, a bez ważenia dokładność 72.06%.

6.3.1.3.7. Dane chorób serca Dane chorób serca (*Cleveland heart disease*) składają się z 303 wektorów. Każdy z nich opisywany jest przez 13 cech. Wektor może być przypisany do jednej z dwóch klas. Dla modelu kNN z 28 sąsiadami ważona wersja pozwoliła uzyskać dokładność 84.21% na zbiorze testowym, a model bez ważenia uzyskał 83.72% (porównaj tabela 6.6). Z kolei dla 1NN z ważeniem dokładność wyniosła 79.3%, a bez ważenia model 1NN uzyskał dokładność 75.45% (oba dla miary Manhattan). Wartości dla części testowej i treningowej były bardzo zbliżone.

6.3.1.3.8. Dane opisujące gatunki win Zbiór danych *Wine* składa się z 178 wektorów opisywanych 13 cechami. Każdy wektor może być przypisany do jednej z 3 klas. Dla tego testu rezultaty uzyskane z ważeniem i bez, są bardzo podobne. Najlepszy model 7NN CV3 z miarą Manhattan uzyskał poprawność 97.35% na części testowej, a nieważony kNN uzyskał 97.31%. Dla 1NN, także z miarą Manhattan, ważona wersja uzyskała dokładność 97.19%, podczas gdy zwykła

minimalnie więcej 97.53%. Tu także wyniki na części testowej i treningowej dały bardzo podobną dokładność.

6.3.1.4. Podsumowanie

Dyskretna quasi-gradientowa metoda ważenia okazała się metodą efektywną i dającą nierzadko znaczące poprawy jakości klasyfikacji, czasem sięgające nawet ponad 10% dokładności. Jak pokazano, algorytm może być użyty do ekstrakcji cech.

Algorytm może startować z pełnej, bądź pustej bazy cech. Oznacza to, że może on rozpoczynać działanie od dodawania najistotniejszych wymiarów, bądź usuwania najmniej istotnych. Może to być bardzo przydatne, gdy mamy do czynienia z danymi wielowymiarowymi.

Ważna jest także możliwość uzyskania alternatywnych zestawów wag dla danego zbioru. Daje to dodatkowe informacje o wpływie poszczególnych cech lub ich grup na jakość informacji opisującej zbiór danych. Może też, na przykład, istotnie wpływać na dalszy dobór wykonywanych testów medycznych (część może okazać się bardziej istotna, a inne zbędne, bądź alternatywne).

Dzięki zastosowaniu algorytmu ważenia cech, algorytm kNN osiąga bardzo dobrą poprawność klasyfikacji, co obrazuje sporą skuteczność metody.

Warto zwrócić uwagę na zbliżone rezultaty na części testowej i treningowej, co najczęściej świadczy o zbliżeniu się do optymalnego modelu rozwiązującego dany problem w danej klasie modeli.

Ważne jest także, że algorytm może być użyty nie tylko do modelu k najbliższych sąsiadów, ale i do różnych innych modeli inteligencji obliczeniowej.

Taki sposób ważenia cech został zaimplementowany w systemie GhostMiner [139].

6.4. Regularyzacja danych

Ten rozdział poświęcony został zupełnie nowej technice, której celem jest automatyczne wyznaczanie błędów w danych, wskazywanie niepewnych danych i metodologia umożliwiająca naprawianie danych, bądź udostępnianie informacji o wiarygodności wektorów dla modeli adaptacyjnych tak, by mogły one uwzględnić wiarygodność wektora(-ów).

Dzięki temu modele adaptacyjne będą stabilniejsze podczas uczenia.

Inną cechą tej metody jest możliwość przypisania wartości ciągłych w miejsce numeru etykiety klas dla problemów klasyfikacyjnych. Umożliwia to odejście od danych *czarno-białych* (danych w których każdy wektor przypisany jest do jednej klasy) i uzyskanie odcieni szarości (każdemu wektorowi przypisujemy wiarygodność przynależności do danej klasy). Może to mieć bardzo pozytywny wpływ na wiele metod, szczególnie tych o charakterze aproksymatorów, które preferują uczenie się na ciągłych wartościach wyjść.

Metoda powinna też wpłynąć pozytywnie na uwiarygodnienie rezultatów osiągniętych przez modele po użyciu danych zregularyzowanych.

Dylematem modeli uczących się jest to, że najczęściej *każe* się im ufać, że dane na których przebiega proces uczenia są prawdziwe na 100%. Podczas gdy realne dane prawie zawsze takie nie są. Rozważmy bardzo prosty przypadek. Powiedzmy, że mamy dwie klasy *zdrowi* i *przeziębieni*. Przypisujemy poszczególnym wektorom jedną z etykiet klas. Oznajmiamy w ten sposób, że mamy tylko czarno-białe przypadki, a przecież wiele przypadków może reprezentować niejednoznaczne przypadki, takie jak nieco przeziębiony, prawie zdrowy.

Część metod radzi sobie z tym, używając różnych typów regularyzacji podczas procesu uczenia. Jak wiemy, metody regularyzacji najczęściej dodają pewien dodatkowy człon do funkcji błędu (porównaj rozdziały 2.1, 2.5.2 i 4.1.1) [209, 120, 253]. Nawet używając metod regularyzacji nie można być do końca pewnym, że problem zniknie. Jedną z przyczyn jest to, że regularyzacja (najczęściej) ma taką samą czułość w każdej części przestrzeni.

Co więcej, bardzo często nawet eksperci danego zagadnienia mają problem z określeniem wiarygodności, jaka jest związana z danym wektorem. Także wszystkie dobrze znane metody wstępnego przetwarzania danych nie próbują nawet rozwiązywać tego problemu.

Poniżej przedstawiona zostanie regularyzacja danych w różnych wariantach, które mogą być przydatne dla różnych typów metod. Schemat regularyzacji w naturalny sposób daje miarę wiarygodności, jaką można przypisać do oryginalnego zbioru danych. Pokazane zostaną także przykłady użycia regularyzacji.

Jak już dobrze wiadomo z poprzednich rozdziałów, celem klasyfikacji i aproksymacji jest poszukiwanie nieznanego odwzorowania:

$$f(\mathbf{x}_i) = y_i, \quad i = 1, 2, \ldots, N \tag{6.19}$$

dla danego zbioru danych $\mathcal{S}$:

$$\mathcal{S} = \{\langle \mathbf{x}_i, y_i \rangle : 1 \leq i \leq N\}, \tag{6.20}$$

gdzie każda para $\langle \mathbf{x}_i, y_i \rangle$ składa się z wektora wejściowego cech $\mathbf{x}_i$ i etykiety klasy, czy pewnej wartości wyjściowej y_i.

Czasami, dla pewnych klasyfikatorów, preferuje się możliwość reprezentacji etykiety klasy y_i przez wektor $\mathbf{v}_i$ z 1 na pozycji równej numerowi klasy y_i i reszcie równej 0 (na przykład dla wielowarstwowego perceptronu):

$$\mathbf{v}_i = [v_1, v_2, \ldots, v_d]^T \quad \text{and} \quad v_k = \begin{cases} 1 & k = y_i \\ 0 & k \neq y_i \end{cases}, \tag{6.21}$$

wtedy zbiór danych składa się z par wektorów:

$$\mathcal{S}^v = \{\langle \mathbf{x}_i, \mathbf{v}_i \rangle : 1 \leq i \leq N\}. \tag{6.22}$$

Bazując na zbiorze danych $\mathcal{S}$ można zdefiniować model $\mathcal{P}$, używając znormalizowanej funkcji gaussowskiej:

$$\bar{G}_i(\mathbf{x};\mathbf{x}_i) = \frac{G(\mathbf{x};\mathbf{x}_i,\sigma)}{\sum_{j=1}^{N} G(\mathbf{x};\mathbf{x}_j,\sigma)}, \tag{6.23}$$

gdzie $G(\mathbf{x};\mathbf{x}_i,\sigma)$ (σ jest pewną stałą) jest zdefiniowane przez

$$G(\mathbf{x};\mathbf{x}_i,\sigma) = e^{-\frac{||\mathbf{x}-\mathbf{x}_i||^2}{\sigma}}. \tag{6.24}$$

Wtedy model $\mathcal{P}$ może być zdefiniowany przez

$$P(k|\mathbf{x},\mathcal{S}) = \sum_{i\in I^k} \bar{G}_i(\mathbf{x};\mathbf{x}_i), \tag{6.25}$$

gdzie $I^k = \{i : \langle \mathbf{x}_i, y_i\rangle \in \mathcal{S} \wedge y_i = k\}$. Można zobaczyć, że

$$\sum_{i=1}^{K} P(i|\mathbf{x},\mathcal{S}) = 1, \tag{6.26}$$

K jest równe liczbie klas.

Wtedy $P(k|\mathbf{x},\mathcal{S})$ może być interpretowane jako prawdopodobieństwo, że dany wektor $\mathbf{x}$ należy do klasy k dla zbioru danych $\mathcal{S}$. Model taki można by nazwać **siecią znormalizowanych funkcji Gaussa** (*ang. normalized radial basis function network (NRBFN)*).

Parametr σ z równania 6.24 definiuje gładkość dla modelu $\mathcal{P}$. Zakładając, że σ jest wystarczająco mała, mamy:

$$P(y_i|\mathbf{x}_i,\mathcal{S}) \approx 1 \tag{6.27}$$

gdzie $\langle \mathbf{x}_i, y_i\rangle$ jest parą ze zbioru $\mathcal{S}$.

Załóżmy, że zbiór danych nie jest bardzo *wrażliwy* (jest dostatecznie gęsty). Wtedy usunięcie dowolnej pary ze zbioru $\mathcal{S}$ nie powinno zmieniać dramatycznie modelu $\mathcal{P}$.

Niech $\mathcal{S}^j$ będzie zbiorem $\mathcal{S}$ z usuniętą parą $\langle \mathbf{x}_j, y_j\rangle$:

$$\mathcal{S}^j = \{\langle \mathbf{x}_k, y_k\rangle : \langle \mathbf{x}_k, y_k\rangle \in \mathcal{S} \wedge k \neq j\}. \tag{6.28}$$

Teraz, używając prawdopodobieństwa

$$P(y_i|\mathbf{x}_i,\mathcal{S}^i) \tag{6.29}$$

określić można wiarygodność, że wektor $\mathbf{x}_i$ jest zgodny ze zbiorem $\mathcal{S}$. Może to być ***test zgodności*** lub ***test spójności*** dowolnego wektora ze zbiorem danych.

Współczynnik σ (6.24), który określa gładkość funkcji gaussowskiej, może być użyty do kontroli siły regularyzacji dla modelu $\mathcal{P}$. Wybór parametru σ

zależy od ufności apriorycznej dla danego zbioru $\mathcal{S}$ (jeśli tylko jest znana) lub może być wyznaczony przez D^2/N (D jest maksymalną odległością pomiędzy dwoma wektorami ze zbioru $\mathcal{S}$).

Test zgodności może być użyty na kilka sposobów w regularyzacji danych. Dwa typy regularyzacji zawarte są w poniższych zbiorach. Stanowią one rozszerzenia zbioru $\mathcal{S}$:

$$\mathcal{S}^P = \{\langle\langle \mathbf{x}_i, y_i\rangle, P(y_i|\mathbf{x}_i, \mathcal{S}^i)\rangle : 1 \leq i \leq N\}, \tag{6.30}$$

$$\mathcal{S}^{Pv} = \{\langle\langle \mathbf{x}_i, y_i\rangle, P(1|\mathbf{x}_i, \mathcal{S}^i), \ldots, P(K|\mathbf{x}_i, \mathcal{S}^i)\rangle : 1 \leq i \leq N\}. \tag{6.31}$$

Tak zdefiniowane zbiory niosą zmodyfikowaną informacje względem zbioru $\mathcal{S}$. W miejscu etykiety klasy y_i mamy prawdopodobieństwo $P(y_i|\mathbf{x}_i, \mathcal{S}^i)$, bądź (w przypadku drugiego zbioru) prawdopodobieństwa dla poszczególnych klas uzyskane z powyższego testu zgodności.

6.4.1. Odcienie szarości

Zbiór danych $\mathcal{S}$ składa się tylko z *czarnych* i *białych* przykładów. Teraz, bazując na powyższych zbiorach $\mathcal{S}^P$ i $\mathcal{S}^{Pv}$. Można stworzyć zbiór z danymi, które będą zawierały *odcienie szarości*:

$$\mathcal{S}^I = \{\langle \mathbf{x}_i, \langle y_i, P(y_i|\mathbf{x}_i, \mathcal{S}^i)\rangle\rangle : 1 \leq i \leq N\} \tag{6.32}$$

lub w wersji z wieloma wyjściami:

$$\mathcal{S}^{II} = \{\langle \mathbf{x}_i, \mathbf{p}_i\rangle : 1 \leq i \leq N\}, \tag{6.33}$$

gdzie

$$\mathbf{p}_i = [P(1|\mathbf{x}_i, \mathcal{S}^i), \ldots, P(K|\mathbf{x}_i, \mathcal{S}^i)]^T. \tag{6.34}$$

6.4.2. Eliminacja złych wektorów i przeetykietowanie klas.

Jest też możliwe, że dla niektórych wektorów $P(y_i|\mathbf{x}_i, \mathcal{S}^i)$ są istotnie mniejsze niż $P(j|\mathbf{x}_i$ ($j \neq y_i$). Oznacza to, że para $\langle \mathbf{x}_i, y_i\rangle$ jest niezgodna z oryginalnym zbiorem danych $\mathcal{S}$. Jednym z wyjść jest usunięcie takiego podejrzanego wektora ze zbioru danych $\mathcal{S}^I$ i $\mathcal{S}^{II}$ (6.32 i 6.33).

Innym wyjściem może być przeetykietowanie niepewnych wektorów, czyli przypisanie im nowych klas. Wtedy dla zbioru $\mathcal{S}^{II}$ każdy zły wektor $\mathbf{x}_i$ będzie przeetykietowany do bardziej prawdopodobnej klasy:

$$\max_{j \neq i} P(j|\mathbf{x}_i, \mathcal{S}^i). \tag{6.35}$$

W przypadku metod, które potrafią korzystać tylko z czarno-białych danych, informacja, która jest w zbiorach $\mathcal{S}^P$ i $\mathcal{S}^{Pv}$ może pomóc wykluczyć, bądź przeetykietować złe wektory ze zbioru $\mathcal{S}$. Dla przykładu można użyć zbioru $\mathcal{S}^{III}$ do przeetykietowania:

$$\mathcal{S}^{III} = \{\langle \mathbf{x}_i, k\rangle : 1 \leq i \leq N\}, \tag{6.36}$$

gdzie $k = \arg\max_j P(j|\mathbf{x}_i, \mathcal{S}^i)$.

Taka regularyzacja danych może zostać użyta do uczenia rożnych modeli sieci neuronowych (MLP, RBF, etc.), jak również może być użyta w różnych funkcjach kosztów innych modeli inteligencji obliczeniowej, do dodania w nich informacji o wiarygodności odpowiednich wektorów ze zbioru danych. Model adaptacyjny może wtedy brać pod uwagę informację o wiarygodności danego wektora.

6.4.3. Przykłady użycia regularyzacji danych

Poniżej zaprezentowano przykłady, które na prostych danych obrazują działanie regularyzacji przedstawionej powyżej.

Dane zostały wygenerowane dla dwóch klas niezależnie, z dwoma różnymi rozkładami gaussowskimi.

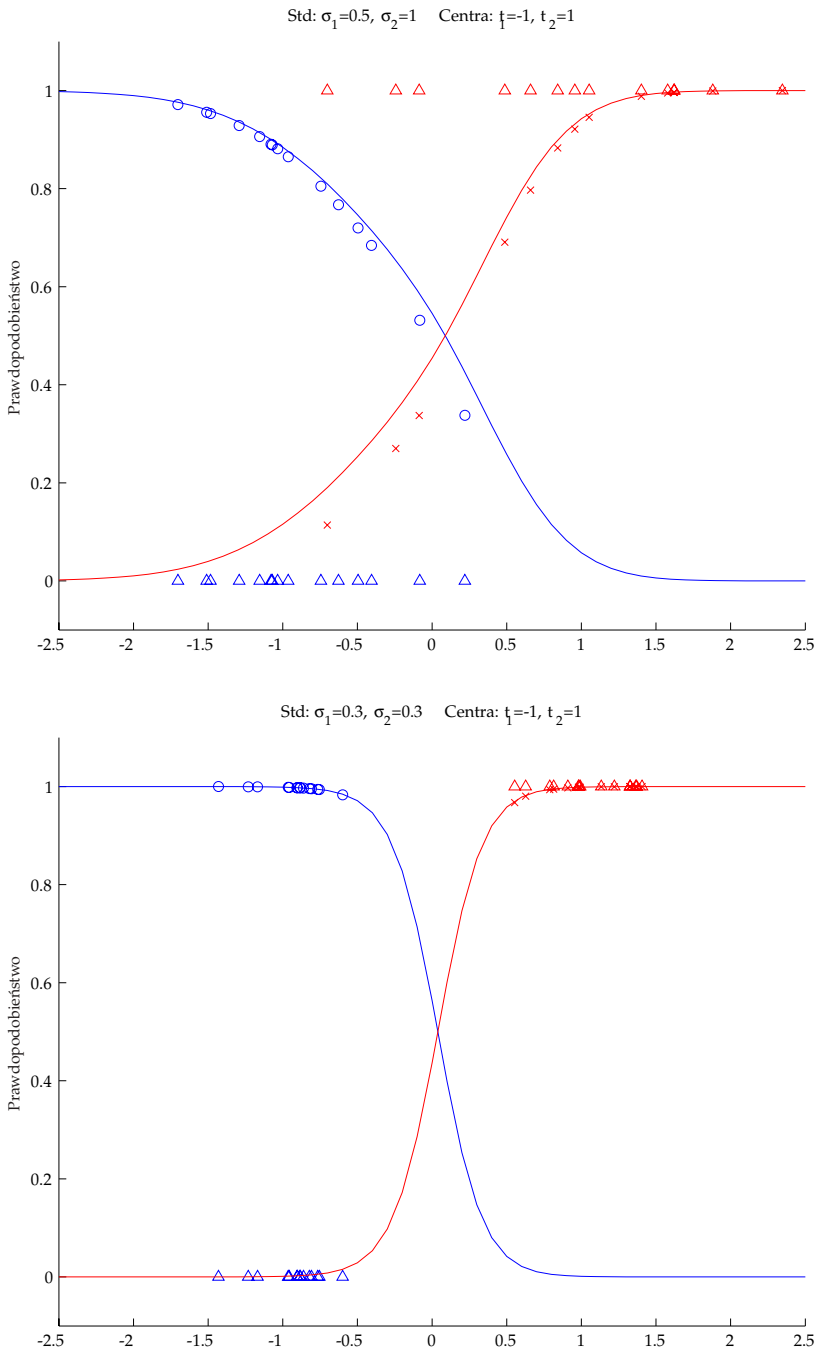
Rysunki 6.2 i 6.3 prezentują dane przed regularyzacją (trójkąty — dolny dla klasy I i górny dla klasy II) i po regularyzacji (kółka dla klasy I, a krzyżyki dla klasy II). Dwie ciągłe linie pokazują prawdopodobieństwo modelu $\mathcal{P}$ dla dwóch klas, zdefiniowane przez równanie 6.25 dla oryginalnych danych ze zbioru $\mathcal{S}$. Kolejne rysunki prezentują rezultaty dla innych dyspersji rozkładów i ich centrów.

6.4.4. Podsumowanie

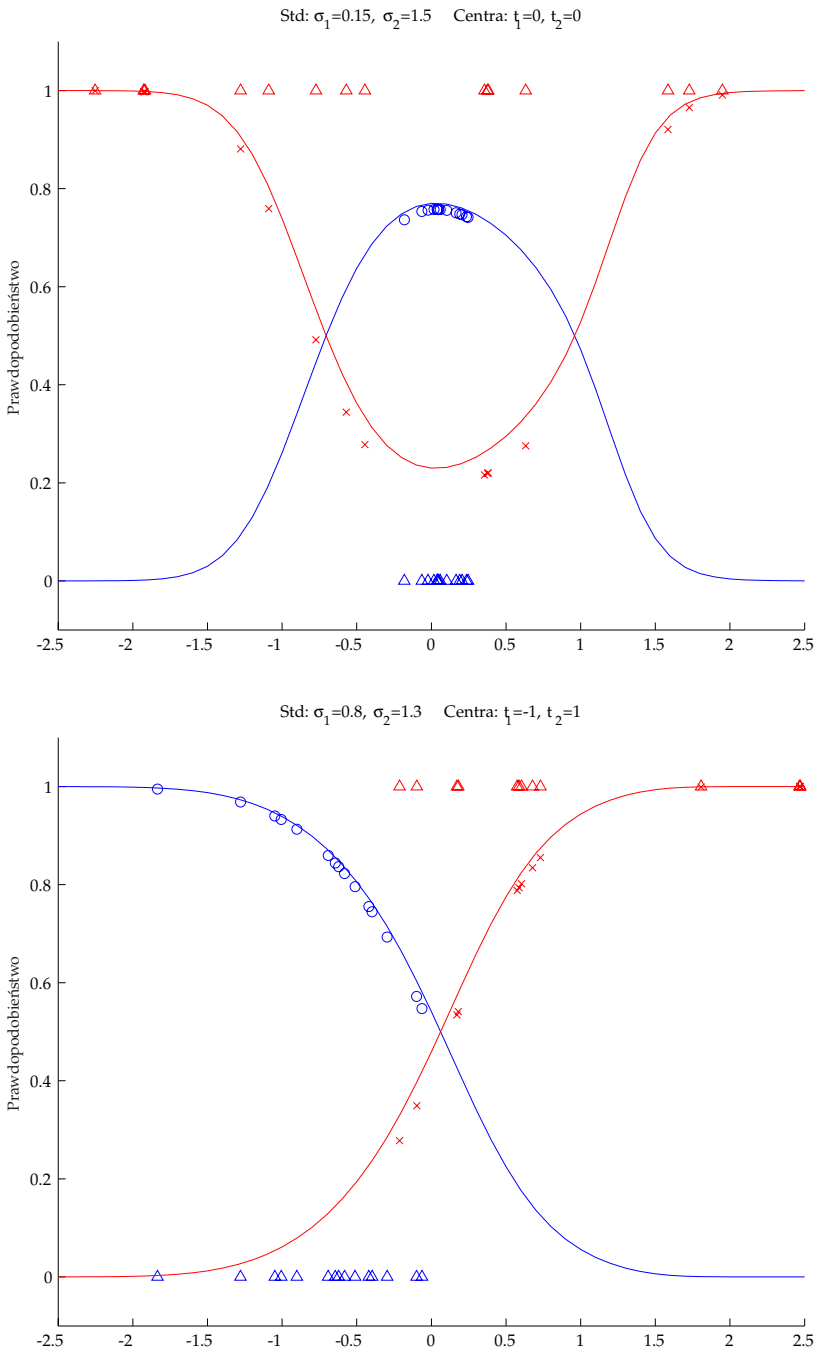
Opisana regularyzacja danych jest zupełnie nowym, bardzo ciekawym narzędziem, które może być stosowane do wielu różnych celów i może być używane z wieloma różnymi modelami inteligencji obliczeniowej. Może być stosowana do algorytmów uczenia, także gdy model wymaga ostrych regionów decyzyjnych, jak również do usuwania niepewnych, złych wektorów z oryginalnych danych. Można dokonać transformacji oryginalnego zbioru danych do zbioru z *odcieniami szarości*, dzięki czemu późniejszy model adaptacyjny może uwzględniać wiarygodność poszczególnych wektorów, co może prowadzić do stabilniejszego procesu uczenia.

6.5. Przedziały ufności, jako narzędzie analizy danych i wizualizacji wyników

Oprócz analizy istotnego podzbioru prawdopodobieństw $\{p(C^i|\mathbf{x}) : i = 1, \ldots, K\}$ dla najbardziej prawdopodobnej klasy można wyznaczyć w poszczególnych wymiarach wejściowych przedział, w którym zmienność wartości owego wymiaru nie zmieni klasyfikacji. Ściślej, zakładając, że wektor $\mathbf{x} = [x_1, x_2, \ldots, x_N]$ został



Rysunek 6.2: Regularyzacja danych I.



Rysunek 6.3: Regularyzacja danych II.

sklasyfikowany jako obiekt klasy k, przedział $[x^r_{min}, x^r_{max}]$ dla cechy r wyznaczony jest przez:

$$x^r_{min} = \min_{\bar{x}} \{C(\bar{\mathbf{x}}) = k \ \wedge \ \forall_{x_r > \hat{x} > \bar{x}} C(\hat{\mathbf{x}}) = k\} \tag{6.37}$$

$$x^r_{max} = \max_{\bar{x}} \{C(\bar{\mathbf{x}}) = k \ \wedge \ \forall_{x_r < \hat{x} < \bar{x}} C(\hat{\mathbf{x}}) = k\}, \tag{6.38}$$

gdzie $\bar{\mathbf{x}} = [x_1, \ldots, x_{r-1}, \bar{x}, x_{r+1}, \ldots, x_N]$, a $\hat{\mathbf{x}} = [x_1, \ldots, x_{r-1}, \hat{x}, x_{r+1}, \ldots, x_N]$.

Tak wyznaczone przedziały $[x^r_{min}, x^r_{max}]$ dla $r = 1, \ldots, N$, opisują możliwe odchylenia wartości dla poszczególnych współrzędnych r klasyfikowanego wektora $\mathbf{x}$, podczas gdy wartości pozostałych cech pozostają niezmienne (równe odpowiednim współrzędnym wektora $\mathbf{x}$). Dalej przedziały te będą nazywane *przedziałami ufności*.

Umiejscowienie wartości współrzędnych wektora $\mathbf{x}$ w odpowiadających im przedziałach ufności, pomaga stwierdzić, czy wektor $\mathbf{x}$ jest na obrzeżu regionu decyzyjnego, czy raczej w jego centrum.

Wyznaczanie przedziałów ufności można rozbudować o próg ufności, tak aby prawdopodobieństwo zwycięskiej klasy było istotnie większe od najbardziej prawdopodobnej klasy alternatywnej. Można to zrealizować, dodając nierówność do wzorów (6.37) i (6.38):

$$x^{r,\beta}_{min} = \min_{\bar{x}} \left\{ C(\bar{\mathbf{x}}) = k \ \wedge \ \forall_{x_r > \hat{x} > \bar{x}} C(\hat{\mathbf{x}}) = k \ \wedge \ \frac{p(C^k|\bar{\mathbf{x}})}{\max_{i \neq k} p(C^i|\bar{\mathbf{x}})} > \beta \right\} \tag{6.39}$$

$$x^{r,\beta}_{max} = \max_{\bar{x}} \left\{ C(\bar{\mathbf{x}}) = k \ \wedge \ \forall_{x_r < \hat{x} < \bar{x}} C(\hat{\mathbf{x}}) = k \ \wedge \ \frac{p(C^k|\bar{\mathbf{x}})}{\max_{i \neq k} p(C^i|\bar{\mathbf{x}})} > \beta \right\}, \tag{6.40}$$

gdzie $\bar{\mathbf{x}} = [x_1, \ldots, x_{r-1}, \bar{x}, x_{r+1}, \ldots, x_N]$, a $\hat{\mathbf{x}} = [x_1, \ldots, x_{r-1}, \hat{x}, x_{r+1}, \ldots, x_N]$. Współczynnik β oznacza wybrany próg. Oczywiście można prowadzić obserwacje zmienności przedziałów, w zależności, od doboru wartości progu β.

Tak zdefiniowane przedziały ufności stanowią silną alternatywę dla reguł logicznych i są narzędziem, które może znacząco wspomóc proces diagnozy, co będzie można prześledzić na poniższych przykładach.

Rysunek 6.4 ilustruje przykład wyznaczonych przedziałów ufności dla jednego przypadku z psychometrycznej bazy danych, szczegółowo opisywanych w podrozdziale 7.2.1. Przedstawione przedziały ufności opisują jednoznaczny przypadek psychozy reaktywnej, dla którego prawdopodobieństwo przynależności wyniosło 0.96. Kolejne podrysunki ilustrują dopuszczalne zmiany wartości poszczególnych cech.

Zielony kwadrat odpowiada wartości cechy analizowanego przypadku, a wartość odciętej to prawdopodobieństwo najbardziej prawdopodobnej z klas

(por. wzór 4.127). Linią ciągłą oznaczono dopuszczalny zakres wartości danej cechy dla najbardziej prawdopodobnej klasy. Wartość na osi pionowej odpowiada wartości wyznaczonego prawdopodobieństwa. Linią przerywaną oznaczono dopuszczalny zakres wartości danej cechy dla drugiej z najbardziej prawdopodobnych klas. Podobnie jak dla najbardziej prawdopodobnej klasy, wartość rzędnej odpowiada wartości wyznaczonego prawdopodobieństwa dla tej klasy. Ten przypadek jest jednak tak jednoznaczny, że zakres drugiej alternatywnej klasy po prostu niemal leży na osi rzędnych. Dzięki temu możemy nie tylko obserwować dopuszczalne zakresy zmian, ale widzimy również stosunek prawdopodobieństw najbardziej istotnych klas.

Dla odróżnienia rysunek 6.5 ilustruje przypadek, który nie jest tak jednoznaczny, jak poprzedni. W wyniku klasyfikacji okazało się, iż najbardziej prawdopodobną klasę stanowią zmiany organiczne 0.56, drugą najbardziej prawdopodobną klasą jest schizofrenia 0.39. Pozostałe klasy nie ma już istotnego wpływu (dla rozpatrywanego przypadku). Na podstawie tego przypadku zauważyć można znacznie bardziej istotny wpływ klasy alternatywnej, schizofrenii.

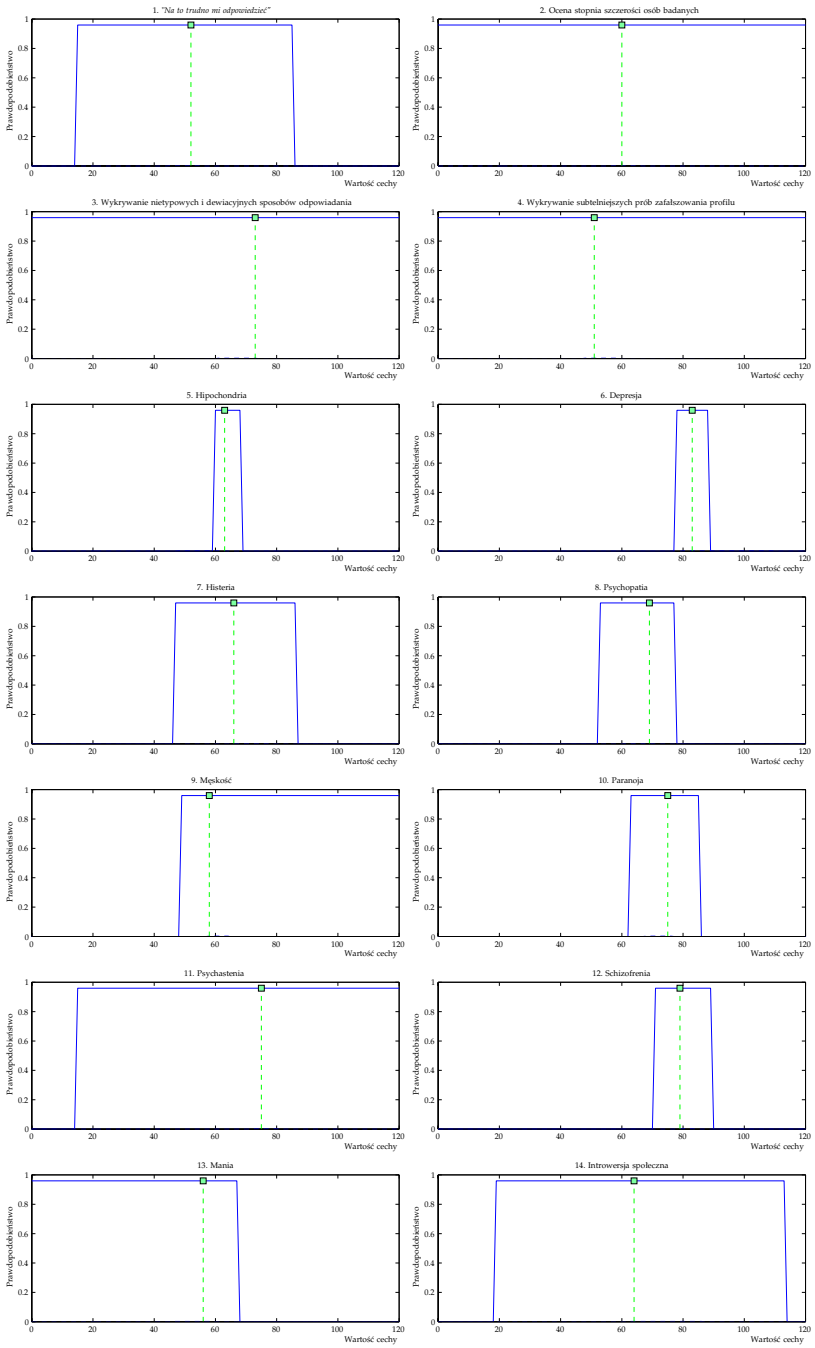
Jednak tak jak i reguły logiczne, przedziały ufności pokazują stałe prawdopodobieństwo podczas gdy wartości odpowiednich współrzędnych ulegają zmianie. Dlatego też znacznie bogatsze w informacje jest zilustrowanie nie tylko przedziałów ufności, ale również wartości prawdopodobieństw towarzyszących zmieniającym się wartościom poszczególnych cech. Takie przedziały będą nazywane *probabilistycznymi przedziałami ufności*. Dzięki takiej zmianie w miejsce *prostokątów*, które symbolizowały przedziały, pojawią się krzywe, które będą pokazywały zmianę prawdopodobieństwa.

Dla powyżej omówionych dwóch przypadków: psychozy reaktywnej i zmian organicznych, wyznaczone zostały probabilistyczne przedziały ufności — patrz rysunek 6.6 i 6.7.

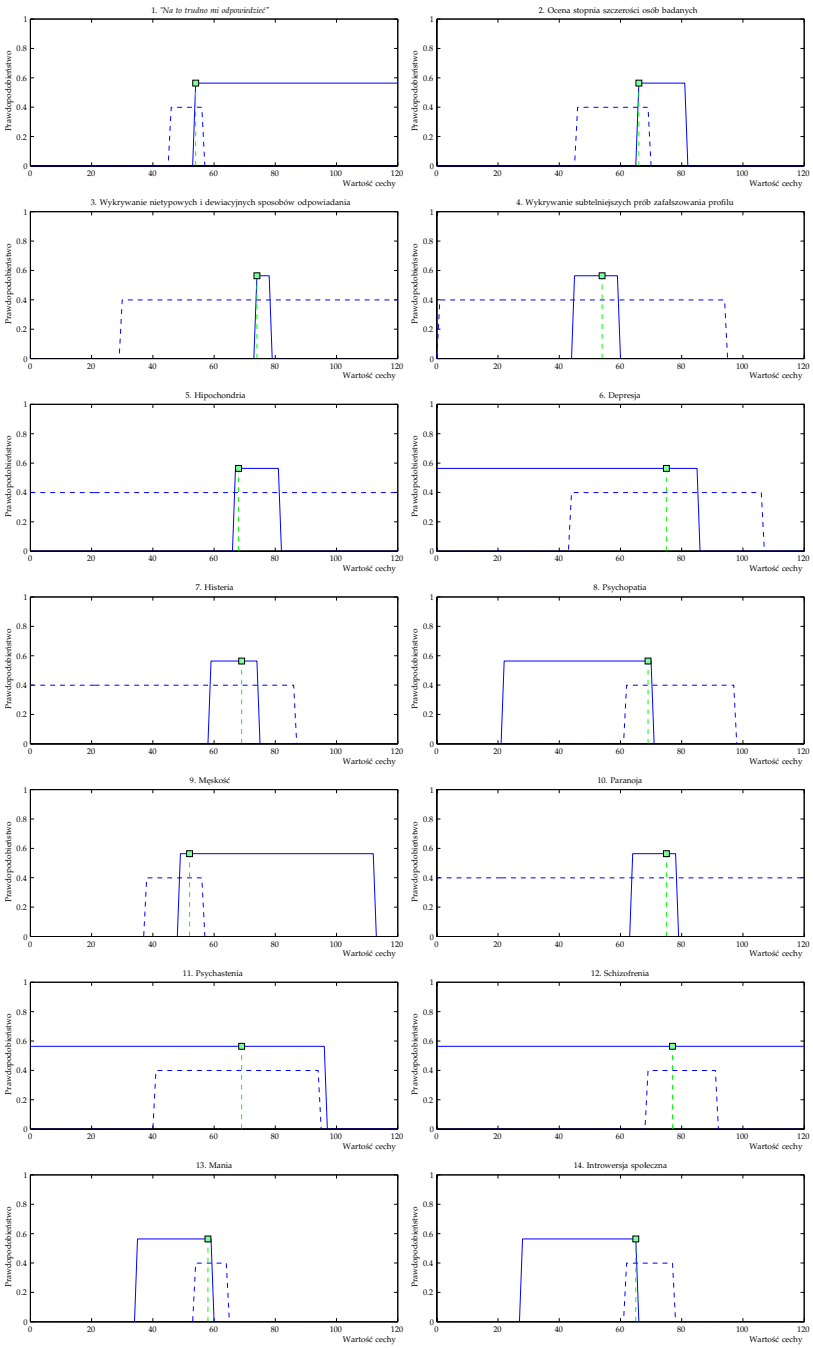
Zielony kwadrat odpowiada wartości cechy analizowanego przypadku, a wartość rzędnej to prawdopodobieństwo najbardziej prawdopodobnej z klas (por. wzór 4.127). Linią ciągłą oznaczono krzywą zmian wartości prawdopodobieństwa zwycięskiej klasy dla zmieniających się wartości odpowiedniej cechy analizowanego przypadku. Krzywa dla cechy r jest opisana przez prawdopodobieństwo $p(C(\mathbf{x})|\bar{\mathbf{x}})$ ($C(\mathbf{x})$ jest zdefiniowane równaniem 4.128), gdzie $\bar{\mathbf{x}} = [x_1, \ldots, x_{r-1}, \bar{x}, x_{r+1}, \ldots, x_N]$ Krzywa kropkowana przedstawia prawdopodobieństwo przynależności do drugiej z najbardziej prawdopodobnych klas rozpatrywanego przypadku dla zmieniających się wartości odpowiedniej cechy analizowanego przypadku. Krzywą dla cechy r opisuje prawdopodobieństwo przez $p(C^{k_2}|\bar{\mathbf{x}})$, gdzie k_2 jest zdefiniowane jako:

$$k_2 = \arg\max_i \{p(C^i|\mathbf{x}),\ C^i \neq C(\mathbf{x})\}. \tag{6.41}$$

Natomiast przerywana krzywa przedstawia prawdopodobieństwo klasy alternatywnej, najbardziej prawdopodobnej dla danej wartości prezentowanej cechy i pozostałych wartości cech zgodnych z rozpatrywanym przypadkiem (dla



Rysunek 6.4: Przedziały ufności. Przypadek psychozy reaktywnej.



Rysunek 6.5: Przedziały ufności. Przypadek zmian organicznych i schizofrenii.

różnych wartości prezentowanej klasy alternatywne mogą być różne). Tą krzywą opisuje prawdopodobieństwo $p(C^{k_M}|\bar{\mathbf{x}})$, gdzie k_M jest zdefiniowane jako:

$$k_M = \arg\max_i \{p(C^i|\bar{\mathbf{x}}),\ C^i \neq C(\mathbf{x})\}. \tag{6.42}$$

Gdy krzywa składa się z kropek i odcinków oznacza to nakładanie się dwóch opisanych powyżej krzywych.

Proszę zwrócić uwagę, iż w równaniu 6.42 wyznacza się maksimum z $p(C^i|\bar{\mathbf{x}})$ w punkcie $\bar{\mathbf{x}}$, natomiast w równaniu 6.41 w punkcie $\mathbf{x}$.

Więcej przykładów probabilistycznych przedziałów ufności można znaleźć w podrozdziale 7.2.1 (rysunki 7.13, 7.14, 7.15, 7.16, 7.17 i 7.18).

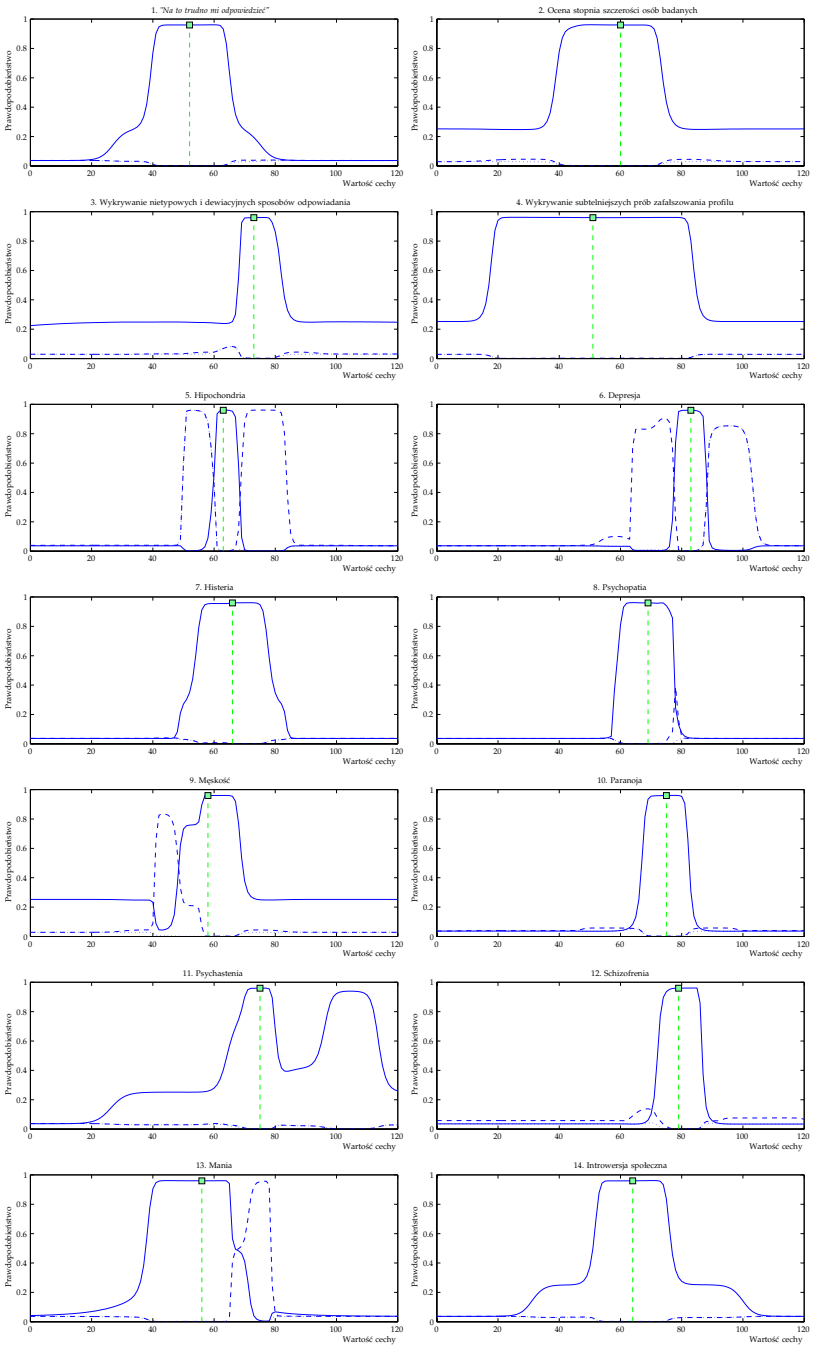
Probabilistyczne przedziały ufności znacznie wzbogacają informację, wspomagając proces klasyfikacji i diagnozy. Analizując zmienność prawdopodobieństwa dla poszczególnych cech łatwo można znaleźć wiele istotnych własności, charakterystycznych dla rozpatrywanego przypadku. Przede wszystkim poprzez analizę wpływu klas alternatywnych (krzywa przerywana) można zbadać, na ile proces klasyfikacji jest jednoznaczny i stabilny ze względu na małe zmiany wartości cech. To wydaje się najważniejszą kwestią nie tylko w medycynie. Patrząc na umiejscowienie rozpatrywanego przypadku w wyznaczonych przedziałach ufności można stwierdzić na ile jest on typowy. Z kolei obserwując charakter zmian zwycięskiej klasy można określić, czy ma ona wpływ na dany przypadek, a jeśli tak, na ile jest on istotny.

6.5.1. Przedziały ufności i probabilistyczne przedziały ufności, a reguły logiczne.

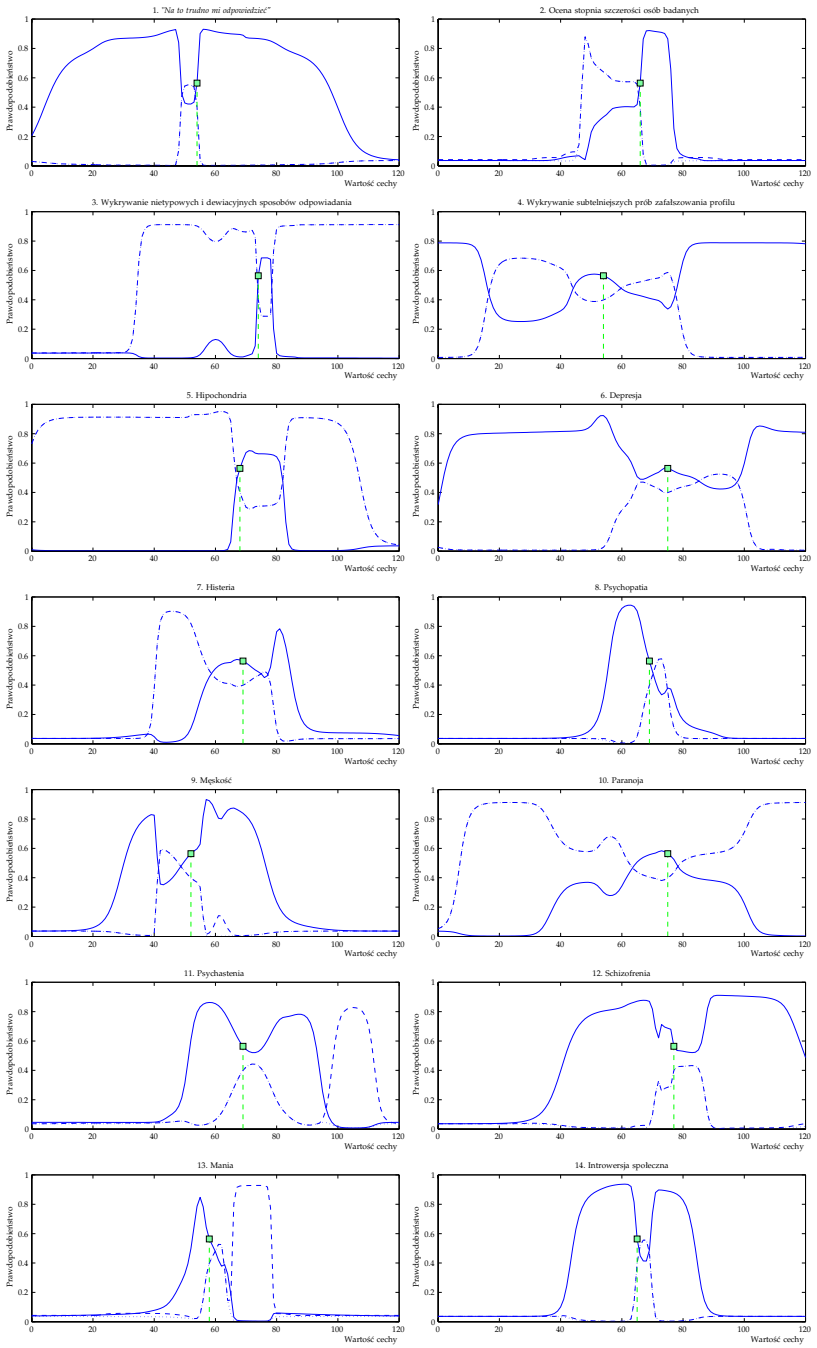
Najbardziej widoczną i znaczącą różnicą jest rodzaj interpretacji: probabilistyczny dla przedziałów ufności i logiczny dla reguł logicznych. W konsekwencji tego odpowiedź w analizie danego przypadku przez reguły to *tak* lub *nie*, nie ma wartości pośrednich. W realnych zastosowaniach, szczególnie w medycynie, taki brak wartości pośrednich znacznie obniża zaufanie do sugerowanej diagnozy. Z kolei, gdy dochodzi do analizy trudnych przypadków, nierzadko napotyka się na wektory, które znajdują się w obszarze więcej niż jednej reguły, które należą do różnych klas lub też okazuje się, że w ogóle żadna z reguł nie pokrywa obszaru, w którym znajduje się analizowany przypadek (por. [184]).

Analiza probabilistycznych przedziałów ufności dostarcza informacji o prawdopodobieństwach najistotniejszych klas, gładko opisując wszelkie ich zmiany. Umożliwia to dogłębną analizę danego przypadku, niezależnie w każdym wymiarze. Prawdopodobieństwa umożliwiają dokładne porównanie różnicy pomiędzy klasą najbardziej prawdopodobną, a klasą lub klasami alternatywnymi — każdy przypadek może mieć choćby minimalnie inny rozkład prawdopodobieństw przynależności do poszczególnych klas.

Wszystko to sprawia, że probabilistyczne przedziały ufności stanowią niezwykle silne narzędzie wspomagania procesu klasyfikacji, jego analizy i wizu-



Rysunek 6.6: Probabilistyczne przedziały ufności. Przypadek psychozy reaktywnej.



Rysunek 6.7: Probabilistyczne przedziały ufności. Przypadek zmian organicznych i schizofrenii.

alizacji. W efekcie czego trudno już taki model postępowania nazwać *czarną skrzynką,* co nierzadko zarzuca się metodom sztucznych sieci neuronowych.

Rozdział 7

Zastosowanie sieci neuronowych do klasyfikacji i analizy danych medycznych i aproksymacji

7.1. Techniki porównywania różnych modeli

Rozmaite modele adaptacyjne, które uczą się z danych, cechują się przeróżnymi właściwościami. Część możliwości porównywania modeli, to analityczne możliwości porównywania podobieństw i różnic matematycznych właściwości poszczególnych modeli. Jednak najczęściej nie jest to wystarczające, co jest spowodowane zbyt ogólnym poziomem takich analiz. Dlatego też najczęstszym sposobem porównywania różnych modeli jest porównywanie błędów, jakie te modele popełniają już po procesie adaptacji. Należy jednak pamiętać, że istnieją dwa źródła błędów modelu. Pierwsze to niedoskonałość samego modelu adaptacyjnego, drugie to niedoskonałość danych.

$$\text{Całkowity Błąd Modelu} = \text{Niedoskonałość Modelu} + \text{Niedoskonałość Danych}$$

Dodatkowo ważnym jest rozróżnienie dwóch typów błędów: *błędu uczenia* i *błędu generalizacji*. Pierwszy, to błąd, który model popełnia na zbiorze, na którym był uczony. Drugi, błąd generalizacji, to błąd, jaki popełnia model na zbiorze testowym, którego elementy nie brały udziału w procesie uczenia. Obserwacja błędu uczenia pokazuje nam postęp (lub jego brak) w procesie uczenia. Końcowa wartość błędu uczenia nie jest niestety wymiernym współczynnikiem

umożliwiającym określenie, tego, co najważniejsze, czyli uzyskanego poziomu generalizacji. Z analizy wielu artykułów wynika, że nadal wiele osób lekceważy ten fakt. Istotnie więcej na temat poziomu generalizacji uzyskuje się z obserwacji błędu uzyskanego na zbiorze testowym.

Skąd bierze się zbiór testowy? Mamy tu dwa główne przypadki:

- gdy mamy do czynienia z pewnym odgórnie zdefiniowanym podziałem zbioru danych na część treningową i testową,
- gdy nie istnieje żaden odgórny podział zbioru danych.

O ile pierwszy przypadek nie pozostawia żadnych wątpliwości, to drugi staje się najczęściej przyczyną powstawania przeróżnych sposobów podziału zbioru danych na część (części) treningową i testową. To z kolei nierzadko jest powodem niemożności dalszego porównywania błędów różnych modeli. Najczęściej powodem uniemożliwiającym dalsze porównywanie wyników jest dysproporcja pomiędzy liczebnością zbioru treningowego i testowego w różnych podziałach.

Istnieje jednak parę standardowych sposobów podziału zbioru danych i wyznaczania dzięki temu poszczególnych błędów.

Pierwszym i najprostszym sposobem jest losowy podział na część testową i treningową. Najczęściej dokonuje się podziału w następujących proporcjach: 90% i 10%, odpowiednio dla zbioru treningowego i testowego; 80% i 20%; 70% i 30%; a czasem i 50% na 50%. W związku z ogromną liczbą możliwych podziałów, wynikających z losowości podziału, należy w praktyce dokonać uśrednienia po wielu takich podziałach, aby uniezależnić wyniki badań od owego losowego podziału. Wtedy, jako błąd, podaje się wartość średnią (odpowiednio dla zbioru treningowego i testowego).

Drugi sposób podziału zbioru danych najczęściej nazywany jest walidacją skośną (*ang. crossvalidation (CV)*), czasem też kroswalidacją, czy testem krzyżowym. Test ten polega na losowym podziale zbioru danych $\mathcal{S}$ na k możliwie równo licznych podzbiorów $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$. Następnie na podstawie dokonanego podziału tworzy się k par zbiorów treningowych i testowych:

$$TRS_i = \bigcup_{j \neq i} \mathcal{S}_j, \tag{7.1}$$

$$TES_i = \mathcal{S}_i, \tag{7.2}$$

dla $i = 1, 2, \ldots, k$.

Kolejnym krokiem jest użycie powyżej zdefiniowanych par zbiorów, kolejno do uczenia i wyznaczenia błędów treningowych i testowych. Ostatecznie umoż-

liwia to wyznaczenie końcowych rezultatów:

$$E_{TRS} = \frac{1}{k}\sum_{i=1}^{k} E_{TRS_i}, \tag{7.3}$$

$$E_{TES} = \frac{1}{k}\sum_{i=1}^{k} E_{TES_i}, \tag{7.4}$$

$$\tag{7.5}$$

gdzie E_{TRS_i} jest błędem modelu uzyskanym po uczeniu na zbiorze treningowym, a E_{TES_i} jest błędem modelu, uzyskanym po uczeniu na zbiorze testowym.

Do najbardziej typowych podziałów należy podział na 10 podzbiorów (10 CV) i na n podzbiorów, gdzie n oznacza liczbę wektorów danych. Ostatni przypadek nazywany jest *leave one out* (LOO).

Czasem spotyka się również odmianę testu krzyżowego, która statystycznie powinna dawać wierniejsze rezultaty od wyżej opisanej metody. Metoda ta dotyczy danych, na których dokonuje się klasyfikacji. Jedyna różnica polega na dodatkowym warunku nakazującym utrzymywanie we wszystkich podzbiorach takich samych proporcji elementów poszczególnych klas do liczby pozostałych elementów, jakie są w całym zbiorze danych. Nazywa się ją walidacją skośną stratyfikowaną (*ang. stratified crossvalidation*), SCV.

Choć czasem różnice pomiędzy wynikami z 10 CV i LOO są nieduże, to na ogół jednak różnica pomiędzy nimi jest statystycznie istotna.

Gdy wartości 10 CV i LOO są zbliżone, może to oznaczać, że zbiór danych dobrze opisuje problem i tym samym usuniecie z niego 10% danych nie powoduje istotnych błędów wynikających z niereprezentatywności próbki w stosunku do (nieznanego) rozkładu prawdopodobieństwa. Prawdą jest również, że bardziej wartościowy jest *dobry* rezultat 10 CV niż rezultat z taka samą wartością dla LOO. Z tego wynika fakt, iż kiedy model A ma taki rezultat dla 10 CV jak model B dla LOO, to z pewnością model A nie powinien być gorszy. Podobne konkluzje znaleźć można w [24].

W problemach klasyfikacji jako miary błędu używa się współczynnika poprawności

$$WP = \frac{\text{ilość poprawnie sklasyfikowanych wektorów}}{\text{ilość wektorów}} \tag{7.6}$$

lub też błędu klasyfikacji

$$WB = 1 - WP = \frac{\text{ilość niepoprawnie sklasyfikowanych wektorów}}{\text{ilość wektorów}}. \tag{7.7}$$

Sumaryczny błąd kwadratowy (*ang. Sum Squared Error, SSE*), jak i pozostałe miary wymienione poniżej, są częściej wykorzystywane w aproksymacji, niż klasyfikacji:

$$SSE = \sum_{i=1}^{n}(F(\mathbf{x}_i) - y_i)^2. \tag{7.8}$$

Jeszcze częściej korzysta się ze średniego błędu kwadratowego (*ang. Mean Squared Error, MSE*):

$$MSE = \frac{1}{n}SSE = \frac{1}{n}\sum_{i=1}^{n}(F(\mathbf{x}_i) - y_i)^2, \tag{7.9}$$

czy też pierwiastka średniego błędu kwadratowego (*ang. Root Mean Squared Error, RMSE*):

$$RMSE = \sqrt{MSE} = \frac{1}{\sqrt{n}}\sqrt{SSE} = \frac{1}{\sqrt{n}}\sqrt{\sum_{i=1}^{n}(F(\mathbf{x}_i) - y_i)^2}. \tag{7.10}$$

Inna stosowaną miarą jest średni błąd procentowy (*ang. Average Percentage Error, APE*):

$$APE = \frac{1}{N}\sum_{i=1}^{n}\left|\frac{F(\mathbf{x}_i) - y_i}{y_i}\right| * 100\%. \tag{7.11}$$

Czasami spotyka się również miarę, która uwzględnia nie tylko błąd, ale i wariancję (*ang. Average Relative Variance, ARV*)

$$ARV = \frac{\sum_{i=1}^{n}(F(\mathbf{x}_i) - y_i)^2}{\sum_{i=1}^{n}(\bar{y} - y_i)^2} \tag{7.12}$$

gdzie $\bar{y} = 1/n\sum_{i=1}^{n} y_i$.

7.2. Medyczne zastosowania sieci IncNet

Sieć IncNet można stosować zarówno do problemów klasyfikacyjnych, jak i w aproksymacji. W bieżącym podrozdziale przedstawione zostaną zastosowania sieci IncNet do klasyfikacji danych medycznych. Pierwszym zastosowaniem będzie klasyfikacja danych psychometrycznych. Zadaniem sieci w tym przypadku jest klasyfikacja osoby do pewnej psychiatrycznej grupy nozologicznej, w oparciu o wyznaczone współczynniki dla danej osoby. W następnym podrozdziale zaprezentowane jest użycie sieci IncNet do klasyfikacji chorób: raka piersi, zapalenia wątroby, cukrzycy, zapalenia wyrostka i chorób tarczycy. Natomiast w kolejnym podrozdziale będzie można prześledzić kilka przykładów zastosowania sieci IncNet w aproksymacji.

7.2.1. Klasyfikacja i analiza danych psychometrycznych

7.2.1.1. Opis problemu

Psychometryczny test *Minnesota Multiphasic Personality Inventory (MMPI)* [34, 32, 33, 9] jest jednym z najczęściej stosowanych testów, które wspomagają dokonywanie klasyfikacji psychiatrycznych typów nozologicznych. Test MMPI składa

się z ponad 550 pytań. Pytania testu dotyczą przeróżnych tematów, związanych z badaną osobą [72] (liczby w nawiasach oznaczają liczbę pytań): ogólnego stanu zdrowia (9 pozycji), symptomów neurologicznych (19), nerwów czaszkowych (11), motoryki i koordynacji ruchowej (6), wrażliwości (5), reakcji wazomotorycznych, zaburzeń mowy, problemów wydzielniczych (10), problemów systemu krążeniowo-oddechowego (5), problemów żołądkowo-jelitowych (11), problemów moczowo-płciowych (5), nawyków (19), spraw rodzinnych i małżeńskich (26), problemów zawodowych (18), problemów szkolnych (12), postaw wobec religii (19), postaw politycznych, stosunku do prawa i porządku (46), postaw społecznych (72), obniżenia nastroju (32), podwyższenia nastroju (24), stanów obsesyjnych i kompulsywnych (15), urojeń, poczucia mocy, halucynacji, iluzji (34), fobii (29), tendencji sadystycznych i/lub masochistycznych (7), morale (33), pozycje odnoszące się do męskości-kobiecości (55) pozycje wskazujące na to, czy jednostka nie próbowała przedstawić siebie w nadmiernie korzystnym świetle (15).

Na podstawie odpowiedzi na pytania testu konstruuje się skale kontrolne i kliniczne.

Na skale kontrolne składają się następujące elementy: ”Na to trudno mi odpowiedzieć” (”?”), ocena stopnia szczerości osób badanych, wykrywanie nietypowych i dewiacyjnych sposobów odpowiadania, wykrywanie subtelniejszych prób zafałszowania profilu.

Z kolei w skład skal klinicznych wchodzą: hipochondria, depresja, histeria, psychopatia męskość, paranoja, psychastenia, schizofrenia, mania, introwersja społeczna.

Celem testu MMPI jest, na podstawie wyżej przedstawionych cech (w postaci współczynników różnych skal), wspomożenie dokonania klasyfikacji psychiatrycznego typu nozologicznego badanej osoby. Część spośród typów jest wspólna dla kobiet i mężczyzn, natomiast inne typy są zróżnicowane. Jeden z możliwych podziałów dokonany przez J. Gomułę i T. Kucharskiego (Uniwersytet M. Kopernika w Toruniu) oddzielny dla kobiet i mężczyzn przedstawiony jest poniżej.

Typy dotyczące kobiet: nerwica, psychopatia, przestępcy, schizofrenia, psychozy reaktywne, psychozy inwolucyjne, symulacja, dewiacyjne style odpowiedzi (grupa składająca się z 6 klas nozologicznych).

Typy dotyczące mężczyzn: nerwica, psychopatia, alkoholizm, przestępcy, schizofrenia, psychozy reaktywne, symulacja, dewiacyjne style odpowiedzi (grupa składające się z 6 klas nozologicznych).

Typy wspólne: norma, psychopatia, narkomania, organika, zespół urojeniowy, psychozy reaktywne, paranoja, stan hipomaniakalny, symulacja, dyssymulacja.

7.2.1.2. Dane

Ostateczna klasyfikacja typu nozologicznego na podstawie skal kontrolnych i klinicznych jest trudna i wymaga bogatej wiedzy specjalistycznej. Powstało więc

pytanie, czy nie można by skonstruować systemu, który mógłby dokonywać automatycznie właściwej klasyfikacji, bazując na wyznaczonych skalach (kontrolnych i klinicznych). W tym celu psycholodzy z Uniwersytetu Mikołaja Kopernika, Jerzy Gomuła i Tomasz Kucharski, opracowali bazy danych w oparciu o liczną grupę pacjentów Akademickiej Poradni Psychologicznej. Bazy te zostały uzupełnione informacje z kilku szpitali psychiatrycznych. Starano się przy tym dobierać odpowiednio liczne grupy osób dla różnych typów nozologicznych. Przestrzegano również różnych ograniczeń, wypływających z założeń przeprowadzania testu MMPI (tj. odpowiedni wiek, nie mniej niż podstawowe wykształcenie, dobry ogólny stan zdrowia). Starano się również, aby zbliżone były do siebie rozkłady związane z takimi zmiennymi jak płeć, wiek, wykształcenie stan cywilny, środowisko, czas trwania choroby oraz charakteru leczenia.

W efekcie powstało kilka baz, które ciągle są rozbudowywane. W poniższych badaniach będą analizowane główne dwie bazy. Każda z nich ma 14 cech, na które składają się skale kontrolne i kliniczne (patrz powyższy opis). Pierwsza baza dotyczy kobiet, a druga mężczyzn. Obie baza zawierają klasy (podklasy) wspólne. Takie klasy oznaczone są poprzez dodanie *–w*. Natomiast klasy kobiet i mężczyzn oznaczone są poprzez dodanie *–k* i *–m* odpowiednio dla kobiet i mężczyzn.

Pierwsza baza składa się z 1027 wektorów, z których każdy może należeć do jednej z 27 klas: norma-w (1), nerwica-w (2), psychopatia-w (3), organika-w (4), schizofrenia-w (5), zespół urojeniowy-w (6), psychoza reaktywna-w (7), psychoza inwolucyjna-w (8), paranoja-w (9), stan (hipo)maniakalny-w (10), przestępcy-w (11), symulacja-w (12), dysymulacja-w (13), narkomania-w (14), norma-k (15), przestępcy-k (16), nerwica-k (17), psychopatia-k (18), organika-k (19), schizofrenia-k (20), symulacja-k (21), dewiacyjny styl odpowiedzi 1-k (22), dewiacyjny styl odpowiedzi 2-k (23), dewiacyjny styl odpowiedzi 3-k (24), dewiacyjny styl odpowiedzi 4-k (25), dewiacyjny styl odpowiedzi 5-k (26), dewiacyjny styl odpowiedzi 6-k(27).

Druga baza składa się z 1167 wektorów, z których każdy może należeć do jednej z 28 klas: norma-w (1), nerwica-w (2), psychopatia-w (3), organika-w (4), schizofrenia-w (5), zespół urojeniowy-w (6), psychoza reaktywna-w (7), psychoza inwolucyjna-w (8), paranoja-w (9), stan (hipo)maniakalny-w (10), przestępcy-w (11), symulacja-w (12), dysymulacja-w (13), narkomania-w (14), norma-m (15), przestępcy-m (16), nerwica-m (17), psychopatia-m (18), alkoholizm-m (19), organika-m (20), schizofrenia-m (21), symulacja-m (22), dewiacyjny styl odpowiedzi 1-m (23), dewiacyjny styl odpowiedzi 2-m (24), dewiacyjny styl odpowiedzi 3-m (25), dewiacyjny styl odpowiedzi 4-m (26), dewiacyjny styl odpowiedzi 5-m (27), dewiacyjny styl odpowiedzi 6-m (28).

Zawartość baz przedstawiono graficznie na rysunkach .3 i .4 dla pierwszej bazy (strony 307 i 308), natomiast na rysunkach .5 i .6 dla drugiej bazy (strony 309 i 310). Dla każdej z cech różnokolorowe kolumny punktów (horyzontalnie nieco przesunięte względem siebie) odpowiadają różnym klasom nozologicznym.

7.2.1.3. Proces uczenia

Jak opisano w podrozdziale 4.3.7, sieć IncNet wykorzystywana do problemów klasyfikacji, składa się z klastra podsieci, a zadaniem każdej z podsieci jest estymacja każdej z klas niezależnie, po czym ostatecznej klasyfikacji dokonuje moduł decyzyjny (który działa w oparciu o zasadę, że zwycięzca bierze wszystko), co ilustruje rys. 4.12. Należy wspomnieć również, że każda z sieci, ucząc się niezależnie, wyznacza w procesie uczenia jak najlepszą dla siebie architekturę, korzystając z mechanizmów kontroli złożoności, dzięki czemu podsieci najczęściej znacznie różnią się pod względem końcowej liczby neuronów (funkcji bazowych).

Poniższe tabele (7.1 i 7.2) prezentują, jak rozkłada się liczba neuronów sieci IncNet w poszczególnych podsieciach. Tabele zawierają informacje uzyskane na podstawie uczenia na zbiorze 27 i 28 klasowym odpowiednio. Proces uczenia trwał 5 epok.

Liczby neuronów w poszczególnych podsieciach																										
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
5	1	4	9	9	6	4	3	8	8	3	11	4	1	8	4	5	8	12	8	1	2	2	1	1	1	1
Całkowita liczba neuronów: 130																										

Tabela 7.1: Rozkład złożoności sieci IncNet dla zbioru 27 klasowego.

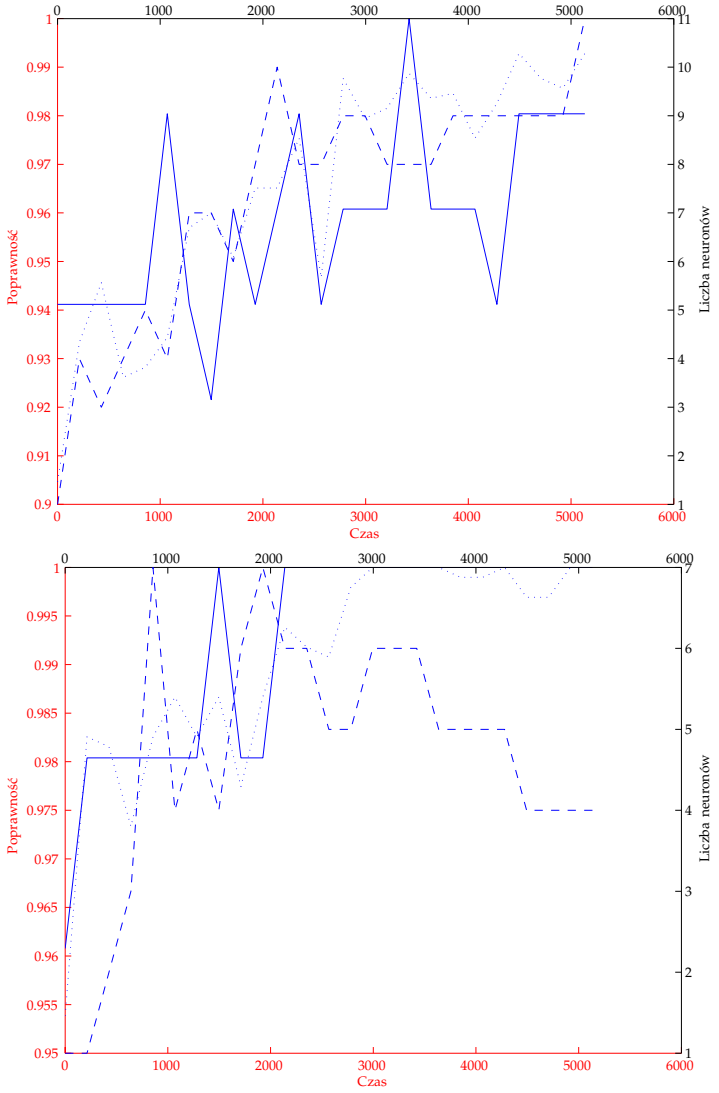
Liczby neuronów w poszczególnych podsieciach																												
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
4	5	4	13	10	2	5	4	7	6	2	15	1	1	6	4	12	6	13	11	11	14	1	1	1	1	1	1
Całkowita liczba neuronów: 162																												

Tabela 7.2: Rozkład złożoności sieci IncNet dla zbioru 28 klasowego.

Jak widać, złożoność poszczególnych podsieci jest znacznie zróżnicowana i waha się od 1 neuronu do 15 neuronów. Dowodzi to, że kontrola złożoności powinna być wbudowana w mechanizm uczenia i działać możliwie sprawnie.

Zmienność liczby neuronów w procesie uczenia, jak i zmianę wartości błędu treningowego i testowego, dla kilku wybranych podsieci, można przeanalizować na kolejnych rysunkach. Widzimy na nich zmiany, które zostały zebrane w 25 punktach kontrolnych (czyli co około 200 iteracji, każda sieć była uczona 5 epok).

Rysunek 7.1 przedstawia przykładowy proces uczenia dla 5-tej i 16-tej klasy 27-klasowej bazy danych. Należy zwrócić uwagę, że jednostką czasu jest tu jedna iteracja, czyli prezentacja jednego wektora treningowego. Czarna krzywa obrazuje liczbę neuronów, czerwona i zielona pokazują poprawność klasyfikacji dla zbioru treningowego i testowego.



Rysunek 7.1: Wykres ilustruje zmieniającą się w czasie poprawność klasyfikacji dla zbioru treningowego (linia kropkowana) i zbioru testowego (linia ciągła), jak i liczbę neuronów (linia przerywana). Dane dla zbioru 27-klasowego, klasy 5-tej i 16-tej. Jednostką czasu jest prezentacja pojedynczego wektora.

Z kolei rysunek 7.2 pokazuje proces uczenia 20-tej klasy dla zbioru 28-klasowego (tutaj kolor czerwony pokazuje poprawność dla zbioru treningowego). Kolejny rysunek – 7.3, otrzymano również na podstawie uczenia dla 28-klasowego zbioru danych. Dokładniej obrazuje on proces uczenia 9-tej klasy.

7.2.1.4. Porównanie i analiza wyników

W celach porównawczych zostały zebrane rezultaty uzyskane za pomocą różnych metod klasyfikacji, jak i metod wyciągania reguł logicznych. Sieć IncNet została porównana z siecią FSM [1, 64], która była wykorzystana jako klasyfikator (FSM z funkcjami Gaussa) i jako metoda wyciągania reguł logicznych (FSM z funkcjami prostokątnymi i FSM z funkcjami prostokątnymi i bicentralnymi z optymalizacją[1]). Do porównania użyto także metody uczenia maszynowego C 4.5 [187] do ekstrakcji reguł logicznych. Były wykonywane próby klasyfikacji przy użyciu innych metod, ale ich rezultaty były istotnie gorsze od zaprezentowanych w poniżej opisanych tabelach 7.3 i 7.4.

W tabeli 7.3 zostało ukazane porównanie, w którym wszystkie modele korzystały przy uczeniu z całego zbioru 27- i 28-klasowego odpowiednio.

Model	Uczenie na całym zbiorze	
	27 klasowym	28 klasowym
IncNet	99.22	99.23
C 4.5	93.67	93.06
FSM+R Opt.	97.57	96.91

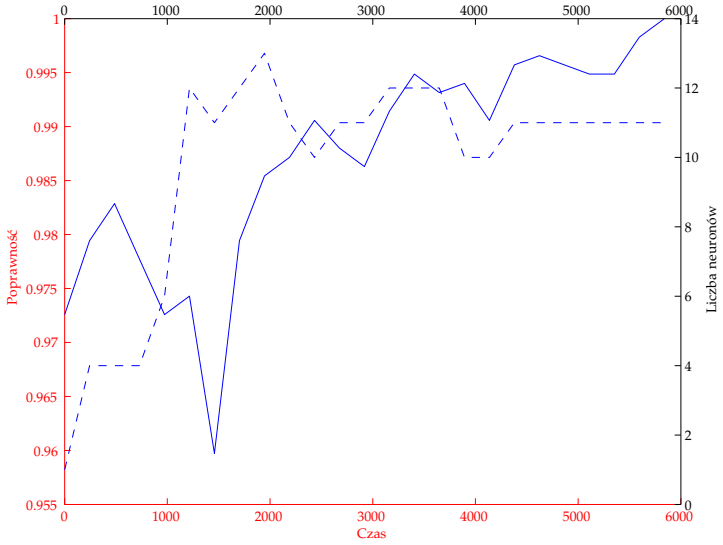
Tabela 7.3: Poprawność klasyfikacji w procentach dla różnych modeli adaptacyjnych. Modele były uczone na całym zbiorze 27- i 28-klasowym.

Tabela 7.4 porównuje możliwości generalizacji wyżej wspomnianych modeli z siecią IncNet. Tak jak i poprzednio użyto obu zbiorów (27- i 28-klasowego). Tabela prezentuje rezultaty uzyskane po uczeniu dla dwóch różnych podziałów. Pierwszy podział to 90% na zbiór treningowy i 10% na zbiór testowy. Drugi to 95% na zbiór treningowy i 5% na zbiór testowy.

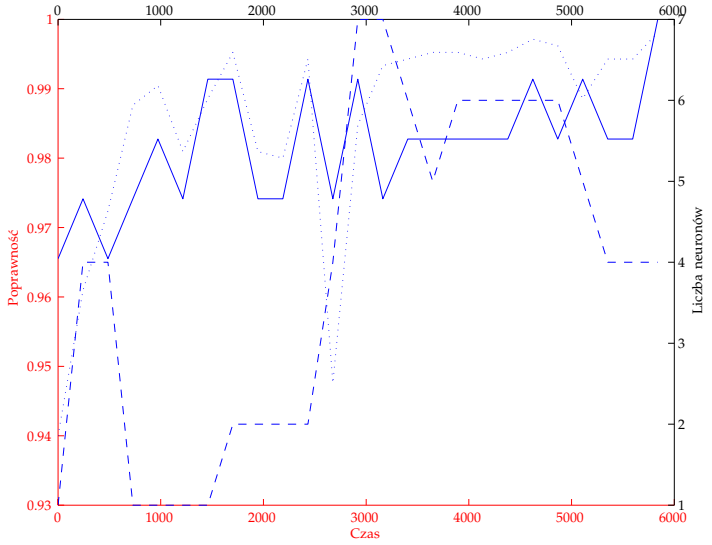
Przedstawione rezultaty pokazują, że sieć IncNet nie tylko uzyskała najlepsze wyniki na zbiorze treningowym, ale przede wszystkim na zbiorze testowym. W tabeli 7.4 widać drastyczną różnice w generalizacji pomiędzy siecią IncNet i regułami logicznych uzyskanymi z modeli C 4.5 i FSM+R. Różnica w poprawności klasyfikacji na zbiorze testowym wyniosła około 10%.

Sprawność sieci IncNet można również prześledzić, analizując macierze rozrzutu, które zostały wyznaczone dla sieci, powstałych przez uczenie na różnych danych i przy różnym podziale na część treningową i testową.

[1]FSM+R Opt. to reguły *miękkie* uzyskane w procesie optymalizacji. Takie *miękkie* reguły nie dają odpowiedzi typu TAK/NIE, lecz wartość z zakresu $[0, 1]$.



Rysunek 7.2: Wykres ilustruje zmieniającą się w czasie poprawność klasyfikacji dla zbioru treningowego (linia ciągła), jak i liczbę neuronów (linia przerywana). Dane dla zbioru 28-klasowego, klasy 20-tej.



Rysunek 7.3: Wykres ilustruje zmieniającą się w czasie poprawność klasyfikacji dla zbioru treningowego (linia kropkowana) i zbioru testowego (linia ciągła), jak i liczbę neuronów (linia przerywana). Dane dla zbioru 28-klasowego, klasy drugiej.

Model	zbiór 27-klasowy				zbiór 28-klasowy			
	10%		5%		10%		5%	
	TRS	TES	TRS	TES	TRS	TES	TRS	TES
IncNet	99.03	93.14	98.77	96.08	98.95	93.10	98.29	94.83
FSM + G	97.65	92.16			97.08	90.20		
FSM + R	96.97	84.70			96.66	82.76		
C 4.5	93.22	83.70			93.13	78.90		

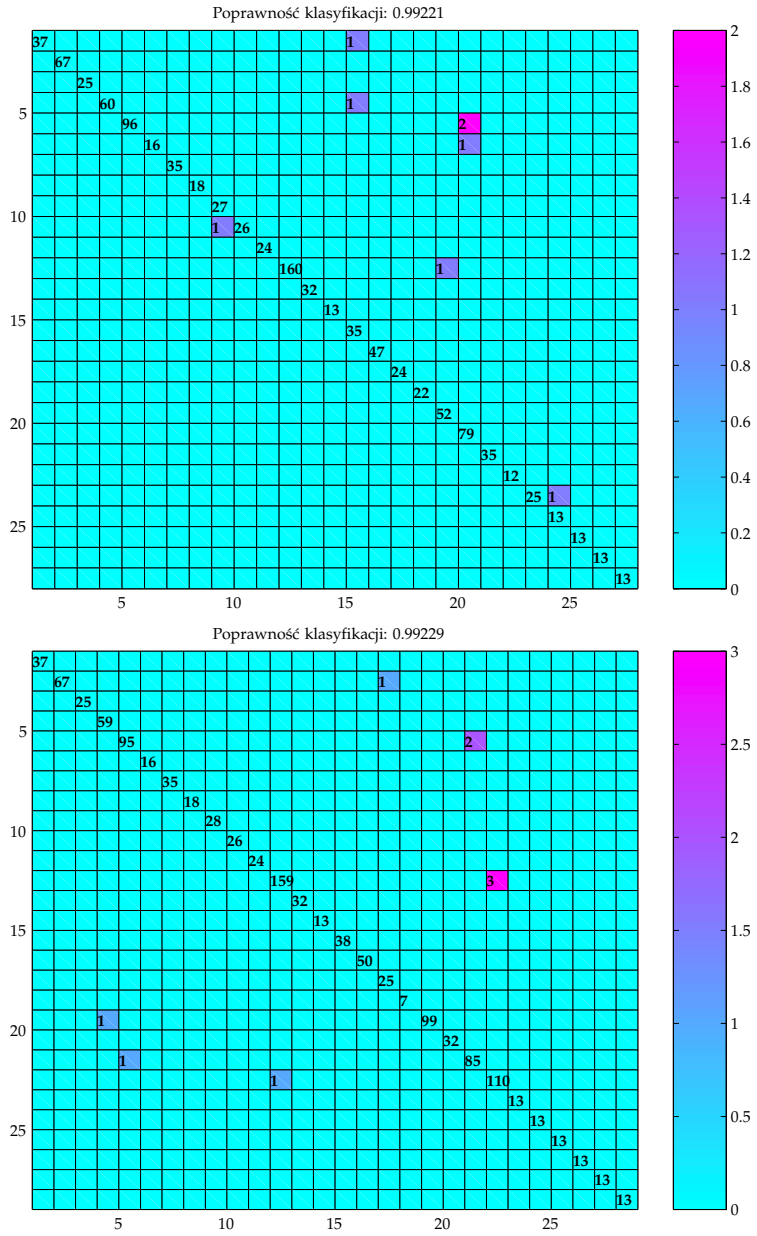
Tabela 7.4: Porównanie poprawność klasyfikacji w procentach danych psychometrycznych dla różnych modeli adaptacyjnych. 10% (lub 5%) oznacza, że 10% (lub 5%) wzorców branych jest do zbioru testowego, a reszta (90% lub 95%) wzorców stanowi zbiór treningowy.

Rysunek 7.4 prezentuje macierze rozrzutu, powstałe przy uczeniu sieci IncNet na całym zbiorze 27-klasowym (u góry) i 28-klasowym (u dołu). Wartości rzędnych i odciętych oznaczają numery poszczególnych klas, patrz opis na str. 240. Oś rzędnych odpowiada numerom klas uzyskanych w wyniku klasyfikacji, natomiast oś odciętych odpowiada numerom klas oczekiwanych. Kolejne dwa rysunki 7.5 i 7.6 pokazują macierze rozrzutu zbioru testowego i treningowego dla 27-klasowego zbioru. Macierze na pierwszym rysunku są wynikiem uczenia sieci IncNet przy podziale 90% + 10% (zbiór treningowy i testowy). Natomiast macierze na drugim rysunku są wynikiem podziału 95% + 5%. Ostatnie dwa rysunki z macierzami rozrzutu 7.7 i 7.8 zostały opracowane tak, jak poprzednie dwa, z wyjątkiem, iż użyto 28-klasowego zbioru danych.

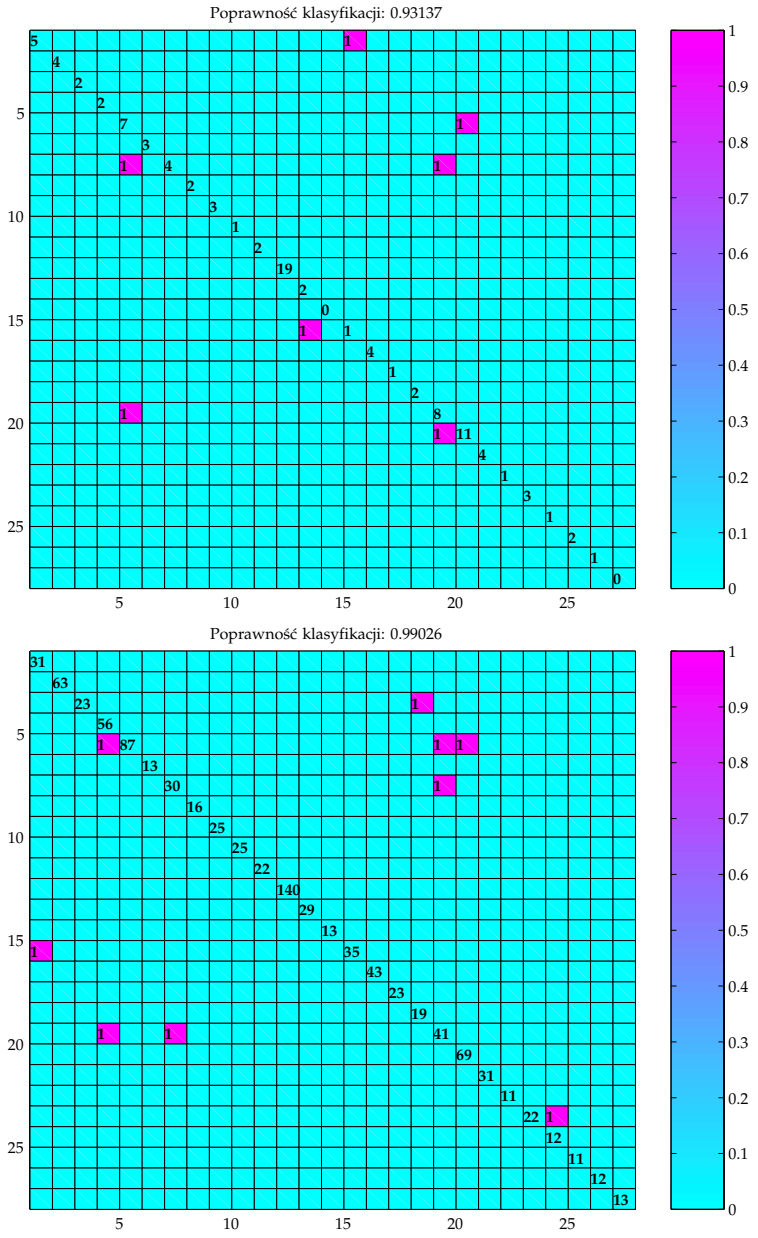
Bardzo ciekawa jest analiza wektorów, które przez sieć zostały źle sklasyfikowane. Na kilku kolejnych rysunkach będzie można porównać wartość otrzymaną na wyjściu sieci *zwycięskiej* klasy z wartością dla klasy oczekiwanej (tj. prawidłowej).

Jak już zostało opisane w podrozdziale 4.3.7, można prowadzić obserwację wartości wyjściowych $C^i(\mathbf{x})$ poszczególnych podsieci (patrz rysunek 4.12), bądź dokonać renormalizacji i obserwować (przybliżone) prawdopodobieństwa $p(C^i|\mathbf{x})$ przynależności do danych klas (patrz równanie 4.126).

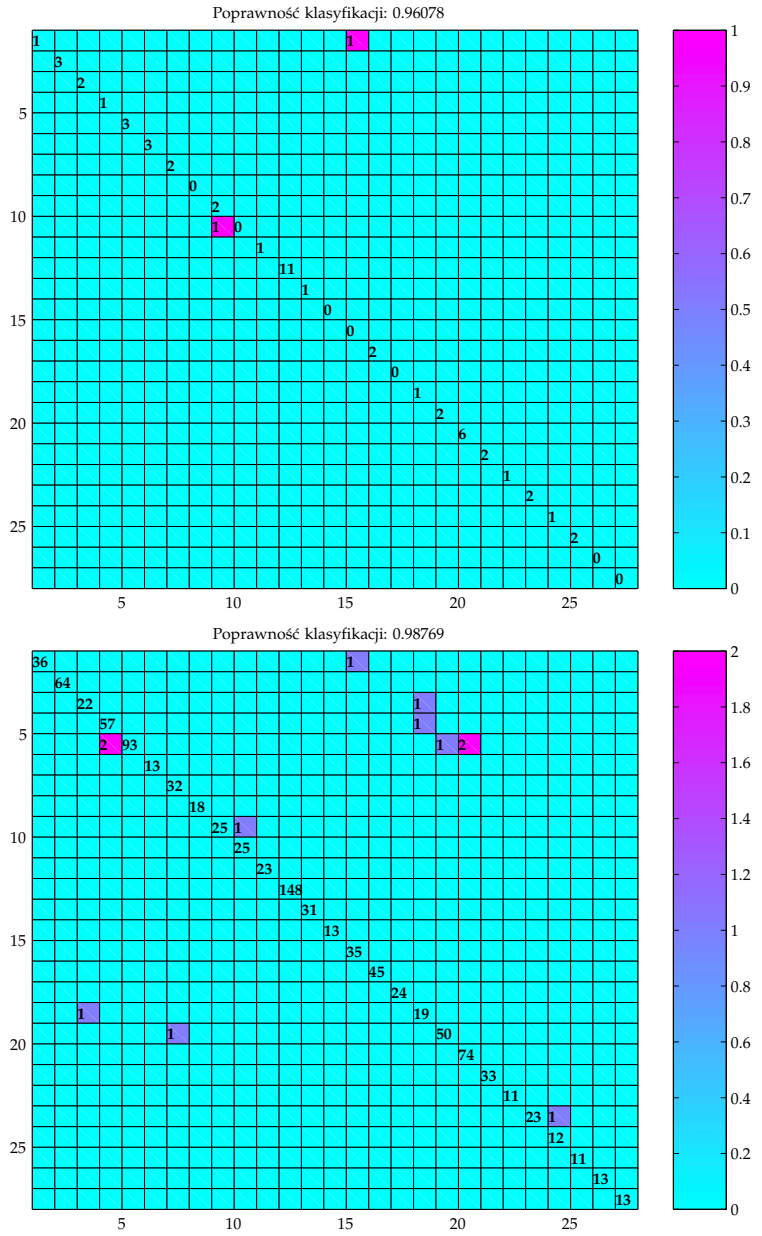
Wartości na lewej osi rysunków 7.9, 7.10, 7.11 i 7.12 odpowiadają wartością dla klasy zwycięskiej, a na osi prawej widać wartość dla klasy oczekiwanej. Rysunek 7.9 prezentuje wartości wyjściowe klasy zwycięskiej i oczekiwanej wektorów źle sklasyfikowanych dla 27- (u góry) i 28- (u dołu) -klasowego zbioru danych. Sieci były uczone na wszystkich danych. Drugi rysunek 7.10 w odróżnieniu od poprzedniego prezentuje wartości prawdopodobieństw dla klasy zwycięskiej i oczekiwanej. Kolejne dwa rysunki 7.11 i 7.12 prezentują również wartości prawdopodobieństw dla źle sklasyfikowanych wektorów dla zbiorów 27- i 28-klasowych. Sieci były uczone na 90% wektorów danych, a 10% danych stanowiło część testową. U góry rysunku przedstawione są źle sklasyfikowane wektory części testowej, a u dołu części treningowej.



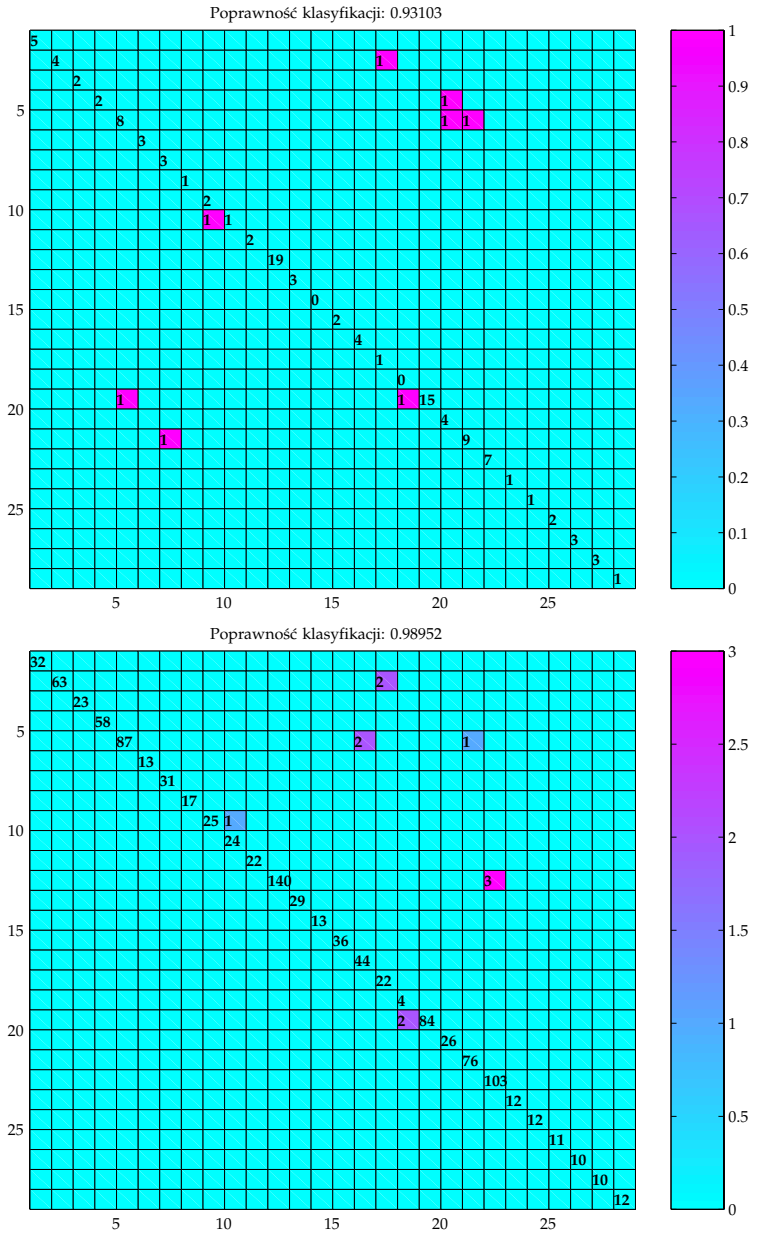
Rysunek 7.4: Macierze rozrzutu powstałe przy uczeniu na całym zbiorze. U góry dla 27-klasowego zbioru, u dołu dla 28-klasowego zbioru. Wartości rzędnych i odciętych oznaczają numery poszczególnych klas, patrz opis na str. 240.



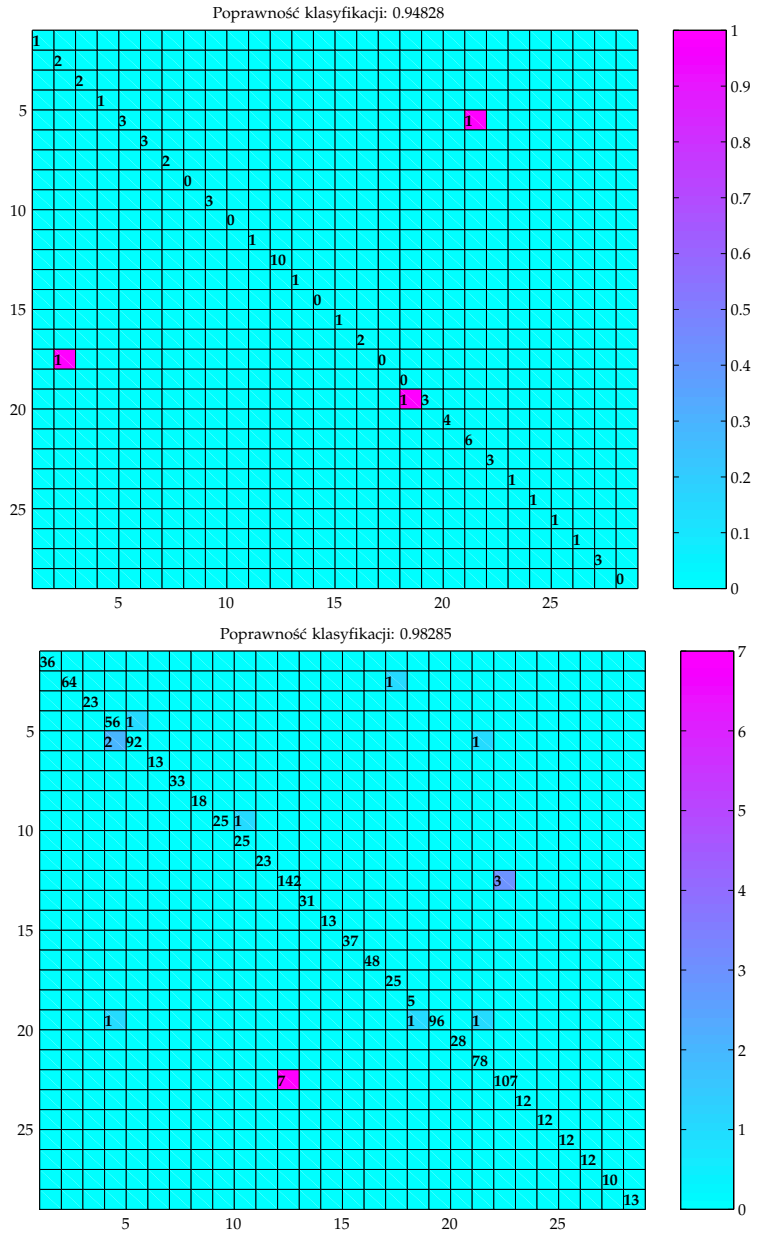
Rysunek 7.5: Macierze rozrzutu powstałe przy uczeniu na 90%-owej części zbioru 27-klasowego. U góry dla zbioru testowego, u dołu dla zbioru treningowego. Wartości rzędnych i odciętych oznaczają numery poszczególnych klas, patrz opis na str. 240.



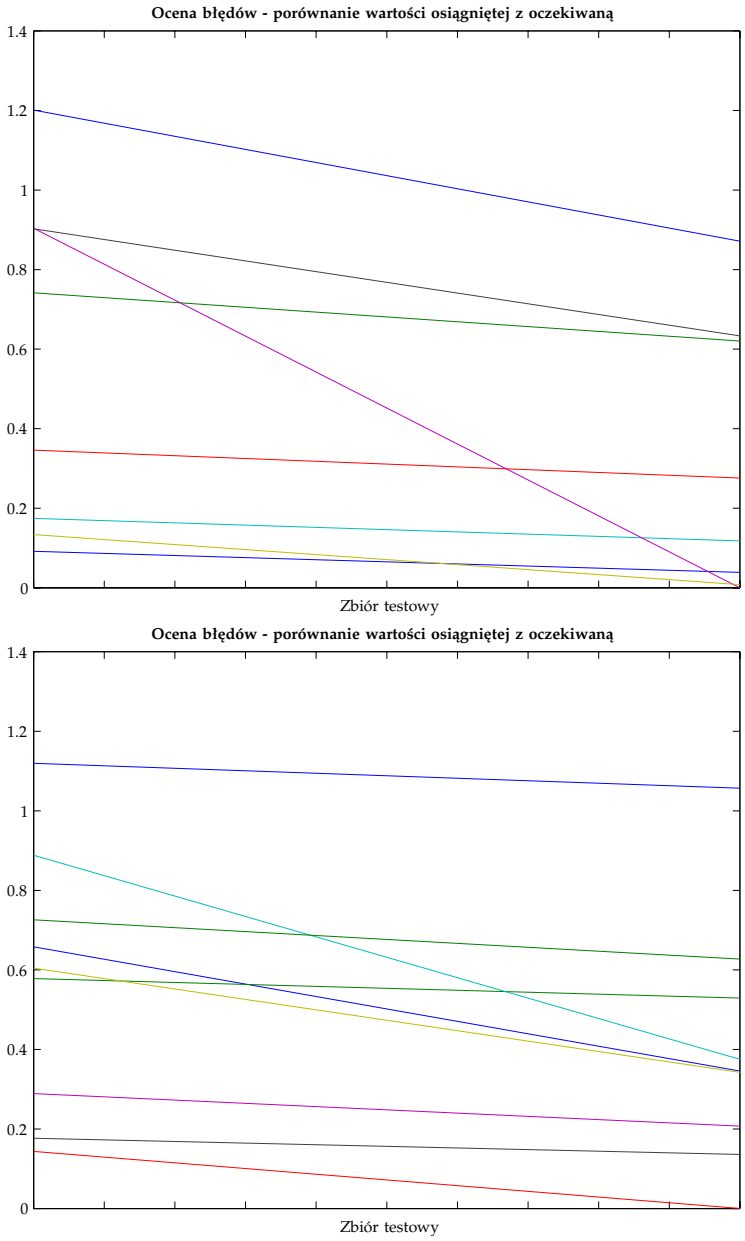
Rysunek 7.6: Macierze rozrzutu powstałe przy uczeniu na 95%-owej części zbioru 27-klasowego. U góry dla zbioru testowego, u dołu dla zbioru treningowego. Wartości rzędnych i odciętych oznaczają numery poszczególnych klas, patrz opis na str. 240.



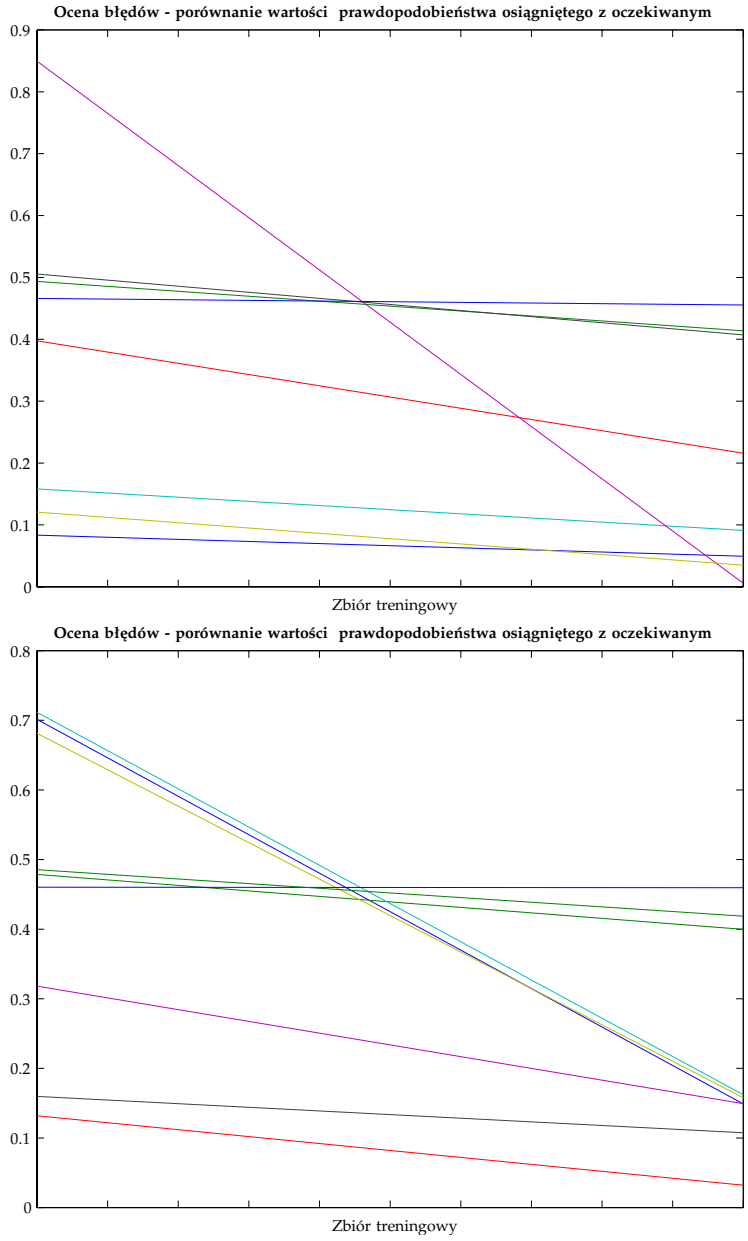
Rysunek 7.7: Macierze rozrzutu powstałe przy uczeniu na 90%-owej części zbioru 28-klasowego. U góry dla zbioru testowego, u dołu dla zbioru treningowego. Wartości rzędnych i odciętych oznaczają numery poszczególnych klas, patrz opis na str. 240.



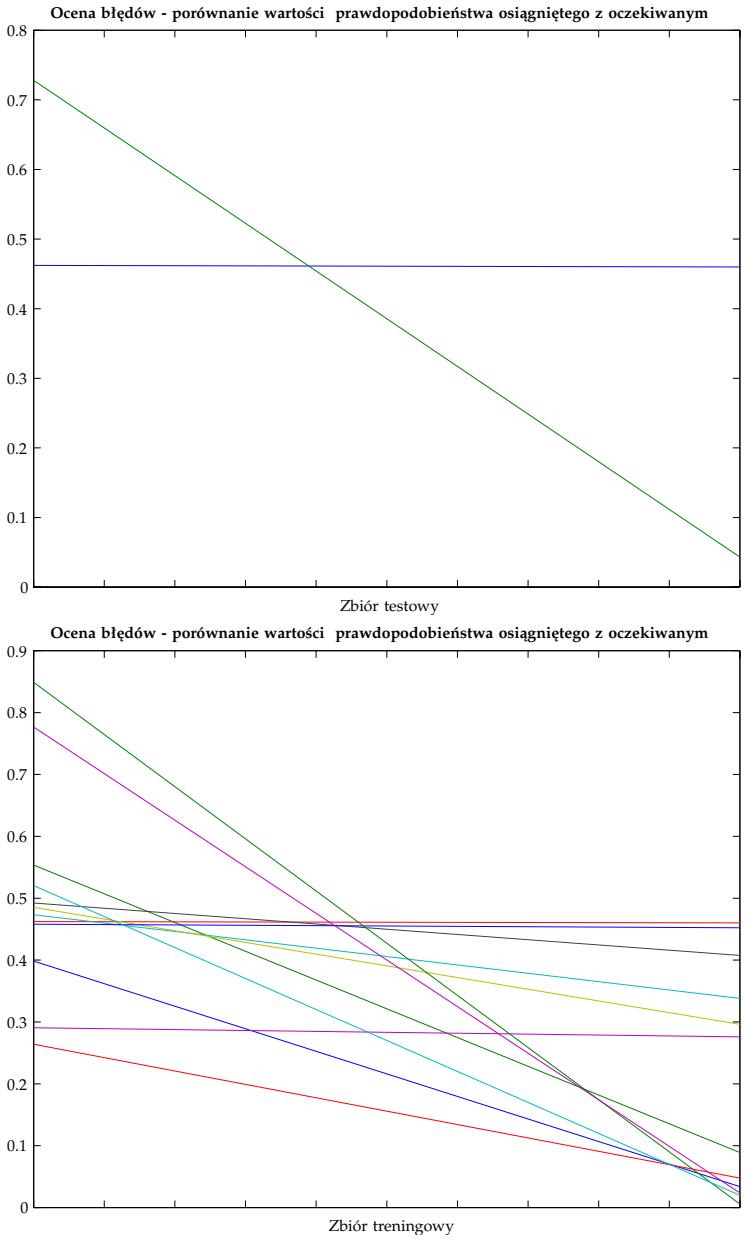
Rysunek 7.8: Macierze rozrzutu powstałe przy uczeniu na 95%-owej części zbioru 28-klasowego. U góry dla zbioru testowego, u dołu dla zbioru treningowego. Wartości rzędnych i odciętych oznaczają numery poszczególnych klas, patrz opis na str. 240.



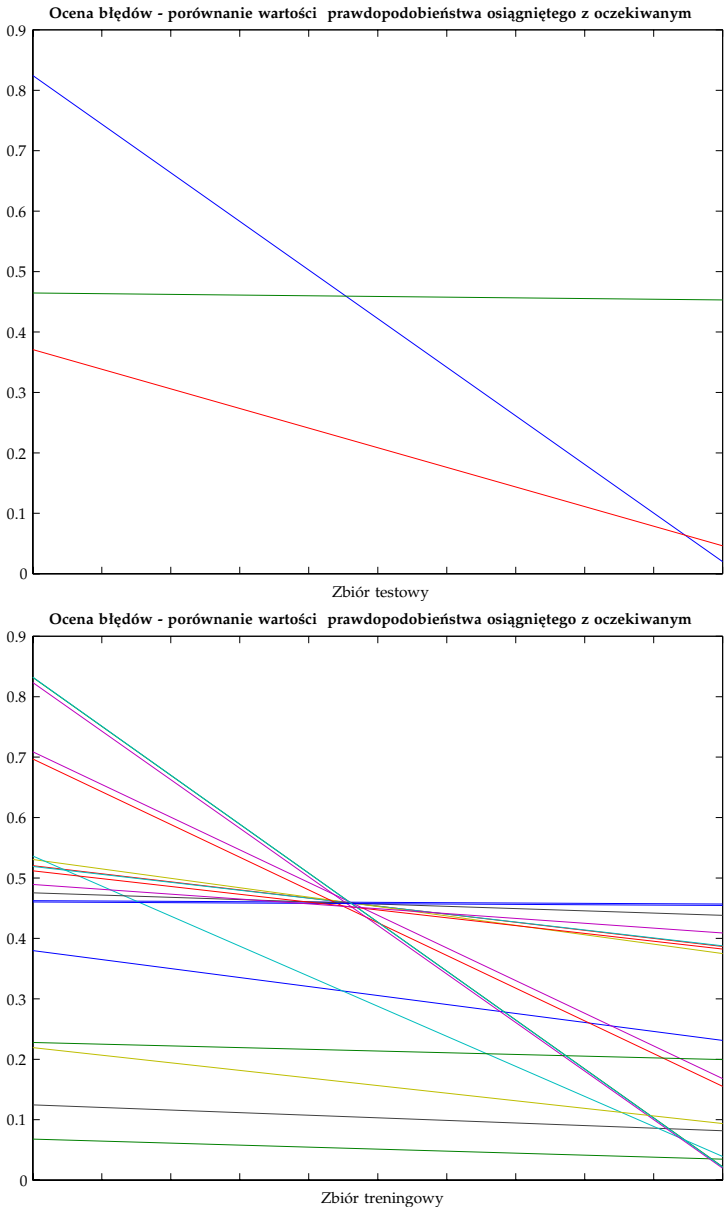
Rysunek 7.9: Porównanie wartości uzyskanych i oczekiwanych na podstawie błędnie sklasyfikowanych wektorów dla 27- i 28-klasowej bazy.



Rysunek 7.10: Porównanie wartości prawdopodobieństw uzyskanych i oczekiwanych na podstawie błędnie sklasyfikowanych wektorów dla 27- i 28-klasowej bazy.



Rysunek 7.11: Porównanie wartości prawdopodobieństw uzyskanych i oczekiwanych na podstawie błędnie sklasyfikowanych wektorów dla 27-klasowej bazy. Sieć uczona była na 95%-wej części danych. U góry dla zbioru testowego, u dołu dla zbioru treningowego.



Rysunek 7.12: Porównanie wartości prawdopodobieństw uzyskanych i oczekiwanych na podstawie błędnie sklasyfikowanych wektorów dla 28-klasowej bazy. Sieć uczona była na 95%-wej części danych. U góry dla zbioru testowego, u dołu dla zbioru treningowego.

Na rysunkach 7.9, 7.10, 7.11 i 7.12 większość linii łączących wartości dla klasy zwycięskiej i oczekiwanej nie są strome. Oznacza to, że klasyfikacja danego przypadku wcale nie była taka jednoznaczna. Tak może być nie tylko dla źle sklasyfikowanych przypadków. Taka niejednoznaczność pokazuje, że nie ma jedynego idealnego rozwiązania, wskazując alternatywne diagnozy, co nie jest odosobnionym przypadkiem w realnych problemach, z jakimi mamy do czynienia. Z niejednoznacznością można mieć do czynienia nie tylko dla źle sklasyfikowanych przypadków, ale i dla dobrze sklasyfikowanych przypadków, co również jest bardzo ważne. Niejednoznaczność przy dobrze nauczonym modelu oznacza, że mamy do czynienia z przypadkiem, który znajduje się *pomiędzy* dwoma obszarami, należącymi do różnych klas (lub znajduje się w obszarze wspólnym klas). Dlatego właśnie należy nie tylko brać pod uwagę zwycięską klasę, ale i rozwiązania alternatywne, które czasem mogą okazać się niemal tak samo prawdopodobne. W przypadku rozpatrywanych danych psychometrycznych jest to bardzo istotne (np. umożliwia płynne śledzenie zmian chorobowych). Obserwując wartości klas, dla których wyznaczone prawdopodobieństwo jest istotnie większe od zera, uwidacznia się właściwy rozkład udziału klasyfikowanego wektora na poszczególne klasy. W przypadku wyraźnego odstępstwa wartości oczekiwanych od otrzymanych (strome krzywe) jest całkiem prawdopodobne, że to psycholog dokonał niewłaściwej diagnozy.

Niżej opisany przypadek pokazuje, że w pewnych skrajnych przypadkach oprócz prawdopodobieństw $p(C^i|\mathbf{x})$ (por. równanie 4.126) przydatna może okazać się także maksymalna spośród wartości $C^i(\mathbf{x})$ $(i = 1, 2, \ldots, K)$. Gdy dla pewnego wektora prawdopodobieństwo nie jest skumulowane wokół jednej klasy lub kilku klas, może to oznaczać, że wektor jest spoza obszaru aktywności wszystkich klas. Czyli leży daleko od wektorów treningowych. Odpowiada to przypadkowi całkiem odmiennemu od tych, jakie obecne są w bazie treningowej. W takim przypadku maksymalna wartości spośród $C^i(\mathbf{x})$ $(i = 1, 2, \ldots, K)$ będzie miała małą wartość.

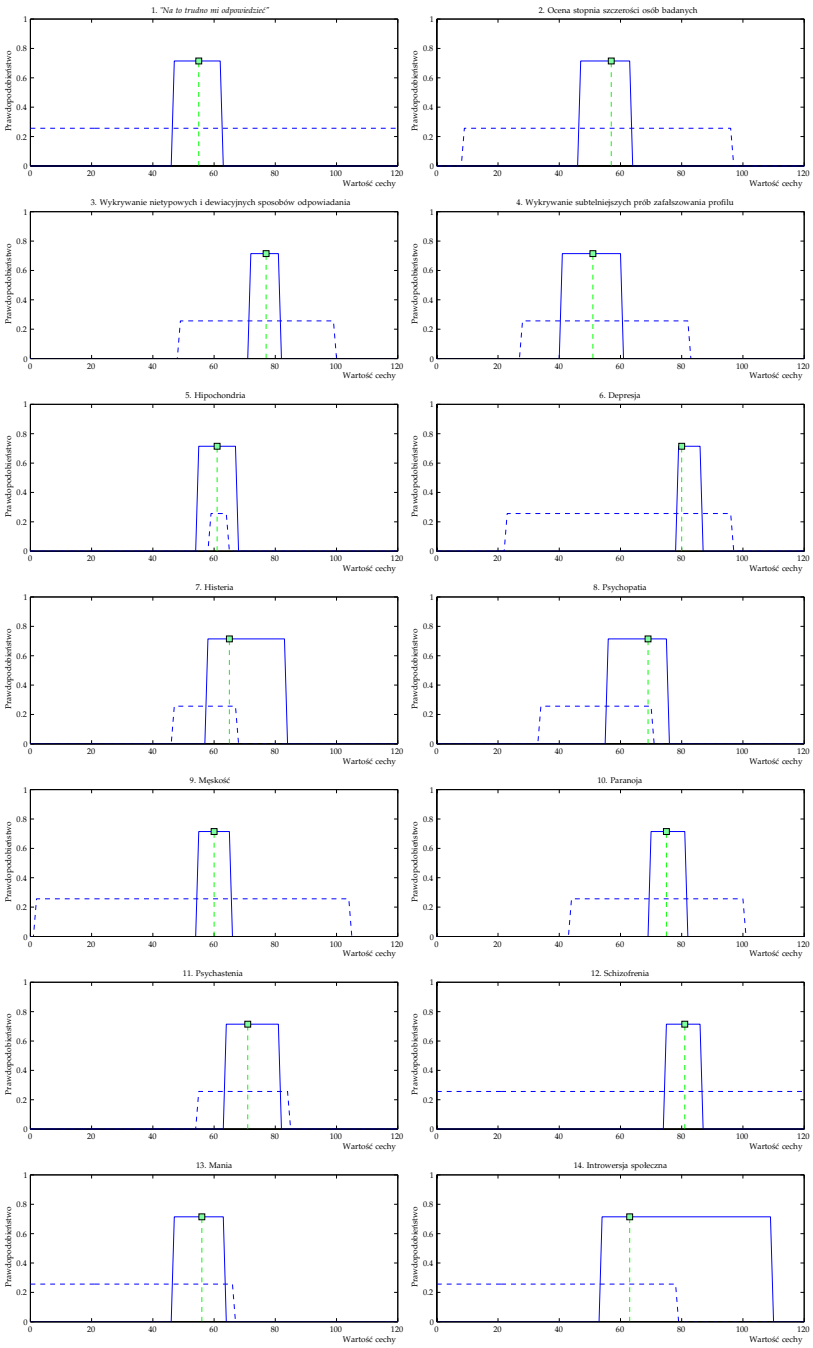
Inne, bardzo ciekawe wyniki można uzyskać wyznaczając przedziały ufności i probabilistyczne przedziały ufności, opisane w podrozdziale 6.5. Jak już zostało wspomniane wcześniej, przedziały ufności i ich probabilistyczna odmiana są bardzo efektywnym narzędziem wspomagania procesu diagnozy i wizualizacji procesu klasyfikacji.

Poniżej przedstawione zostaną różne przykłady z psychometrycznych baz danych. Pierwszy niech będzie nawiązaniem do pierwszego przypadku z podrozdziału 6.5. Ten przypadek okazał się mniej jednoznaczny, ale został dobrze sklasyfikowany przez sieć. Przypadek został zaklasyfikowany do psychoz reaktywnych, uzyskując prawdopodobieństwo 0.71, natomiast do klasy schizofrenii z prawdopodobieństwem 0.26. Rysunek 7.13 przedstawia przedziały ufności, a rysunek 7.14 przedstawia probabilistyczne przedziały ufności. Szczególnie wyraźnie widać różnicę pomiędzy rysunkiem 6.6, a rysunkiem 7.14. Bez jakichkolwiek trudności można ocenić jednoznaczność procesu klasyfikacji i wpływ poszczególnych cech. Rysunek 7.14 uwidacznia nakładanie się rozkładów klas

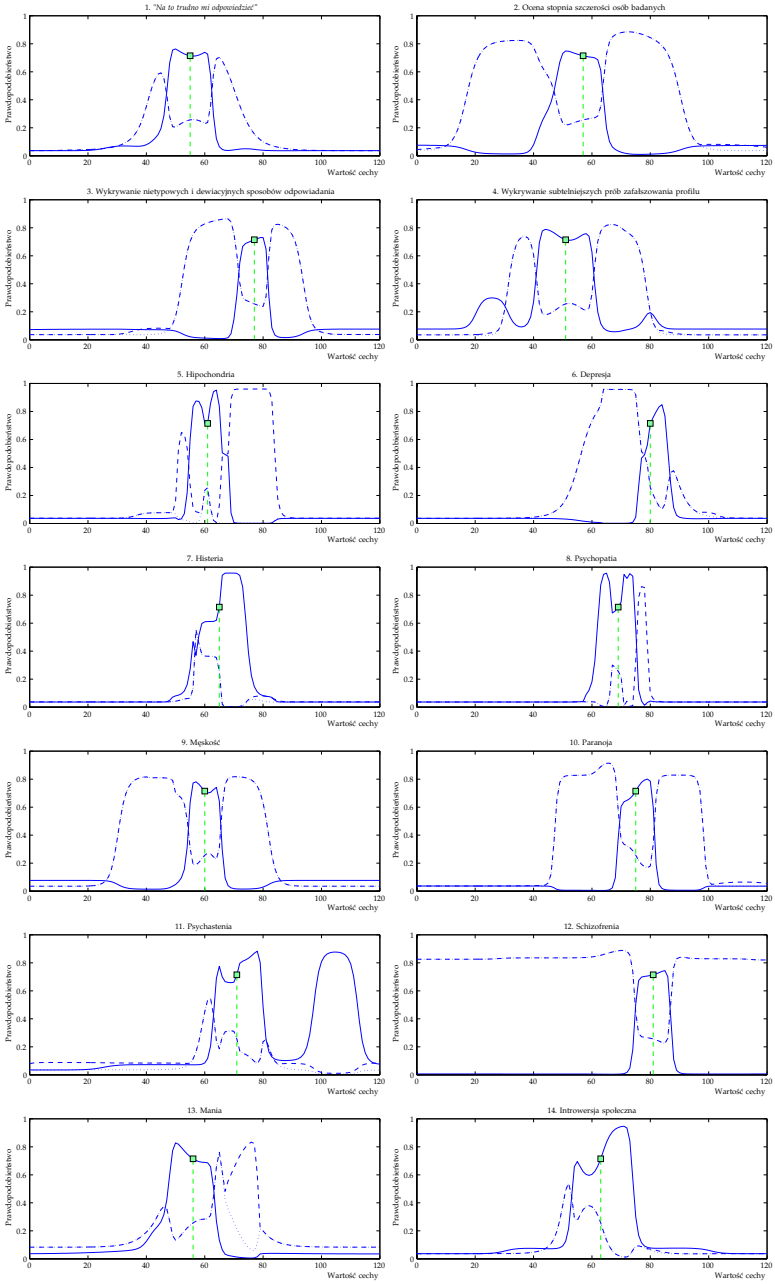
psychoz reaktywnych i schizofrenii.

Kolejny przykład został opracowany na podstawie przypadku, który został jednoznacznie zaklasyfikowany do zespołów urojeniowych. Prawdopodobieństwo klasyfikacji wyniosło 0.96. Przedziały ufności zostały przedstawione na rysunku 7.15 i 7.16.

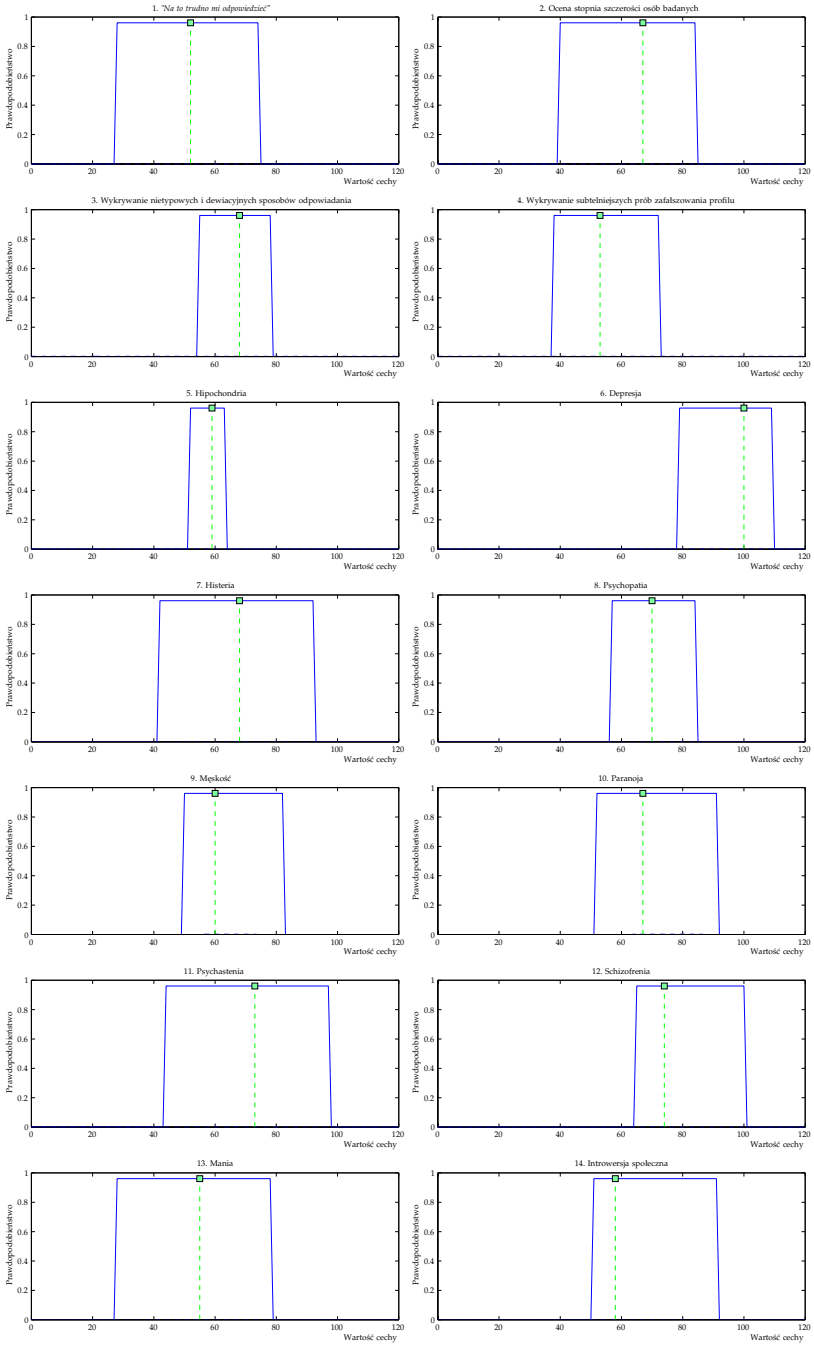
Rysunki 7.17 i 7.18 ilustrują przedziały ufności przypadku, który choć niezupełnie jednoznacznie, ale został zaklasyfikowany do schizofrenii wspólnej (mężczyzn i kobiet), a przez psychologów został on sklasyfikowany jako schizofrenia kobiet. Prawdopodobieństwa tych dwóch klas wyniosły: 0.63 i 0.023. Jednak dziś trudno powiedzieć, gdzie tak naprawdę leży błąd, czy po stronie modelu adaptacyjnego, czy też błąd został popełniony przez psychologa.



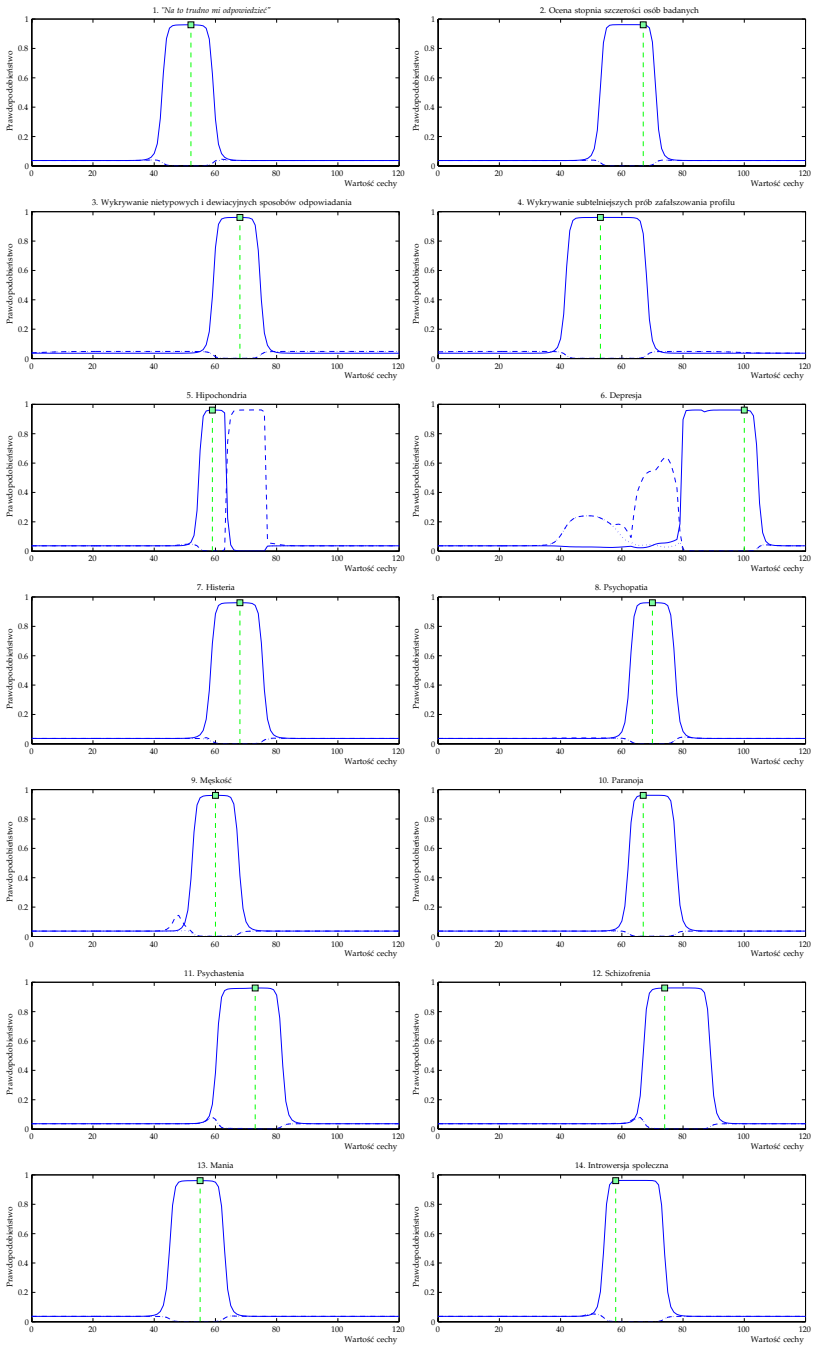
Rysunek 7.13: Przedziały ufności. Przypadek psychozy reaktywnej.



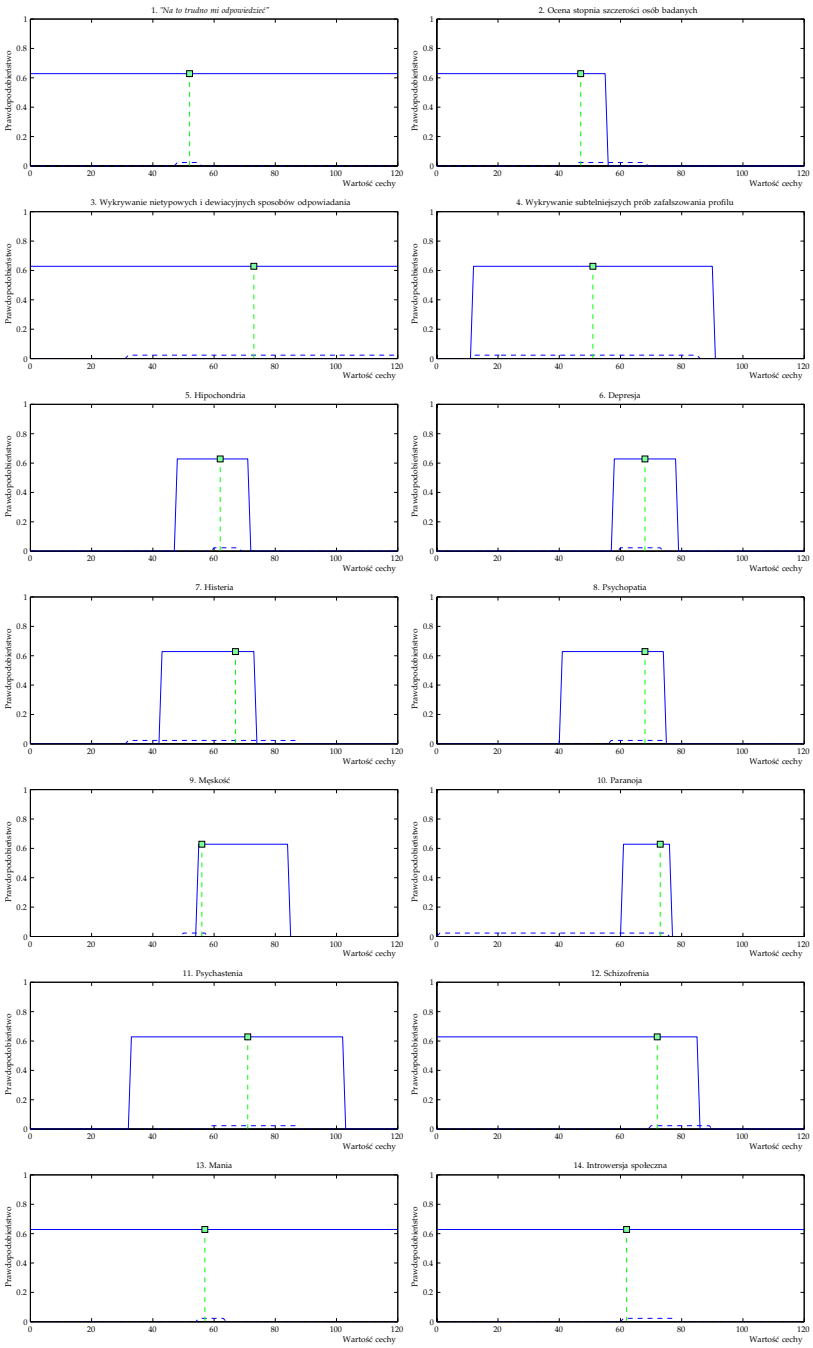
Rysunek 7.14: Probabilistyczne przedziały ufności. Przypadek psychozy reaktywnej.



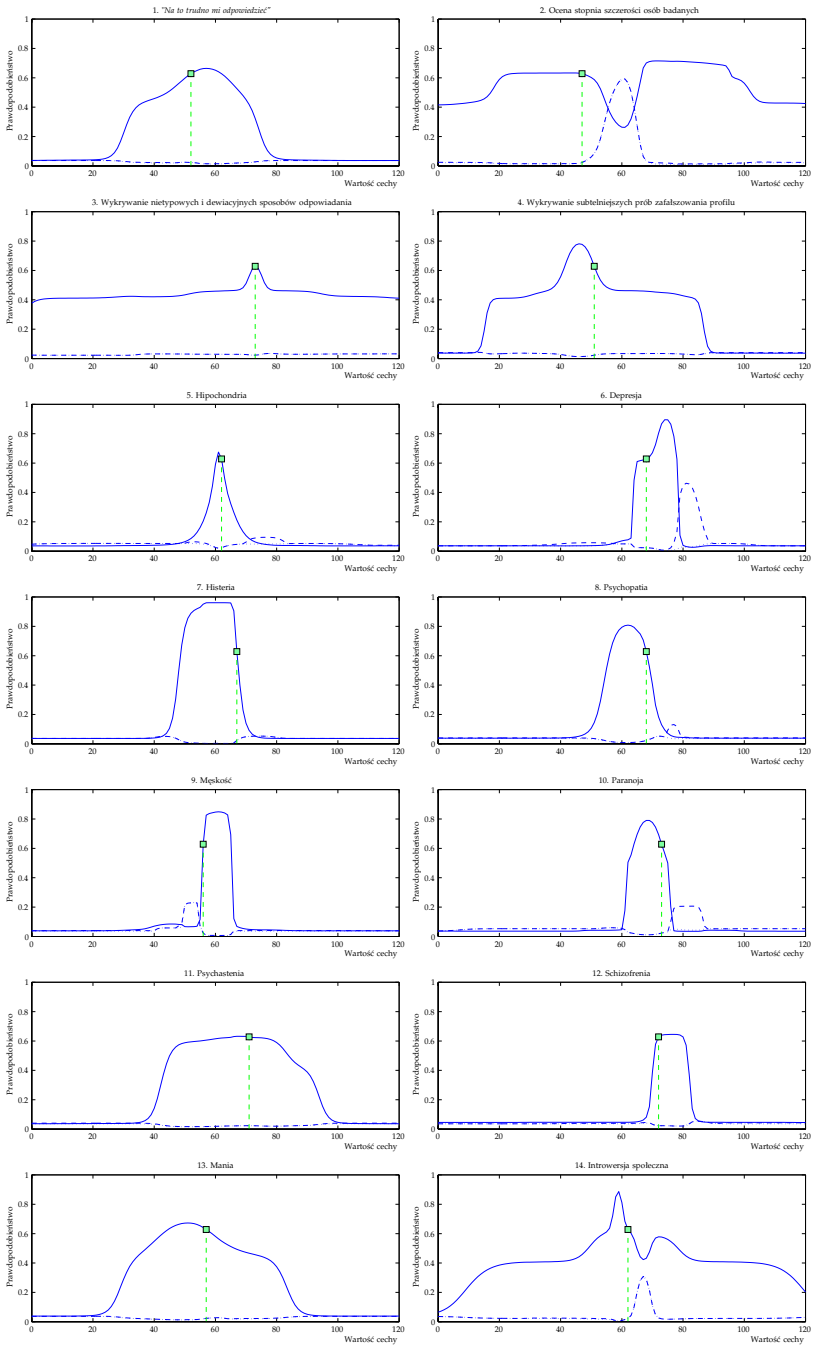
Rysunek 7.15: Przedziały ufności. Zespół urojeniowy.



Rysunek 7.16: Probabilistyczne przedziały ufności. Zespół urojeniowy.



Rysunek 7.17: Przedziały ufności. Przypadek schizofrenii.



Rysunek 7.18: Probabilistyczne przedziały ufności. Przypadek schizofrenii.

7.2.2. Typowe medyczne dane porównawcze

Aby porównać działanie sieci IncNet z innymi znanymi z literatury metodami adaptacyjnymi należało użyć danych, które są ogólnie dostępne i były używane do porównań przeróżnych modeli.

W związku z tym, iż większość ogólnie dostępnych baz danych nie zawiera pełnych opisów źródeł danych, jak i opisu samych danych (opis cech, zakresy i znaczenie skal) przedstawione zostaną wszystkie dostępne informacje.

7.2.2.1. Zapalenie wyrostka robaczkowego

Dane dotyczące zapalenia wyrostka robaczkowego (*ang. appendicitis*) otrzymano od prof. Shalom Weiss z Rutgers University. Dane składają się ze 106 wektorów. Każdy wektor opisany jest przez 8 cech i może być przypisany do jednej z dwóch klas (80.2% wektorów należy do pierwszej klasy opisującej przypadki ostrego zapalenia, natomiast 19.8% do drugiej klasy opisującej inne problemy).

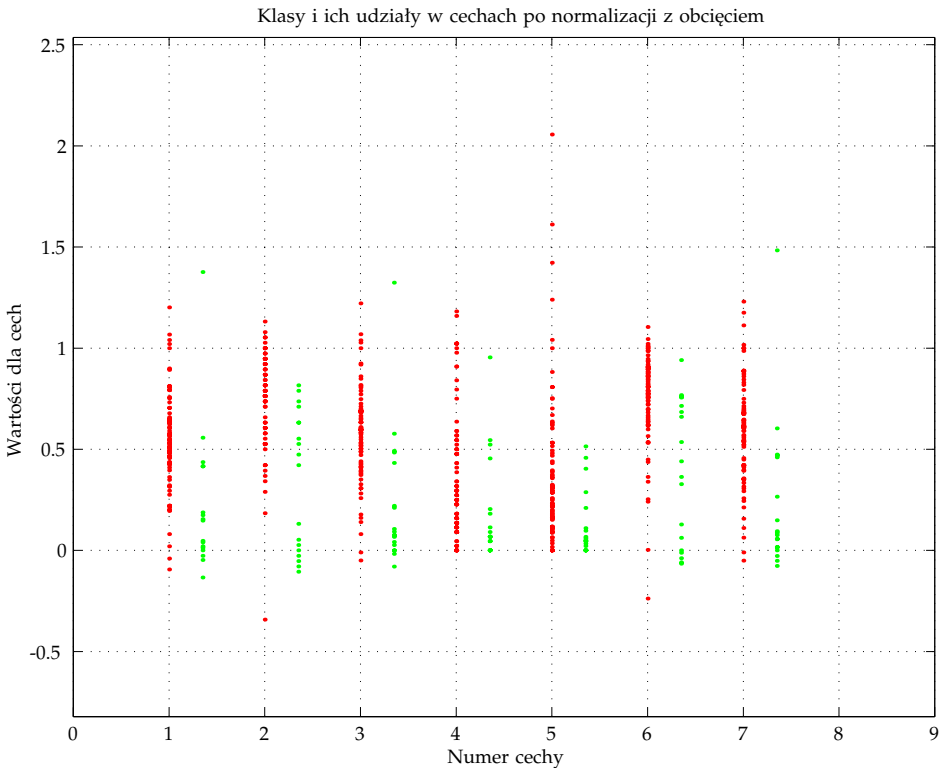
W tabeli 7.5 zostały umieszczone rezultaty uzyskane przy użyciu testu 10 CV. Sieć IncNet uzyskała poprawność 90.11% na zbiorze treningowym i 90.90% na zbiorze testowym dla testu CV 10. Uczenie wynosiło 1100 iteracji. Średnio sieć składała się z 30 neuronów (30 neuronów na dwie podsieci). Do uzyskania takiego samego rezultatu w teście CV 10 można było użyć jednej sieci IncNet, która uczyła się rozpoznawania wektorów klasy pierwszej, wtedy średnia liczba neuronów spada do 20. Dane były normalizowane z 5% obcięciem. Dane można obejrzeć na rysunku 7.19. Dla każdej z cech różnokolorowe kolumny punktów (horyzontalnie nieco przesunięte względem siebie) odpowiadają różnym klasom.

KMK w opisie źródła oznacza, że wynik został otrzymany w naszym zespole przy użyciu ogólnie dostępnej wersji jakiegoś symulatora, bądź symulatorem opracowanym w naszym zespole.

Metoda	Poprawność	Źródło wyniku
IncNet	90.9	KMK
6-NN	88.0	KMK
FSM	84.9	KMK [64]
MLP+BP (Tooldiag)	83.9	KMK
RBF (Tooldiag)	80.2	KMK

Tabela 7.5: Zapalenie wyrostka robaczkowego — porównanie rezultatów dla CV 10.

Natomiast w tabeli 7.6 zostały umieszczone rezultaty otrzymane przy użyciu testu LOO dla różnych metod. Pomimo faktu, że zazwyczaj rezultaty testu LOO są lepsze, to wartość testu CV 10 dla sieci IncNet jest lepsza od każdej wartości testu LOO otrzymanych dla różnych modeli zaprezentowanych w tabeli 7.6.



Rysunek 7.19: Baza danych wyrostka robaczkowego.

7.2.2.2. Dane dotyczące raka piersi.

Dane dotyczące raka piersi zebrane zostały w szpitalu Uniwersytetu Wisconsin [179, 263] przez Dr. William H. Wolberga i są obecnie dostępne w Instytucie Informacji i Informatyki na Uniwersytecie Kalifornijskim w Irvine [182] (*Wisconsin breast cancer*). Baza składa się z 699 przypadków opisywanych przez 9 cech. Cechy opisują m. in. rozmiar (grupy) guzów, rozmiar komórek i ich kształt, przyleganie, pewne cechy krwi. Wektory pierwszej z dwóch klas stanowią 65.5% wektorów całego zbioru. Pierwsza klasa opisuje nowotwory niezłośliwe, natomiast druga złośliwe.

Sieć IncNet na zbiorze treningowym uzyskała poprawność 97.6% i 97.1% na zbiorze testowym. Średnio końcowa sieć składała się z około 40 neuronów. Uczenie wynosiło 3000 iteracji. Dane zostały znormalizowane. Dane można obejrzeć na rysunku 7.20. Dla każdej z cech różnokolorowe kolumny punktów (horyzontalnie nieco przesunięte względem siebie) odpowiadają różnym klasom.

W tabeli 7.7 zostały umieszczone rezultaty otrzymane przy użyciu testu 10 CV.

Metoda	Poprawność	Źródło wyniku
PVM (reguły logiczne)	89.6	Weiss, Kapouleas [255]
C-MLP2LN (reguły logiczne)	89.6	KMK
k-NN	88.7	KMK
RIAC	86.9	Hamilton m. in. [112]
MLP+BP	85.8	Weiss, Kapouleas [255]
CART, C4.5	84.9	Weiss, Kapouleas [255]
FSM	84.9	KMK [64]

Tabela 7.6: Zapalenie wyrostka robaczkowego — porównanie rezultatów dla testu LOO.

7.2.2.3. Dane dotyczące zapalenia wątroby.

Baza danych opisująca przypadki zapalenia wątroby (*ang. hepatitis*) pochodzi od G. Gong z Uniwersytetu Carnegie-Mellon i także jest dostępna w UCI [182]. Na dane składa się 155 wektorów, a przestrzeń wejściowa opisywana jest przez 19 cech. Cechy opisują wiek, płeć, steryd, zużycie, samopoczucie, anoreksje, czy wątroba jest duża, twardość, bilirubinę i inne. Każdy z wektorów może przynależeć do jednej z dwóch klas (bardziej liczna klasa zawiera 79.4% wszystkich przypadków), które opisują przeżywalność.

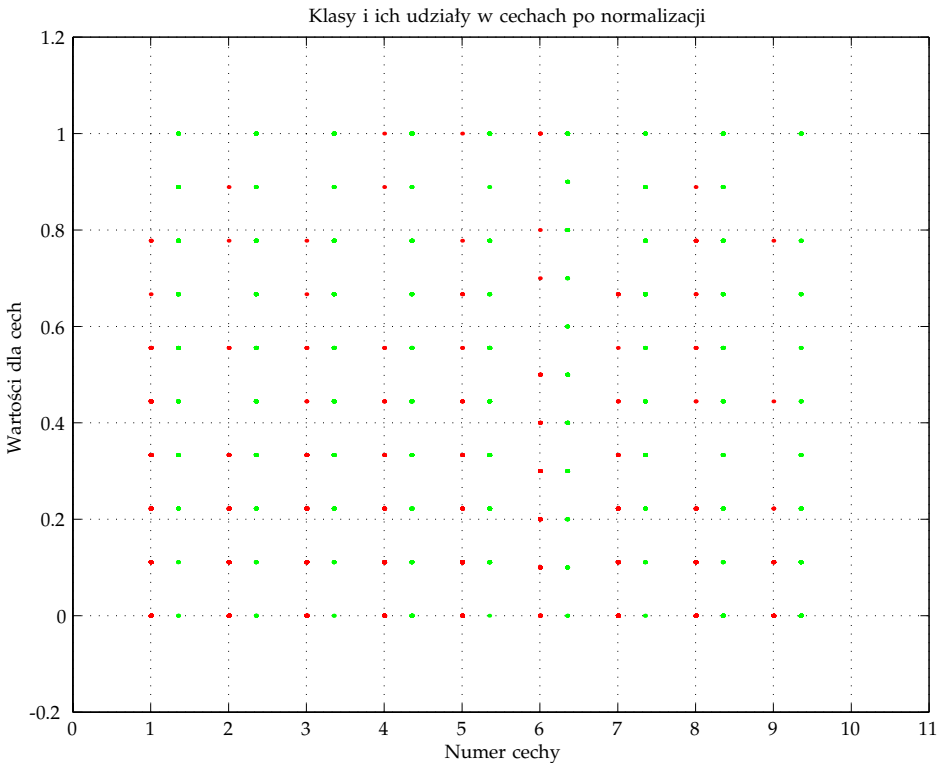
Sieć IncNet uzyskała poprawność 99.36% na zbiorze treningowym i 86% na zbiorze testowym dla testu CV 10. Użyto bicentralnych funkcji transferu z możliwością obrotu. Uczenie wynosiło 2200 iteracji. Średnio sieć składała się z 34 neuronów (34 neurony na dwie podsieci dla każdej klasy). Dane były normalizowane z 5% obcięciem. Dane można obejrzeć na rysunku 7.21. Dla każdej z cech różnokolorowe kolumny punktów (horyzontalnie nieco przesunięte względem siebie) odpowiadają różnym klasom.

W tabeli 7.8 zostały umieszczone rezultaty otrzymane przy użyciu testu 10 CV.

7.2.2.4. Dane dotyczące cukrzycy

Baza danych chorób cukrzycy pochodzi z National Institute of Diabetes and Digestive and Kidney Diseases [234]. Można ją także znaleźć w UCI [182] (*Pima Indian diabetes*). Na całość zbioru składa się 768 przypadków, w tym 65.1% stanowią przypadki pierwszej klasy. Przynależność do klasy drugiej oznacza iż osoba jest chora na cukrzycę. Każdy przypadek jest opisywany przez 8 cech. Cechy opisują, ile razy pacjentka była w ciąży, test tolerancji glukozy, ciśnienie rozkurczowe, poziom insuliny, masa ciała, czy ktoś w rodzinie był chory na cukrzycę, wiek.

Sieć IncNet uzyskała poprawność 77.2% na zbiorze treningowym i 77.6% na zbiorze testowym dla testu CV 10. Uczenie wynosiło 5000 iteracji. Średnio sieć składała się z 100 neuronów. Dane zostały znormalizowane. Dane można



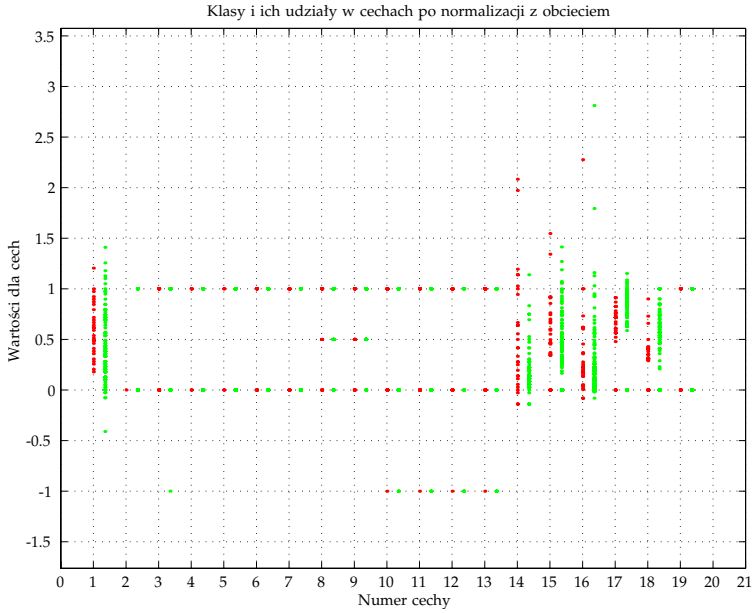
Rysunek 7.20: Baza danych raka piersi.

obejrzeć na rysunku 7.22. Dla każdej z cech różnokolorowe kolumny punktów (horyzontalnie nieco przesunięte względem siebie) odpowiadają różnym klasom.

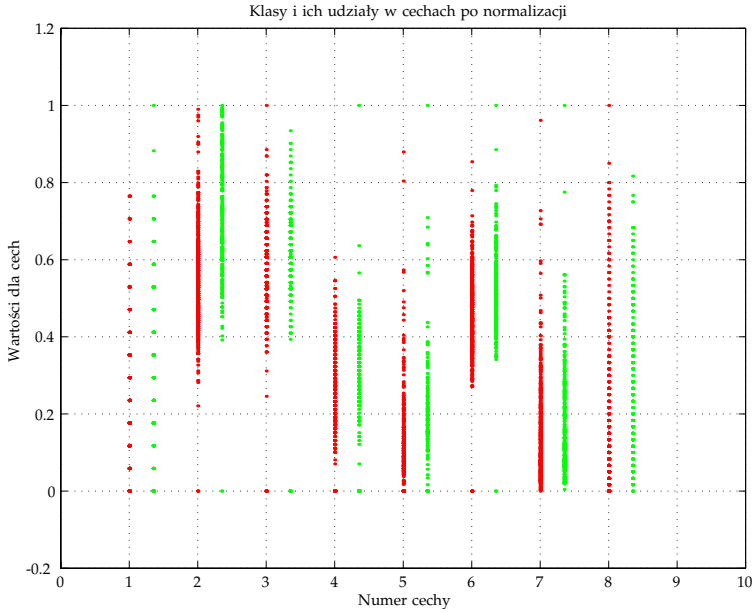
W tabeli 7.9 zostały umieszczono rezultaty uzyskane przy użyciu testu 10 CV.

7.2.2.5. Choroby tarczycy

Dane dotyczące chorób tarczycy również można znaleźć w UCI [182] (*Thyroid disease*). Dane chorób tarczycy składają się z dwóch zbiorów: treningowego, który składa się z 3772 przypadków (2.47% – pierwsza klasa, 5.06% – druga klasa i 92.47% – trzecia klasa) i testowego na który składa się 3428 przypadków (2.13% – pierwsza klasa, 5.16% – druga klasa i 92.71% – trzecia klasa). Trzy klasy opisują niedoczynność, nadczynność i normalną czynność tarczycy. Dominują przypadki zdrowej tarczycy, gdyż dane pochodzą z dwóch kolejnych lat badań przesiewowych. Przestrzeń wejściowa składa się z 21 cech (15 binarnych



Rysunek 7.21: Baza danych zapalenia wątroby.



Rysunek 7.22: Baza danych cukrzycy.

Metoda	Poprawność	Źródło wyniku
IncNet	97.1	KMK
3-NN, miara Manhattan	97.1	KMK
20-NN, miara euklidesowa	96.9	KMK
Analiza dysk. Fishera	96.8	Ster& Dobnikar [236]
MLP + Wsteczna propagacja	96.7	Ster& Dobnikar [236]
LVQ (ang. Learning vector quantization)	96.6	Ster& Dobnikar [236]
KNN	96.6	Ster& Dobnikar [236]
Analiza bayesowska	96.6	Ster& Dobnikar [236]
FSM - (ang. Feature Space Mapping)	96.5	KMK [64]
Analiza bayesowska + niezależność cech	96.4	Ster& Dobnikar [236]
DB-CART (drzewa decyzyjne)	96.2	Shang & Breiman [232]
Liniowa analiza dysk.	96.0	Ster& Dobnikar [236]
RBF (Tooldiag)	95.9	KMK
ASI (ang. Assistant-I)	95.6	Ster& Dobnikar [236]
ASR (ang. Assistant-R)	94.7	Ster& Dobnikar [236]
LFC (ang. Lookahead feature constructor)	94.4	Ster& Dobnikar [236]
CART (drzewa decyzyjne)	94.2	Ster& Dobnikar [236]
Kwadratowa analiza dysk.	34.5	Ster& Dobnikar [236]

Tabela 7.7: Dane dotyczące raka piersi — porównanie rezultatów.

i 6 ciągłych). Cechy opisują wiek, płeć, czy pacjent jest leczony tyroksyną, być może pacjent jest leczony tyroksyną, czy pacjent bierze leki przeciwtarczycowe, czy pacjent jest chory (czy źle się czuje), czy pacjentka była w ciąży, czy były wykonywane operacje tarczycy, czy pacjent był leczony jodem J131, test na niedoczynność, test na nadczynność, czy jest stosowany cytrynian litowy, czy występuje wole, czy jest nowotwór, czy występuje niedoczynność przysadki mózgowej, symptomy psychologiczne, poziom TSH, poziom T3, poziom TT4, poziom T4U, poziom FTI.

W pierwszym etapie dokonano selekcji istotnych cech z pomocą specjalnie do tego celu opracowanej wersji kNN [67]. Istotne okazały się cechy które opisują czy pacjent zażywa tyroksynę, czy przeszedł operacje tarczycy, poziom TSH, poziom TT4, poziom FTI. Po selekcji dane zostały poddane transformacji opisanej przez równanie 6.7, a następnie dane zostały poddane standaryzacji.

Dane można obejrzeć na rysunku .8. Dla każdej z cech różnokolorowe kolumny punktów (horyzontalnie nieco przesunięte względem siebie) odpowiadają różnym klasom.

Metoda	Poprawność	Źródło wyniku
18-NN, miara Manhattan	90.2	KMK
FSM z rotacją f. transferu	89.7	KMK [64]
15-NN	89.0	KMK
FSM	88.5	KMK [64]
Liniowa analiza dysk.	86.4	Stern & Dobnikar [236]
Analiza bayesowska + niezależność cech	86.3	Stern & Dobnikar [236]
IncNet Rot	86.0	KMK
Kwadratowa analiza dysk.	85.8	Stern & Dobnikar [236]
1-NN	85.3	Stern & Dobnikar [236]
ASR (ang. Assistant-R)	85.0	Stern & Dobnikar [236]
Analiza dysk. Fishera	84.5	Stern & Dobnikar [236]
LVQ (ang. Learning vector quantization)	83.2	Stern & Dobnikar [236]
CART (drzewa decyzyjne)	82.7	Stern & Dobnikar [236]
MLP+BP	82.1	Stern & Dobnikar [236]
ASI (ang. Assistant-I)	82.0	Stern & Dobnikar [236]
LFC (ang. Lookahead feature constructor)	81.9	Stern & Dobnikar [236]
RBF (Tooldiag)	79.0	KMK
MLP+BP (Tooldiag)	77.4	KMK

Tabela 7.8: Zapalenie wątroby — porównanie rezultatów.

Sieć IncNet uzyskała poprawność 99.68% na zbiorze treningowym i 99.24% na zbiorze testowym, czyli jedynie 0.12% gorzej niż najlepszy model. Uczenie wynosiło 200000 iteracji. Końcowa sieć składała się z 9 neuronów. Pierwsza podsieć składała się z 2 neuronów, druga z 1 neuronu, a ostatnia z 6 neuronów. Macierze rozrzutu dla danych treningowych i testowych można zobaczyć na rys. 7.23.

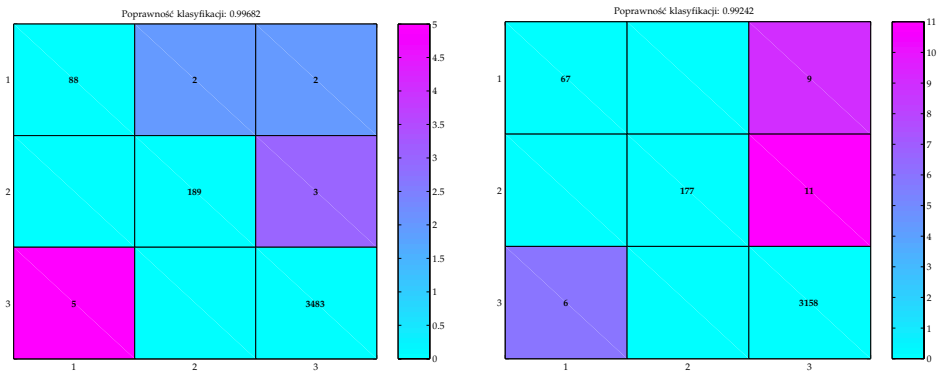
W tabeli 7.10 zostały umieszczone rezultaty otrzymane w oparciu o zbiór treningowy i testowy dla różnych modeli.

7.3. Aproksymacja

Innymi bardzo dobrymi przykładami ilustrującymi efektywność sieci IncNet, są problemy aproksymacji funkcji. W poniższym podrozdziale zostaną przedstawione aproksymacje kilku, najczęściej spotykanych w literaturze funkcji.

Metoda	Poprawność	Źródło wyniku
Logdisc	77.7	Statlog [185]
IncNet	77.6	KMK
DIPOL92	77.6	Statlog [185]
Liniowa analiza dysk.	77.5	Statlog [185], Stern & Dobnikar [236]
SMART	76.8	Statlog [185]
ASI (ang. Assistant-I)	76.6	Stern & Dobnikar [236]
Liniowa analiza Fishera	76.5	Stern & Dobnikar [236]
MLP+BP	76.4	Stern & Dobnikar [236]
MLP+BP	75.2	Statlog [185]
LVQ (ang. Learning vector quantization)	75.8	Stern & Dobnikar [236]
RBF	75.7	Statlog [185]
LFC (ang. Lookahead feature constructor)	75.8	Stern & Dobnikar [236]
Analiza bayesowska + niezależność cech	75.5	Stern & Dobnikar [236]; Statlog
Analiza bayesowska	75.4	Stern & Dobnikar [236]
CART	74.5	Stalog [185]
DB-CART	74.4	Shang & Breiman [232]
ASR (ang. Assistant-R)	74.3	Stern & Dobnikar [236]
CART	72.8	Stern & Dobnikar [236]
C 4.5	73.0	Statlog [185]
Kohonen	72.7	Statlog [185]
kNN	71.9	Stern & Dobnikar [236]
kNN	67.6	Statlog [185]
Kwadratowa analiza dysk.	59.5	Ster& Dobnikar [236]

Tabela 7.9: Choroby cukrzycy — porównanie rezultatów.



Rysunek 7.23: Macierze rozrzutu dla bazy danych chorób tarczycy. Po lewej dla zbioru treningowego, po prawej dla zbioru testowego.

7.3.1. Funkcja Hermita

Pierwszą prezentowaną funkcją jest funkcja Hermita, zdefiniowana przez:

$$f_{her}(x) = 1.1(1 - x + 2x^2)\exp(-1/2x^2) \tag{7.13}$$

Na dane treningowe składa się 40 losowych punktów z przedziału $[-4, 4]$, natomiast zbiór testowy składa się ze 100 losowych wartości, pochodzących z tego samego przedziału. Wszystkie sieci były uczone przez 800 iteracji (tj. prezentacji jednego wektora treningowego).

Tablica 7.11 prezentuje rezultaty uzyskane dla sieci IncNet z funkcjami bicentralnymi (IncNet), jak i z funkcjami Gaussa (IncNet G). Przedstawione są także rezultaty dla sieci RAN-EKF [146, 149] i RAN [204] uzyskanymi przez Kadirkamanathana. Do porównania użyto miary błędu MSE (7.9).

7.3.2. Funkcja Gabora i Girosiego

Problemem aproksymacji obu poniżej zdefiniowanych funkcji, zajmował się Girosi, Jones i Poggio [106]. Funkcja Gabora jest zdefiniowana przez:

$$f_{gab}(x, y) = e^{-||\mathbf{x}||^2}\cos(.75\pi(x + y)) \tag{7.14}$$

a druga (Girosiego) przez:

$$f_{gir}(x, y) = sin(2\pi x) + 4(y - 0.5)^2 \tag{7.15}$$

Proces uczenia jest bardzo trudny, ponieważ opiera się jedynie o 20 wektorów treningowych, losowo wybranych z przedziału $[-1, 1] \times [-1, 1]$ dla funkcji Gabora i z przedziału $[0, 1] \times [0, 1]$ dla funkcji Girosiego. Natomiast na zbiór testowy składa się 10,000 losowych punktów z tych samych przedziałów.

Metoda	Poprawność treningowa	Poprawność testowa	Źródło wyniku
C-MLP2LN rules + ASA	99.9	99.36	KMK
CART	99.8	99.36	Weiss [255]
PVM	99.8	99.33	Weiss [255]
IncNet	99.68	99.24	KMK
MLP init+ a,b opt.	99.5	99.1	KMK
C-MLP2LN rules	99.7	99.0	KMK
Cascade correlation	100.0	98.5	Schiffmann [225]
Local adapt. rates	99.6	98.5	Schiffmann [225]
BP+genetic opt.	99.4	98.4	Schiffmann [225]
Quickprop	99.6	98.3	Schiffmann [225]
RPROP	99.6	98.0	Schiffmann [225]
3-NN, Euclides, 3 features used	98.7	97.9	KMK
1-NN, Euclides, 3 features used	98.4	97.7	KMK
Best backpropagation	99.1	97.6	Schiffmann [225]
1-NN, Euclides, 8 features used	–	97.3	KMK
Bayesian classif.	97.0	96.1	Weiss [255]
BP+conj. gradient	94.6	93.8	Schiffmann [225]
1-NN Manhattan, std data	100	93.8	KMK
default: 250 test errors		92.7	
1-NN Manhattan, raw data	100	92.2	KMK

Tabela 7.10: Choroby tarczycy — porównanie rezultatów.

Tablica 7.12 opisuje modele testowane przez Girosiego i in. [106]. Natomiast tabela 7.13 opisuje rezultaty, uzyskane przy użyciu tych modeli i przy użyciu sieci IncNet z bicentralnymi funkcjami transferu (IncNet), jak i funkcjami bicentralnymi z rotacją (IncNet Rot). Do porównania użyto miary błędu MSE (7.9).

Choć sieć IncNet nie zawsze jest najlepsza, to jednak po uśrednieniu rezultatów, wysuwa się na czoło hierarchii, dzięki swej elastyczności (patrz tablica 7.13). Uzyskane rezultaty pokazują, że funkcje bicentralne z rotacją również mogą być efektywnie wykorzystane przez sieć IncNet, pomimo wprowadzenia $N-1$ dodatkowych parametrów adaptacyjnych.

Dla funkcji Gabora (7.14) końcowa sieć IncNet składała się z 4 neuronów dla bicentralnych funkcji transferu (zmieniając parametry uczenia tak, aby końcowa liczba neuronów była większa, uzyskiwało się gorszy poziom generalizacji) i 6 neuronów dla funkcji bicentralnych z rotacją.

Dla funkcji Girosiego (7.15) sieć IncNet składała się z 8 neuronów dla funkcji bicentralnych i bicentralnych z rotacją.

Przykładowy proces uczenia można prześledzić na rysunku 7.24. Jak widać, na początku sieć dodaje i usuwa neurony, aż do ustalenia ostatecznej

Błąd MSE			
IncNet	IncNet G	RAN-EKF	RAN
0.000225	0.0029	0.0081	0.0225

Tabela 7.11: Aproksymacja funkcji Hermita (7.13).

Model1	$\sum_{i=1}^{20} c_i[e^{-\left(\frac{(x-x_i)^2}{\sigma_1}+\frac{(y-y_i)^2}{\sigma_2}\right)}+e^{-\left(\frac{(x-x_i)^2}{\sigma_2}+\frac{(y-y_i)^2}{\sigma_1}\right)}]$	$\sigma_1=\sigma_2=0.5$
Model2	$\sum_{i=1}^{20} c_i[e^{-\left(\frac{(x-x_i)^2}{\sigma_1}+\frac{(y-y_i)^2}{\sigma_2}\right)}+e^{-\left(\frac{(x-x_i)^2}{\sigma_2}+\frac{(y-y_i)^2}{\sigma_1}\right)}]$	$\sigma_1=10,\sigma_2=0.5$
Model3	$\sum_{i=1}^{20} c_i[e^{\frac{(x-x_i)^2}{\sigma}}+e^{-\frac{(y-y_i)^2}{\sigma}}]$	$\sigma=0.5$
Model4	$\sum_{\alpha=1}^{7} b_\alpha e^{-\frac{(x-t_x^\alpha)^2}{\sigma}}+\sum_{\beta=1}^{7} c_\beta e^{-\frac{(y-t_y^\beta)^2}{\sigma}}$	$\sigma=0.5$
Model5	$\sum_{\alpha=1}^{n} c_\alpha e^{-(\mathbf{W}_\alpha\cdot\mathbf{X}-t_\alpha)^2}$	
Model6	$\sum_{i=1}^{20} c_i[\sigma(x-x_i)+\sigma(y-y_i)]$	
Model7	$\sum_{\alpha=1}^{7} b_\alpha\sigma(x-t_x^\alpha)+\sum_{\beta=1}^{7} c_\beta\sigma(y-t_y^\beta)$	
Model8	$\sum_{\alpha=1}^{n} c_\alpha\sigma(\mathbf{W}_\alpha\cdot\mathbf{X}-t_\alpha)$	

Tabela 7.12: Definicje modeli użytych do aproksymacji funkcji Gabora i Girosiego.

architektury.

7.3.3. Funkcja Sugeno

Do kolejnego testu zostanie użyta funkcja zaproponowana przez Sugeno [237], zdefiniowana przez:

$$f(x,y,z)=(1+x^{0.5}+y^{-1}+z^{-1.5})^2 \tag{7.16}$$

Poniżej zostaną zaprezentowane rezultaty uzyskane dla sieci IncNet z funkcjami bicentralnymi i bicentralnymi z rotacją. Zostaną one porównane z rezultatami uzyskanymi przez Sugeno [237], Kosińskiego i in. [160] i Horikawy i in. [121].

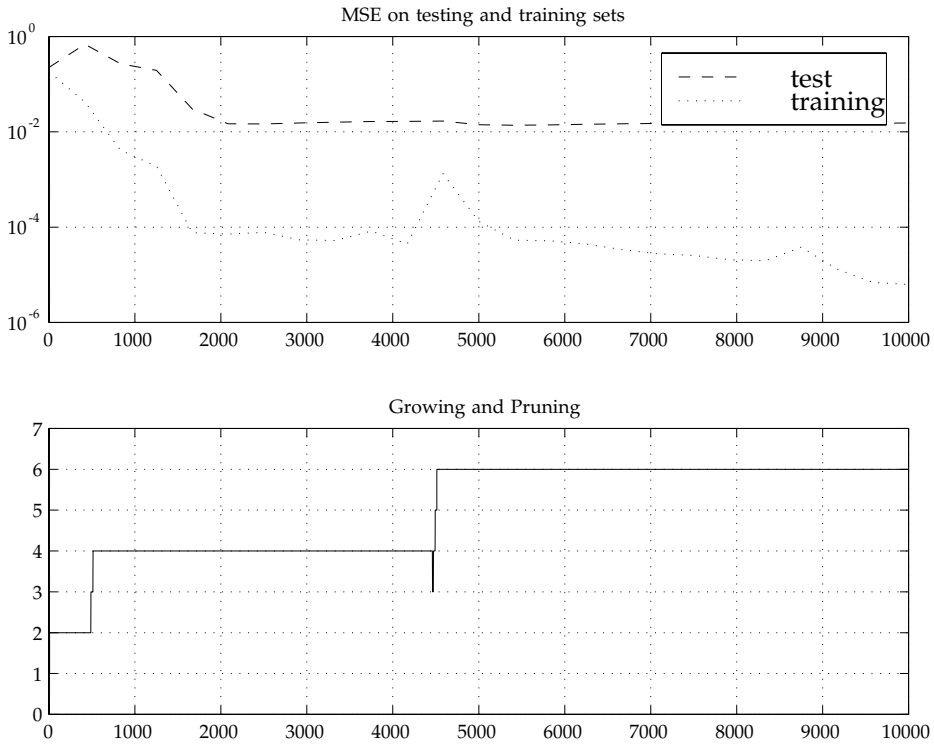
Dane treningowe stanowi 216 punktów o wartościach losowych z przedziału $[1,6]$. Dane testowe składają się ze 125 punktów, również o wartościach losowych, z przedziału $[1.5,5.5]$. Wszystkie testy były wykonywane przy podobnych parametrach początkowych. Do porównań użyto miary błędu APE zdefiniowanej wzorem 7.11.

Końcowa sieć IncNet składała się z 11 neuronów w warstwie ukrytej. Rezultaty zostały zaprezentowane w tabeli 7.14.

Funkcja Girosiego — Błąd MSE treningowy/testowy									
IncNet Rot	IncNet	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8
.00000133	.00000232	.000036	.000067	.000001	.000001	.000170	.000001	.000003	.000743
0.000859	.000082	.011717	.001598	.000007	.000009	.001422	.000015	.000020	.026699

Funkcja Gabora — Błąd MSE treningowy/testowy									
IncNet Rot	IncNet	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8
.000006	.000025	.000000	.000000	.000000	.345423	.000001	.000000	.456822	.000044
0.015316	0.025113	.003818	.344881	67.9523	1.22211	.033964	98.4198	1.39739	.191055

Tabela 7.13: Aproksymacja funkcji Gabora (7.14) i Girosiego (7.15).



Rysunek 7.24: Adaptacja sieci IncNet dla problemu aproksymacji funkcji Sugeno. Błąd MSE dla zbioru treningowego i testowego (u góry). Liczba neuronów (u dołu).

Model	Błąd APE	
	treningowy	testowy
GMDS model Kongo [160]	4.7	5.7
Fuzzy model 1 Sugeno [237]	1.5	2.1
Fuzzy model 2 Sugeno [237]	0.59	3.4
FNN Type 1 Horikawa [121]	0.84	1.22
FNN Type 2 Horikawa [121]	0.73	1.28
FNN Type 3 Horikawa [121]	0.63	1.25
M - Delta model [160]	0.72	0.74
Fuzzy INET [160]	0.18	0.24
Fuzzy VINET [160]	0.076	0.18
IncNet	0.119	0.122
IncNet Rot	0.053	0.061

Tabela 7.14: Porównanie rezultatów aproksymacji funkcji Sugeno (7.16).

Rozdział 8

Zakończenie

Materiał książki, którą mają Państwo w rękach, można by rozbudować o jeszcze wiele pokrewnych tematów, chociażby rozdział „Support vector machines". Jednakże podejmowana w książce tematyka jest tak szeroka, że jej wyczerpanie w pojedynczej publikacji nie jest możliwe. Sporo metod przedstawionych w poszczególnych rozdziałach ma naturę bardzo ogólną i może być użyta z powodzeniem w innych niż pokazano modelach, bądź w połączeniu z innymi systemami adaptacyjnymi. Na przykład informacje o funkcjach transferu przedstawione w rozdziale pierwszym, choć nakreślone głównie w kontekście sieci *feed forward*, z powodzeniem mogą być stosowane do sieci z rekurencją. Z kolei użycie niektórych funkcji transferu wraz z metodami regularyzacji może dać interesujące metody selekcji cech, jak i czasem agregacji cech. W następnym kroku można by sprawdzić jak różne zestawy cech będą działały dla różnych modeli, być może także dla niektórych komitetów modeli opisanych w jednym z rozdziałów.

Pomimo, że różnych metod przedstawiono stosunkowo wiele, to jednak analizując przeróżne wyniki uzyskane za ich pomocą, wyraźnie rysuje się związek pomiędzy złożonością modelu adaptacyjnego i złożonością problemu. Modele, aby działać skutecznie, powinny być wyposażone w mechanizmy, które automatycznie dobiorą parametry do danego problemu, czyli jego złożoności.

Myślę, że w stosunkowo niedługim czasie powinniśmy uzyskać na tyle ogólne metody, które będą nie tylko w ogóle się uczyć, ale będą tak naprawdę meta-modelami zarządzającymi grupami metod różnych typów. Meta-modele odpowiednio sterowane powinny prowadzić do automatycznej ekstrakcji rozwiązań możliwie bliskich do optymalnych i jednocześnie nie gorszych niż te, które jesteśmy w stanie sami znaleźć ręcznie. Trzeba jednak zwrócić uwagę na fakt, iż ręczne poszukiwanie ostatecznego modelu (gdy ekspert dobiera odpowiedni zestaw cech, dobiera najwłaściwszy model i jego parametry) wymaga wielogodzinnych analiz. Takie meta-modele dadzą możliwość uzyskiwania nie-

typowych rozwiązań poprzez nie tylko automatyczny dobór parametrów, lecz również dobór najbardziej odpowiedniego typu algorytmu adaptacyjnego dla danego problemu. Końcowym wynikiem działania takiego meta-modelu nie musi być tylko pojedynczy model, lecz kilka modeli, z których każdy może komplementarnie uzupełniać pozostałe nie tylko w sensie dokładności klasyfikacji, ale przede wszystkim rodzajem wiedzy, jaki dany model prezentuje (sieci neuronowe, reguły logiczne, prototypowy opis wiedzy, istotności poszczególnych atrybutów, etc.).

Bibliografia

[1] R. Adamczak, W. Duch, N. Jankowski. New developments in the feature space mapping model. *Third Conference on Neural Networks and Their Applications*, s. 65–70, Kule, Poland, 1997.

[2] J. Allison. Multiquadratic radial basis functions for representing multidimensional high energy physics data. *Computer Physics Communications*, 77:377–395, 1993.

[3] P. Allison. Multiple imputation for missing data: A cautionary tale, 2000.

[4] H. Almuallim, T. G. Dietterich. Efficient algorithms for identifying relevant features. *Proceedings of the Ninth Canadian Conference on Artificial Intelligence*, s. 38–45, Vancouver, 1992. Morgan Kaufmann.

[5] J. A. Anderson. Logistic discrimination. *Handbook of statistics 2: Classification, pattern recognition and reduction of dimensionality*, s. 169–191. North Holland, Amsterdam, 1982.

[6] J. A. Anderson. *An Introduction to Neural Networks*. Bradford Book, 1995.

[7] A. R. Barron. Universal approximation bounds for superpositions of a sigmoid function. *IEEE Transaction on Neural Networks*, 39:930–945, 1993.

[8] E. Bauer, Ron Kohavi. An empirical comparison of voting classification algorithms bagging boosting and variants. *Machine Learning*, 36:106–142, 1999.

[9] Y. S. Ben-Porath, N. Sherwood. *The MMPI-2 content component scales: Development, psychometric characteristics, and clinical applications*. University of Minnesota Press, Minneapolis, 1993.

[10] Kristin P. Bennett, J. Blue. A support vector machine approach to decision trees. Raport techniczny, Rensselaer Polytechnic Institute, Troy, NY, 1997.

[11] C. M. Bishop. Improving the generalization properties of radial basis function neural networks. *Neural Computation*, 3(4):579–588, 1991.

[12] C. M. Bishop. Training with noise is equivalent to Tikhonov regularization. *Neural Computation*, 7(1):108–116, 1991.

[13] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[14] C. M. Bishop, M. Svensén, C. K. I. Williams. EM optimization of latent-variable density models. D. S. Touretzky, M. C. Mozer, M. E. Hasselmo, redaktorzy, *Advances in Neural Information Processing Systems*, wolumen 8, Cambridge, MA, 1996. MIT Press.

[15] L. Bobrowski. Piecewise–linear classifiers, formal neurons and separability of learning sets. *Proceedings of ICPR*, s. 224–228, 1996.

[16] L. Bobrowski. Generowanie sieci neuropodobnych oraz drzew decyzyjnych w oparciu o kryterium dipolowe. *Symulacja w badaniach i rozwoju*, Jelenia Góra, 1997.

[17] L. Bobrowski, M. Krętowska, M. Krętowski. Design of neural classifying networks by using dipolar criterions. *Third Conference on Neural Networks and Their Applications*, Kule, Poland, 1997.

[18] L. Bolc, J. Zaremba. *Wprowadzenie do uczenia się maszyn*. Akademicka Oficyna Wydawnicza, Warszawa, 1992.

[19] L. Bottou, V. Vapnik. Local learning algorithms. *Neural Computation*, 4(6):888–900, 1992.

[20] L Breiman. Arcing classifiers. Raport techniczny, Statistics Department, University of California, Berkeley, 1996.

[21] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[22] L Breiman. Stacked regressions. *Machine Learning*, 24:49–64, 1996.

[23] L Breiman. Arcing the edge. Raport techniczny, Statistics Department, University of California, Berkeley, 1997.

[24] L. Breiman. Bias-variance, regularization, instability and stabilization. C. M. Bishop, redaktor, *Neural Networks and Machine Learning*, s. 27–56. Springer-Verlag, 1998.

[25] L. Breiman, J. H. Friedman, A. Olshen, C. J. Stone. *Classification and regression trees*. Wadsworth, Belmont, CA, 1984.

[26] J. S. Bridle. Probabilistic interpretation of feedforward classification network outputs with relationships to statistical pattern recognition. F. Fogelman Soulié, J. Hérault, redaktorzy, *Neurocomputing: Algorithms, Architectures and Applications*, s. 227–236. Springer-Verlag, New York, 1990.

[27] D. S. Broomhead, D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[28] I. Bruha. From machine learning to knowledge discovery: Preprocessing and postprocessing. *Machine Learning and Applications. Workshop on Preprocessing and Postprocessing on machine learning and data mining: theoretical aspects and applications*, s. 1–17, Greece, 1999.

[29] J. M. Buhman, N. Tishby. A statistical learning theory of data clustering. C. M. Bishop, redaktor, *Neural Networks and Machine Learning*, s. 57–68. Springer-Verlag, 1998.

[30] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121—167, 1998.

[31] N. Burgess. A constructive algorithm that converges for real-valued input patterns. *International journal of neural systems*, 5(1):59–66, 1994.

[32] J. N. Butcher, J. R. Graham, C. L. Williams, Y. Ben-Porath. *Development and use for the MMPI-2 Content Scales*. University of Minnesota Press, Minneapolis, University of Minnesota Press.

[33] J. N. Butcher, C. L. Williams. *Essential of MMPI-2 and MMPI-A interpretation*. University of Minnesota Press, Minneapolis, 1992.

[34] J. N. Buther, W. G. Dahlstrom, J. R. Graham, A. Tellegen, B. Kaem-mer. *Minnesota Multiphasic Personality Inventory-2 (MMPI-2): Manual for administration and scoring*. University of Minnesota Press, Minneapolis, 1989.

[35] Cacoullos. Estimation of multivariate density. *Annals of Institute of Statistical Mathematics*, 18:179–189, 1966.

[36] C. Campbell. Constructive learning techniques for designing neural network systems, 1997.

[37] C. Campbell, C. V. Perez. Target switching algorithm: a constructive learning procedure for feed-forward neural networks. *Neural Networks*, s. 1221–1240, 195.

[38] C. Campbell, C. V. Perez. Constructing feed-forward neural networks for binary classification task. *European symposium on artificial neural networks*, s. 241–246, Brussels, 1995.

[39] J. V. Candy. *Signal processing: The model based approach*. McGraw-Hill, New York, 1986.

[40] C.-C. Chang, C.-J. Lin. Training $\nu-$support vector classifiers: Theory and algorithms. *Neural Computation*, 13(9):2119–2147, 2001.

[41] C.-C. Chang, C.-J. Lin. Training $\nu-$support vector regression: Theory and algorithms. *Neural Computation*, 14(8):1959–1977, 2002.

[42] Mu-Song Chen. *Analyses and Design of Multi-Layer Perceptron Using Polynomial Basis Functions*. Praca doktorska, The University of Texas at Arlington, 1991.

[43] S. Chen, S. A. Billings, W. Luo. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control*, 50(5):1873–1896, 1989.

[44] S. Chen, C. F. N. Cowan, P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transaction on Neural Networks*, 2(2):302–309, 1991.

[45] V. Cherkassky, F. Mulier. *Learning from data*. Adaptive and learning systems for signal processing, communications and control. John Wiley & Sons, Inc., New York, 1998.

[46] P. Cichosz. *Systemy uczące się*. Wydawnictwa Naukowo–Techniczne, Warszawa, 2000.

[47] C. Cortes, V. Vapnik. Support vector networks. *Machine Learning*, 20:273—-297, 1995.

[48] M. Cottrell, B. Girard, Y. Girard, M. Mangeas, C. Muller. Neural modeling for time series: a statistical stepwise method for weight elimination. *IEEE Transaction on Neural Networks*, 6(6):1355–1364, 1995.

[49] T. M. Cover, P. E. Hart. Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 13(1):21–27, 1967.

[50] Y. Le Cun, B. Boser, J. Denker, D. Henderson, W. Hubbart, L. Jackel. Handwritten digit recognition with a back-propagation network. D. S. Touretzky, redaktor, *Advances in Neural Information Processing Systems 2*, s. 396–404, San Mateo, CA, 1990. Morgan Kauffman.

[51] Y. Le Cun, J. Denker, S. Solla. Optimal brain damage. D. S. Touretzky, redaktor, *Advances in Neural Information Processing Systems 2*, San Mateo, CA, 1990. Morgan Kauffman.

[52] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[53] M. Dash, H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3), 1997.

[54] T. Denoeoux, R. Lengellé. Initializing back propagation networks with prototypes. *Neural Networks*, 6(3):351–363, 1993.

[55] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging boosting and randomization. *Machine Learning*, s. 1–22, 1999.

[56] G. Dorffner. A unified framework for MLPs and RBFNs: Introducing conic section function networks. *Cybernetics and Systems*, 25(4):511–554, 1994.

[57] W. Duch, R. Adamczak, G. H. F. Diercksen. Distance-based multilayer perceptrons. *International Conference on Computational Intelligence for Modelling Control and Automation*, s. 75–80, Vienna, Austria, 1999.

[58] W. Duch, R. Adamczak, G. H. F. Diercksen. Neural networks in non-euclidean spaces. *Neural Processing Letters*, 10:1–10, 1999.

[59] W. Duch, R. Adamczak, K. Grąbczewski. Extraction of crisp logical rules using constrained backpropagation networks. *International Conference on Artificial Neural Networks (ICANN'97)*, s. 2384–2389, 1997.

[60] W. Duch, R. Adamczak, K. Grąbczewski. Extraction of logical rules from backpropagation networks. *Neural Processing Letters*, 7:1–9, 1998.

[61] W. Duch, R. Adamczak, K. Grąbczewski. Methodology of extraction, optimization and application of logical rules. *Intelligent Information Systems VIII*, s. 22–31, Ustroń, Poland, 1999.

[62] W. Duch, R. Adamczak, N. Jankowski. Initialization of adaptive parameters in density networks. *Third Conference on Neural Networks and Their Applications*, s. 99–104, Kule, Poland, 1997.

[63] W. Duch, R. Adamczak, N. Jankowski. Initialization and optimization of multilayered perceptrons. *Third Conference on Neural Networks and Their Applications*, s. 105–110, Kule, Poland, 1997.

[64] W. Duch, R. Adamczak, N. Jankowski. New developments in the feature space mapping model. Raport techniczny CIL-KMK-2/97, Computational Intelligence Lab, DCM NCU, Toruń, Poland, 1997. (long version).

[65] W. Duch, G. H. F. Diercksen. Feature space mapping as a universal adaptive system. *Computer Physics Communications*, 87:341–371, 1995.

[66] W. Duch, K. Grudziński. Search and global minimization in similarity-based methods. *International Joint Conference on Neural Networks*, s. 742, Washington, 1999.

[67] W. Duch, K. Grudziński. The weighted k-NN with selection of features and its neural realization. *4th Conference on Neural Networks and Their Applications*, s. 191–196, Zakopane, Poland, 1999.

[68] W. Duch, Ł. Itert, K. Grudziński. Competent undemocratic committees. L. Rutkowski, J. Kacprzyk, redaktorzy, *Neural Networks and Soft Computing. Proceedings of the 6th International Conference on Neural Networks and Soft Computing (ICNNSC)*, Advances in Soft Computing, s. 412–417, Zakopane, Poland, 2002. Springer-Verlag.

[69] W. Duch, N. Jankowski. New neural transfer functions. *Journal of Applied Mathematics and Computer Science*, 7(3):639–658, 1997.

[70] W. Duch, N. Jankowski. Survey of neural transfer functions. *Neural Computing Surveys*, 2:163–212, 1999. (PDF).

[71] W. Duch, N. Jankowski, A. Naud, R. Adamczak. Feature space mapping: a neurofuzzy network for system identification. *Proceedings of the European Symposium on Artificial Neural Networks*, s. 221–224, Helsinki, 1995.

[72] W. Duch, T. Kucharski, J. Gomuła, R. Adamczak. *Metody uczenia maszynowego w analizie danych psychometrycznych. Zastosowanie do wielowymiarowego kwestionariusza osobowości MMPI–WISKAD*. Toruń, Poland, 1999.

[73] Włodzisław Duch, Józef Korbicz, Leszek Rutkowski, Ryszard Tadeusiewicz, redaktorzy. *Sieci Neuronowe*. Biocybernetyka i inżynieria biomedyczna. Akademicka Oficyna Wydawnicza, Warszawa, 2000.

[74] R. O. Duda, P. E. Hart. *Patter Classification and Scene Analysis*. Wiley, 1973.

[75] R. O. Duda, P. E. Hart, D. G. Stork. *Patter Classification and Scene Analysis*. Wiley, wydanie 2, 1997.

[76] N. Dyn. Interpolation and approximation by radial and related functions. C. K. Chiu, L. L. Schumaker, J. D. Watts, redaktorzy, *Approximation Theory VI*. Academic Press, San Diego, 1989.

[77] S. E. Fahlman. The recurrent cascade-correlation architecture. Raport techniczny, Carnegie Mellon University, School of Computer Science, 1991.

[78] S. E. Fahlman, C. Lebiere. The cascade-correlation learning architecture. D. S. Touretzky, redaktor, *Advances in Neural Information Processing Systems 2*, s. 524–532. Morgan Kaufmann, 1990.

[79] S. E. Fahlman, C. Lebiere. The cascade-correlation learning architecture. Raport techniczny CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990.

[80] M. Fernández, C. Hernández. How to select the inputs for a multilayer feedforward by using the training set. *5th International Work Conference on Artificial an Natural Neural Networks*, s. 477–486, Alicante, Spain, 1999.

[81] E. Fiesler. Comparative bibliography of ontogenic neural networks. *Proceedings of the International Conference on Artificial Neural Networks*, s. 793–796, 1994.

[82] W. Finnoff, F. Hergert, H. G. Zimmermann. Improving model detection by nonconvergent methods. *Neural Networks*, 6(6):771–783, 1993.

[83] W. Finnoff, H. G. Zimmermann. Detecting structure in small datasets by network fitting under complexity constrains. *Proceedings of the second annual workshop on computational learning theory and natural learning systems*, Berkeley, CA, 1991.

[84] R. Franke. Scattered data interpolation: test of some methods. *Math Computation*, 38:181–200, 1982.

[85] M. Frean. *Small nets and short paths: optimizing neural computation*. Praca doktorska, Center for cognitive science. University of Edinburgh, 1990.

[86] M. Frean. The upstart algorithm: a method for constructing and training feedforward neural networks. *Neural Computation*, 2(2):198–209, 1990.

[87] N. de Freitas, M. Milo, P. Clarkson, M. Niranjan, A. Gee. Sequential support vector machines. 1999.

[88] N. de Freitas, M. Niranjan, A. Gee. Hierarchical bayesian-kalman models for regularization and ard in sequential learning. Raport techniczny CUED/F-INFENG/TR 307, Cambridge University Engineering Department, England, 1998.

[89] Y. Freund, R. E. Schapire. Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference*, 1996.

[90] Y. Freund, R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[91] J. H. Friedman. Multivariate adaptive regression splines (with discussion). *Ann. Stat.*, 19:1–141, 1991.

[92] J. H. Friedman. Flexible metric nearest neighbor classification. Raport techniczny, Department of Statistics, Stanford University, 1994.

[93] J. H. Friedman. Greedy function approximation: a gradient boosting machine. http://www-stat.stanford.edu/ jhf/ftp/trebst.ps, 1999.

[94] B. Fritzke. Fast learning with incremental RBF networks. *Neural Processing Letters*, 1(1):2–5, 1994.

[95] B. Fritzke. Supervised learning with growing cell structures. J. D. Cowan, G. Tesauro, J. Alspector, redaktorzy, *Advances in Neural Information Processing Systems 6*, s. 255–262, San Mateo, CA, 1994. Morgan Kaufman.

[96] B. Fritzke. A growing neural gas network learns topologies. G. Tesauro, D. S. Touretzky, T. K. Leen, redaktorzy, *Advances in Neural Information Processing Systems 7*, s. 625–632, Cambridge, MA, 1995. MIT Press.

[97] B. Fritzke. A self-organizing network that can follow non-stationary distributions. W. Gerstner, A. Germond, M. Hasler, J. Nicoud, redaktorzy, *7th International Conference on Artificial Neural Networks*, s. 613–618, Lausanne, Switzerland, 1997. Springer-Verlag.

[98] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, San Diego, 1972.

[99] S. Fusi, M. Mattia. Collective behavior of networks with linear (VLSI) integrate and fire neurons. *Neural Computation*, 1997.

[100] S. I. Gallant. Optimal linear discriminants. *IEEE Proceedings of 8th Conference on Pattern Recognition*, s. 849–852, 1986.

[101] S. I. Gallant. Three constructive algorithms for network learning. *Proceedings of the eight annual conference of the cognitive science society*, s. 652–660, Hillsdale, NJ, 1986. Lawrence Erlbaum.

[102] S. I. Gallant. *Neural networks learning and expert systems*. MIT Press, 1993.

[103] Zoubin Ghahramani, Michael I. Jordan. Supervised learning from incomplete data via an EM approach. Jack D. Cowan, Gerald Tesauro, Joshua Alspector, redaktorzy, *Advances in Neural Information Processing Systems*, wolumen 6, s. 120–127. Morgan Kaufmann Publishers, Inc., 1994.

[104] B. G. Giraud, A. Lapedes, L. C. Liu, J. C. Lemm. Lorentzian neural nets. *Neural Networks*, 8(5):757–767, 1995.

[105] F. Girosi. An equivalence between sparse approximation and support vector machines. *Neural Computation*, 10(6):1455–1480, 1998.

[106] F. Girosi, M. Jones, T. Poggio. Priors stabilizers and basis functions: From regularization to radial, tensor and additive splines. Raport techniczny, MIT, Cambridge, Massachusetts, 1993.

[107] F. Girosi, M. Jones, T. Poggio. Regularization theory and neural networks. *Neural Computation*, 7:219–269, 1995.

[108] F. Girosi, T. Poggio. Networks and the best approximation property. AI Lab. Memo, MIT, 1989.

[109] G. H. Golub, M. Heath, G. Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–213, 1979.

[110] K. Grąbczewski, W. Duch. A general purpose separability criterion for classification systems. *4th Conference on Neural Networks and Their Applications*, s. 203–208, Zakopane, Poland, 1999.

[111] K. Grąbczewski, Włodzisław Duch. The separability of split value criterion. L. Rutkowski, R. Tadeusiewicz, redaktorzy, *Neural Networks and Soft Computing*, s. 202–208, Zakopane, Poland, 2000.

[112] H. J. Hamilton, N. Shan, N. Cercone. RIAC: a rule induction algorithm based on approximate classification. Raport techniczny CS 96-06, Regina University, 1996.

[113] E. Hartman, J. D. Keeler. Predicting the future: Advantages of semilocal units. *Neural Computation*, 3(4):566–578, 1991.

[114] E. J. Hartman, J. D. Keeler, J. M. Kowalski. Layered neural networks with gaussian hidden units as universal approximations. *Neural Computation*, 2(2):210–215, 1990.

[115] B. Hassibi, D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in Neural Information Processing Systems 5*, s. 164–171. Morgan Kaufmann, 1993.

[116] B. Hassibi, D. G. Stork, G. J. Wolff. Optimal brain surgeon and general network pruning. Raport techniczny CRC-TR-9235, RICOH California Research Center, Menlo Park, CA, 1992.

[117] T. Hastie, R. Tibshirani. Discriminant adaptive nearest neighbor classification. *IEEE PAMI 18*, s. 607–616, 1996.

[118] S. Haykin. *Neural Networks - A Comprehensive Foundation*. Maxwell MacMillian Int., New York, 1994.

[119] S. Haykin. *Adaptive filter theory*. Printice-Hall international, New Jersey, USA, 1996.

[120] G. E. Hinton. Learning translation invariant recognition in massively parallel networks. J. W. de Bakker, A. J. Nijman, P. C. Treleaven, redaktorzy, *Proceedings PARLE Conference on Parallel Architectures and Languages Europe*, s. 1–13, Berlin, 1987. Springer-Verlag.

[121] S. Horikawa, Takeshi Furuhashi, Yoshiki Uchikawa. On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm. *IEEE Transaction on Neural Networks*, 3(5):801–806, 1992.

[122] K. Hornik, M. Stinchcombe, H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[123] K. Hornik, M. Stinchcombe, H. White, P. Auer. Degree of approximation results for feedforward networks approximating unknown mappings and their derivatives. *Neural Computation*, 6(6):1262–1275, 1994.

[124] Nicholas J. Horton, Stuart R. LIPSITZ. Multiple imputation in practice: Comparison of software packages for regression models with missing variables.

[125] T. Hrycej. *Modular neural networks*. Wiley, New york, 1992.

[126] C. C. Hsu, D. Gubovic, M. E. Zaghloul, H. H. Szu. Chaotic neuron models and their VLSI implementations. *IEEE Transaction on Neural Networks*, 7(6):1339–1350, 1996.

[127] C.-W. Hsu, C.-J. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transaction on Neural Networks*, 13:415–425, 2002.

[128] J. M. Hutchinson. *A Radial Basis Function Approach to Financial Time Series Analysis*. Praca doktorska, MIT, Cambridge, MA, 1993.

[129] J. M. Hutchinson, A. W. Lo, T. Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *Journal of Finance*, 49(3):851–889, 1994.

[130] Institute of Parallel and Distributed High-Performance Systems (IPVR). *Stuttgart Neural Networks Simulator (SNNS)*. http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/snns.html.

[131] R. A. Jacobs, Jordan M. L., S. J. Nowlan, J. E. Hinton. Adaptive mixtures of local exports. *Neural Computation*, 79(3), 1991.

[132] K. Jajuga. *Statystyczna analiza wielowymiarowa*. Wydawnictwo Naukowe PWN, Warszawa, 1993.

[133] N. Jankowski. Controlling the structure of neural networks that grow and shrink. *Second International Conference on Cognitive and Neural Systems*, Boston, USA, 1998.

[134] N. Jankowski. Approximation and classification in medicine with IncNet neural networks. *Machine Learning and Applications. Workshop on Machine Learning in Medical Applications*, s. 53–58, Chania, Greece, 1999. (PDF).

[135] N. Jankowski. Approximation with RBF-type neural networks using flexible local and semi-local transfer functions. *4th Conference on Neural Networks and Their Applications*, s. 77–82, Zakopane, Poland, 1999. (PDF).

[136] N. Jankowski. Flexible transfer functions with ontogenic neural. Raport techniczny, Computational Intelligence Lab, DCM NCU, Toruń, Poland, 1999. (PDF).

[137] N. Jankowski. *Ontogenic neural networks and their applications to classification of medical data*. Praca doktorska, Department of Computer Methods, Nicholas Copernicus University, Toruń, Poland, 1999. (PDF).

[138] N. Jankowski, K. Grąbczewki. Toward optimal SVM. *The Third IASTED International Conference on Artificial Intelligence and Applications*, 2003. submitted.

[139] N. Jankowski, K. Grąbczewski, W. Duch. *GhostMiner 2.0*. FQS Poland, 2003.

[140] N. Jankowski, V. Kadirkamanathan. Statistical control of RBF-like networks for classification. *7th International Conference on Artificial Neural Networks*, s. 385–390, Lausanne, Switzerland, 1997. Springer-Verlag.

[141] N. Jankowski, V. Kadirkamanathan. Statistical control of growing and pruning in RBF-like neural networks. *Third Conference on Neural Networks and Their Applications*, s. 663–670, Kule, Poland, 1997.

[142] T. Joachims. Advances in kernel methods — support vector learning. rozdział Making large-scale SVM learning practical. MIT Press, Cambridge, MA., 1998.

[143] M. Jordan. Why the logistic function? A tutorial discussion on probabilities and neural networks. Raport techniczny 9503, Computational Cognitive Science, MIT, Cambridge, MA, 1995.

[144] T. Kacprzak, K. Ślot. *Sieci neuronowe komórkowe. Teoria, projekty, zastosowanie*. Państwowe Wydawnictwa Naukowe, 1995.

[145] V. Kadirkamanathan. *Sequential learning in artificial neural networks*. Praca doktorska, Cambridge University Engineering Department, 1991.

[146] V. Kadirkamanathan. A statistical inference based growth criterion for the RBF networks. Vlontzos, redaktor, *Proceedings of the IEEE. Workshop on Neural Networks for Signal Processing*, s. 12–21, New York, 1994.

[147] V. Kadirkamanathan, M. Niranjan. Nonlinear adaptive filtering in nonstationary environments. *Proceedings of the international conference on acoustic, speech and signal processing*, Toronto, 1991.

[148] V. Kadirkamanathan, M. Niranjan. Application of an architecturally dynamic network for speech pattern classification. *Proceedings of the Institute of Acoustics*, 14:343–350, 1992.

[149] V. Kadirkamanathan, M. Niranjan. A function estimation approach to sequential learning with neural networks. *Neural Computation*, 5(6):954–975, 1993.

[150] V. Kadirkamanathan, M. Niranjan, F. Fallside. Sequential adaptation of radial basis function neural networks and its application to time-series prediction. D. S. Touretzky, redaktor, *Advances in Neural Information Processing Systems*, wolumen 2. Morgan Kaufmann, 1990.

[151] N. Kasabov. *Evolving Connectionist Machines (Methods and Applications for Modeling and Knowledge Discovery in Life Sciences and Engineering)*. Springer Verlag, London, 2002.

[152] T. Kasahara, M. Nakagawa. A study of association model with periodic chaos neurons. *Journal of Physical Society*, 64:4964–4977, 1995.

[153] S. S. Keerthi, E. G. Gilbert. Convergence of a generalized SMO algorithm for svm classifier design. *Machine Learning*, 46:351–360, 2002.

[154] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, K. R. K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13:637–649, 2001.

[155] M. J. Kirby, R. Miranda. Circular nodes in neural networks. *Neural Computations*, 8(2):390–402, 1996.

[156] K. Kobayasji. On the capacity of neuron with a non-monotone output function. *Network*, 2:237–243, 1991.

[157] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[158] T. Kohonen. *Self-organizing maps*. Springer-Verlag, Heidelberg Berlin, 1995.

[159] J. Korbicz, A. Obuchowicz, D. Uciński. *Sztuczne sieci neuronowe. Podstawy i zastosowania*. Akademicka Oficyna Wydawnicza, Warszawa, 1994.

[160] W. Kosiński, M. Weigl. Mapping neural networks and fuzzy inference systems for approximation of multivariate function. E. Kącki, redaktor, *System Modeling Control, Artificial Neural Networks and Their Applications*, wolumen 3, s. 60–65, Łódź, Poland, 1995.

[161] B. Kosko. *Neural Networks and Fuzzy Systems*. Prentice Hall International, 1992.

[162] P. R. Krishnaiah, L. N. Kanal. *Handbook of statistics 2: Classification, pattern recognition and reduction of dimensionality*. North Holland, Amsterdam, 1982.

[163] V. Kurkova. Approximation of functions by perceptron networks with bounded number of hidden units. *Neural Networks*, 8(5):745–750, 1995.

[164] V. Kurkova, P. C. Kainen, V. Kreinovich. Estimates of the number of hidden units and variation with respect to half-spaces. *Neural Networks*, 10(6):1061–1068, 1997.

[165] H. Leung, S. Haykin. Rational neural networks. *Neural Computation*, 5(6):928–938, 1993.

[166] S. P. Liao, H.-T. Lin, C. J. Lin. A note on the decomposition methods for support vector regression. *Neural Computation*, 14:1267–1281, 2002.

[167] W. A. Light. Some aspects of radial basis function approximation. S. P. Singh, redaktor, *Approximation theory, spline functions and applications*, wolumen 256, s. 163–190. Kluwer Academic Publishers, Boston, MA, 1992.

[168] C. J. Lin. Linear convergence of a decomposition method for support vector machines. Raport techniczny, Department of Computer Science and Information Engineering, National Taiwan University, 2001. (http://www.csie.ntu.edu.tw/˜cjlin/papers/linearconv.pdf).

[169] C. J. Lin. Asymptotic convergence of an smo algorithm without any assumptions. *IEEE Transaction on Neural Networks*, 13:248–250, 2002.

[170] C. J. Lin. A formal analysis of stopping criteria of decomposition methods for support vector machines. *IEEE Transaction on Neural Networks*, 13:415–425, 2002.

[171] K. M. Lin, C. J. Lin. A study on reduced support vector machines. (http://www.csie.ntu.edu.tw/˜cjlin/papers/rsvmTEX.pdf), 2002.

[172] D. Lowe. Adaptive radial basis function nonlinearities, and the problem of generalization. *1st IEE International Conference on Artificial Neural Networks*, s. 171–175, London, UK, 1989.

[173] D. Lowe. On the iterative inversion of RBF networks: A statistical interpretation. *2nd IEE International Conference on Artificial Neural Networks*, s. 29–33, London, UK, 1991.

[174] D. Lowe. Novel "topographic" nonlinear. *3rd IEE International Conference on Artificial Neural Networks*, s. 29–33, London, UK, 1993.

[175] D. Lowe. Radial basis function networks. M. A. Arbib, redaktor, *The handbook of brain theory and neural networks*. MIT Press, Cambridge, MA, 1995.

[176] W. Maas. Lower bounds for the computational power of networks of spiking neurons. *Neural Computations*, 8:1–40, 1996.

[177] R. Maclin. Boosting classifiers regionally. *Proceeding of AAAI*, 1998.

[178] R. Maclin, D. Opitz. An empirical evaluation of bagging and boosting. *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, s. 546—-551, 1997.

[179] O. L. Mangasarian, W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23(5):1–18, 1990.

[180] W. S. McCulloch, W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[181] R. Meir, G. Rätsch. Advanced lectures on machine learning, LNCS. rozdział An introduction to boosting and leveraging, s. 119–184. Springer, 2003.

[182] C. J. Merz, P. M. Murphy. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html, (UCI).

[183] M. Mezard, J. P. Nadal. Learning in feedforward layered networks: the tiling algorithm. *Journal of physics*, 22:2191–2203, 1989.

[184] R. Michalski. Concept learning and natural induction. *Machine Learning and Applications*, Chania, Greece, 1999.

[185] D. Michie, D. J. Spiegelhalter, C. C. Taylor. *Machine learning, neural and statistical classification*. Elis Horwood, London, 1994.

[186] T. Miki, M. Shimono, T. Yamakawa. A chaos hardware unit employing the peak point modulation. *Proceedings of the International Symposium on Nonlinear Theory and its Applications*, s. 25–30, Las Vegas, Nevada, 1995.

[187] T. Mitchell. *Machine learning*. McGraw Hill, 1997.

[188] J. Moody, C. J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[189] M. Morita. Associative memory with nonmonotone dynamics. *Neural Networks*, 6:115–126, 1993.

[190] B. A. Murtagh, M. A. Saunders. Minos 5.4 user's guide. Raport techniczny SOL 83.20, Stanford University, 1993.

[191] M. T. Musavi, R. J. Bryant, M. Qiao, M. T. Davisson, E. C. Akeson, B. D. French. Mouse chromosome classification by radial basis function networks with fast orthogonal search. *Neural Networks*, 11(4):769–777, 1998.

[192] M. Nakagawa. An artificial neuron model with a periodic activation function. *Journal of Physical Society*, 64:1023–1031, 1995.

[193] N.J. Nillson. *Learning machines: Foundations of trainable pattern classifying systems*. McGraw-Hill, New York, 1965.

[194] M Niranjan, F. Fallside. Neural networks and radial basis functions in classifying static speech patterns. *Computer speech and language*, 4:275–289, 1990.

[195] P. Niyogi, F. Girosi. On the relationship between generalization error, hypothesis complexity, and sample complexity for radial basis functions. *Neural Computation*, 8(4):819–842, 1996.

[196] S. J. Nowlan, G. E. Hinton. Simplifying neural networks by soft weight sharing. *Neural Computation*, 4(4):473–493, 1992.

[197] M. Orr. Introduction to radial basis function networks. Raport techniczny, Centre for Cognitive Science, University of Edinburgh, 1996.

[198] S. Osowski. *Sieci neuronowe*. Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 1996.

[199] S. Osowski. *Sieci neuronowe w ujęciu algorytmicznym*. Wydawnictwa Naukowo–Techniczne, Warszawa, 1996.

[200] E. Osuna, R. Freund, F. Girosi. Training support vector machines: An application to face detection. *In Proceedings of CVPR'97*, s. 130–136, New York, NY, 1997. IEEE.

[201] Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley, Reading, MA, 1989.

[202] R. Parekh, J. Yang, V. Honavar. Constructive neural-network learning algorithms for pattern classification. *IEEE Transaction on Neural Networks*, 11(2):436–451, 2000.

[203] J. Park, I. W. Sandberg. Universal approximation using radial basis function networks. *Neural Computation*, 3(2):246–257, 1991.

[204] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3:213–225, 1991.

[205] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. B. Sch.olkopf, C. J. C. Burges, A. J. Smola, redaktorzy, *Advances in Kernel Methods — Support Vector Learning*. MIT Press, Cambridge, MA., 1998.

[206] T. Poggio. Learning sparse representations for vision. *Second International Conference on Cognitive and Neural Systems*, Boston, USA, 1998.

[207] T. Poggio, S. Edelman. A network that learns to recognize three three-dimentional objects. *Nature*, 343:263–266, 1990.

[208] T. Poggio, F. Girosi. A theory of networks for approximation and learning. Raport techniczny A. I. Memo 1140, MIT, Massachusetts, 1989.

[209] T. Poggio, F. Girosi. Network for approximation and learning. *Proceedings of the IEEE*, 78:1481–1497, 1990.

[210] H. Poulard. Barycentric correction procedure: A fast method of learning threshold units. *Proceedings of the World Congress on Neural Networks*, wolumen I, s. 710–713, Washington D.C., 1995.

[211] M. J. D. Powell. Radial basis functions for multivariable interpolation: A review. J. C. Mason, M. G. Cox, redaktorzy, *Algorithms for Approximation of Functions and Data*, s. 143–167, Oxford, 1987. Oxford University Press.

[212] S. Qian, Y. C. Lee, R. D. Jones, C. W. Barnes, K. Lee. Function approximation with an orthogonal basis net. *Proceedings of the IJCNN*, wolumen 3, s. 605–619, 1990.

[213] J. R. Quinlan. Programs for machine learning. San Mateo, CA: Morgan Kaufmann, 1993.

[214] J. R. Quinlan. Bagging, boosting, and C4.5. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, s. 725—-730, 1996.

[215] S. Ridella, S. Rovetta, R. Zunino. Circular backpropagation networks for classification. *IEEE Transaction on Neural Networks*, 8(1):84–97, 1997.

[216] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, 1996.

[217] D. B. Rubin. Multiple imputation after 18+ years. *Journal of the American Statistical Association*, 91:473—-489, 1996.

[218] D. Rutkowska. *Inteligentne systemy obliczeniowe. Algorytmy i sieci neuronowe w systemach rozmytych*. Akademicka Oficyna Wydawnicza, Warszawa, 1997.

[219] D. Rutkowska, M. Piliński, L. Rutkowski. *Sieci neuronowe, algorytmy genetyczne i systemy rozmyte*. Państwowe Wydawnictwa Naukowe, Warszawa, 1997.

[220] L. Rutkowski. *Filtry adaptacyjne i adaptacyjne przetwarzanie sygnałów*. Wydawnictwa Naukowo–Techniczne, Warszawa, 1994.

[221] L. Rutkowski, redaktor. *Sieci neuronowe i neurokomputery*. Wydawnictwo Politechniki Częstochowskiej, Częstochowa, 1996.

[222] A. Saranli, B. Baykal. Complexity reduction in radial basis function (RBF) networks by using radial B-spline functions. *Neurocomputing*, 18(1-3):183–194, 1998.

[223] C. Saunders, M. O. Stitson, J. Weston, L. Bottou, B. Schölkopf, A. J. Smola. Support vector machine reference manual. Raport techniczny CSD-TR-98-03, Department of Computer Science, Royal Holloway, University of London, Egham, UK, 1998.

[224] R. E. Schapire, Y. Freund, Peter Bartlett, Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[225] W. Schiffman, M. Joost, R. Werner. Comparison of optimized backpropagation algorithms. *Proceedings of ESANN'93*, s. 97–104, Brussels, 1993.

[226] B. Schölkopf, P. L. Bartlett, A. Smola, R. Williamson. Shrinking the tube: a new support vector regression algorithm. M. S. Kearns, S. A. Solla, D. A. Cohn, redaktorzy, *Advances in Neural Information Processing Systems*, wolumen 11, s. 330–336, Cambridge, MA, 1999. MIT Press.

[227] B. Schölkopf, C. Burges, A. Smola. *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[228] B. Schölkopf, A. J. Smola, R. C. Williamson, P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207—-1245, 2000.

[229] Bernhard Schölkopf, Alex Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[230] N. N. Schraudolph. A fast, compact approximation of the exponential function. Raport techniczny, IDSIA.

[231] A. K. Seewald, J. Fürnkranz. Evaluation of grading classifiers. *Advances in Intelligent Data Analysis: Proceedings of the Fourth International Symposium (IDA-01)*, Berlin, 2001. Springer-Verlag.

[232] N. Shang, L. Breiman. Distribution based trees are more accurate. *Proceedings of ICONIP'96*, 1996.

[233] S. Singhal, L. Wu. Training feed-forward networks with the extended kalman filter. *IEEE International Conference on Acoustic, Speech and Signal Processing*, s. 1187–1190, Glasgow, UK, 1989.

[234] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler nad R. S. Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. *Proceedings of the Symposium on Computer Applications and Medical Care*, s. 261–265. IEEE Computer Society Press, 1988.

[235] D. F. Specht. Probabilistic neural networks. *Neural Networks*, 3:109–118, 1990.

[236] B. Šter, A. Dobnikar. Neural networks in medical diagnosis: Comparison with other methods. A. B. Bulsari et al., redaktor, *Proceedings of the International Conference EANN '96*, s. 427–430, 1996.

[237] M. Sugeno, G. T. Kang. Structure identification of fuzzy model. *Fuzzy Sets and Systems*, 28:13–33, 1988.

[238] J. A. Sutherland. Holographic model of memory, learning and expression. *International Journal of Neural Systems*, 1(1):256–267, 1990.

[239] R. Tadeusiewicz. *Sieci neuronowe*. Akademicka Oficyna Wydawnicza, Warszawa, 1993.

[240] R. Tadeusiewicz. *Elementarne wprowadzenie od sieci neuronowych z przykładowymi programami*. Akademicka Oficyna Wydawnicza, Warszawa, 1998.

[241] R. Tadeusiewicz, A. Izworski, J. Majewski. *Biometria*. Wydawnictwa AGH, Kraków, 1993.

[242] Ryszard Tadeusiewicz. *W stronę uśmiechniętych maszyn*. Wydawnictwa ALFA, Warszawa, 1989.

[243] K. M. Ting, I. H. Witten. Stacked generalization; when does it work? *Proceedings of the 15th International Join Conference on Artificial Intelligence*, 1997.

[244] K. M. Ting, I. H. Witten. Stacking bagged and dagged models. *Proceedings of the 14th International Conference on Machine Learning*, 1997.

[245] V. Tresp. Committee machines. Yu Hen Hu, Jenq-Neng Hwang, redaktorzy, *Handbook of Neural Network Signal Procesing*. CRC Press, 2001.

[246] J. Żurada, M. Barski, W. Jędruch. *Sztuczne sieci neuronowe*. Państwowe Wydawnictwa Naukowe, 1996.

[247] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, Hingham, MA, 1997.

[248] R. J. Vanderbei. Loqo user's manual — version 3.10. Raport techniczny SOR-97-08, Princeton University, Statistics and Operations Research, 1997. Code available at http://www.princeton.edu/~rvdb/.

[249] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[250] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, NY, 1998.

[251] V. Vapnik. The support vector method of function estimation. C. M. Bishop, redaktor, *Neural Networks and Machine Learning*, s. 239–268. Springer-Verlag, 1998.

[252] A. S. Weigend, D. E. Rumelhart, B. A. Huberman. Back–propagation, weight elimination and time series prediction. *Proceedings of the 1990 Connectionist Models Summer School*, s. 65–80, Los Altos/Palo Alto/San Francisco, 1990. Morgan Kaufmann.

[253] A. S. Weigend, D. E. Rumelhart, B. A. Huberman. Generalization by weight elimination with application to forecasting. R. P. Lipmann, J. E. Moody, D. S. Touretzky, redaktorzy, *Advances in Neural Information Processing Systems 3*, s. 875–882, San Mateo, CA, 1991. Morgan Kaufmann.

[254] A. S. Weigned, H. Zimmermann, Ralph Neuneier. Clearing. P. Refenes, Y. Abu-Mostafa, J. Moody, A. Weigend, redaktorzy, *Proceedings of NNCM, Neural Networks in Financial Engineering*, Singapore, 1995. World Scientific.

[255] S.M. Weiss, I. Kapouleas. An empirical comparison of pattern recognition, neural nets and machine learning classification methods. J.W. Shavlik, T.G. Dietterich, redaktorzy, *Readings in Machine Learning*. Morgan Kauffman, 1990.

[256] D. Wettschereck, T. Dietterich. Improving the performance of radial basis function networks by learning center locations. J. E. Moody, S. J. Hanson, R. P. Lipmann, redaktorzy, *Advances in Neural Information Processing Systems*, wolumen 4, s. 1133–1140, San Mateo, CA, 1992. Morgan Kaufman.

[257] D. R. Wilson. *Advances in Instance-Based Learning Algorithms*. Praca doktorska, Department of Computer Science Brigham Young University, 1997.

[258] D. R. Wilson, T. R. Martinez. Heterogeneous radial basis function networks. *Proceedings of the International Conference on Neural Networks*, wolumen 2, s. 1263–1276, 1996.

[259] D. R. Wilson, T. R. Martinez. Instance-based learning with genetically derived attribute weights. *International Conference on Artificial Intelligence, Expert Systems and Neural Networks*, s. 11–14, 1996.

[260] D. R. Wilson, T. R. Martinez. Value difference metrics for continuously valued attributes. *Proceedings of the International Conference on Artificial Intelligence, Expert Systems and Neural Networks*, s. 11–14, 1996.

[261] D. R. Wilson, T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[262] Z. Świątnicki, R. Wantoch-Rekowski. *Sieci neuronowe w zastosowaniach wojskowych*. Dom wydawniczy Bellona, 2000.

[263] William H. Wolberg, O.L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences*, wolumen 87, s. 9193–9196, USA, 1990.

[264] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[265] C. T. Wolverton, T. J. Wagner. Recursive estimates of probability densities. *IEEE Transactions on SSC*, 5(3), 1969.

[266] T. Yamakawa, M. Shimono, T. Miki. Design criteria for robust associative memory employing non-equilibrium network. *Proceedings of the 4th International Conference on Soft Computing (IIZUKA'96)*, s. 688–691, Iizuka, 1996.

[267] H. F. Yanai, S. Amari. A theory of neural net with non-monotone neurons. *Proceedings of IEEE International Conference on Neural Networks*, s. 1385–1390, 1993.

[268] H. F. Yanai, S. Amari. Auto-associative memory with two-stage dynamics of non-monotonic neurons. *IEEE Transaction on Neural Networks*, 1998. (submitted).

[269] H. F. Yanai, Y. Sawada. Associative memory network composed of neurons with histeretic property. *Neural Networks*, 3:223–228, 1990.

[270] H. F. Yanai, Y. Sawada. Integrator neurons for analogue neural networks. *IEEE Transaction on Neural Networks*, 37:854–856, 1990.

[271] S. Yoshizawa, M. Morita, S. Amari. Capacity of associative memory using a nonmonotonic neuron model. *Neural Networks*, 6:167–176, 1993.

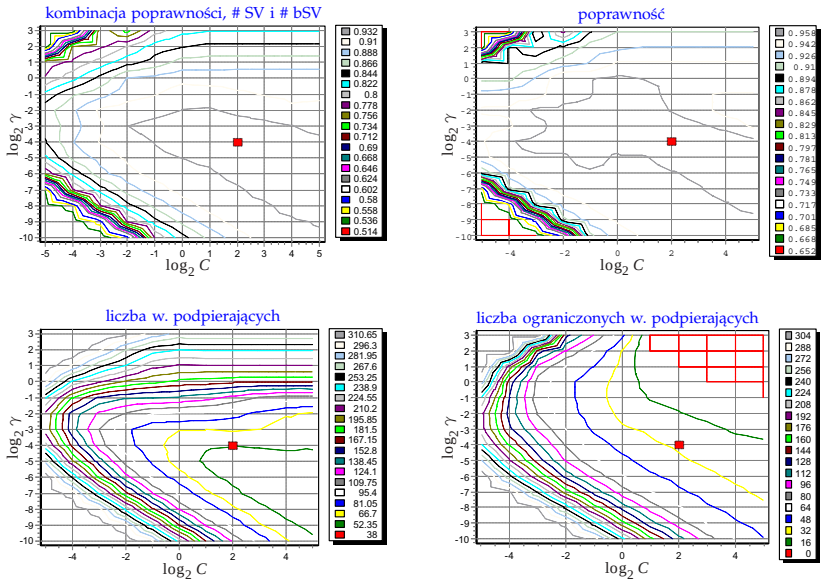
[272] Frederick Zarndt. A comprehensive case study: An examination of machine learning and connectionist algorithms. Praca magisterska, Department of Computer Science Brigham Young University, 1995.

[273] Bernard Zenko, Ljupco Todorovski, Saso Dzeroski. A comparison of stacking with meta decision trees to bagging, boosting, and stacking with other methods. *ICDM*, s. 669–670, 2001.

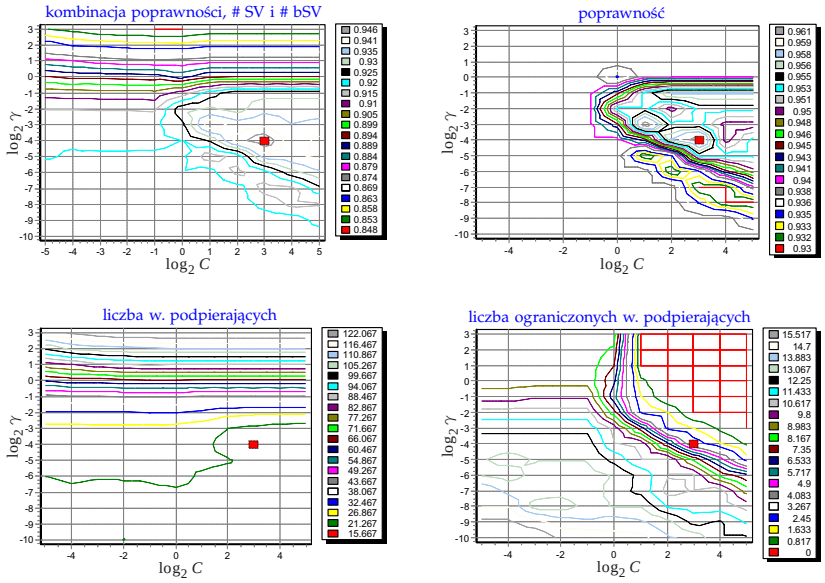
Skorowidz

Dodatek

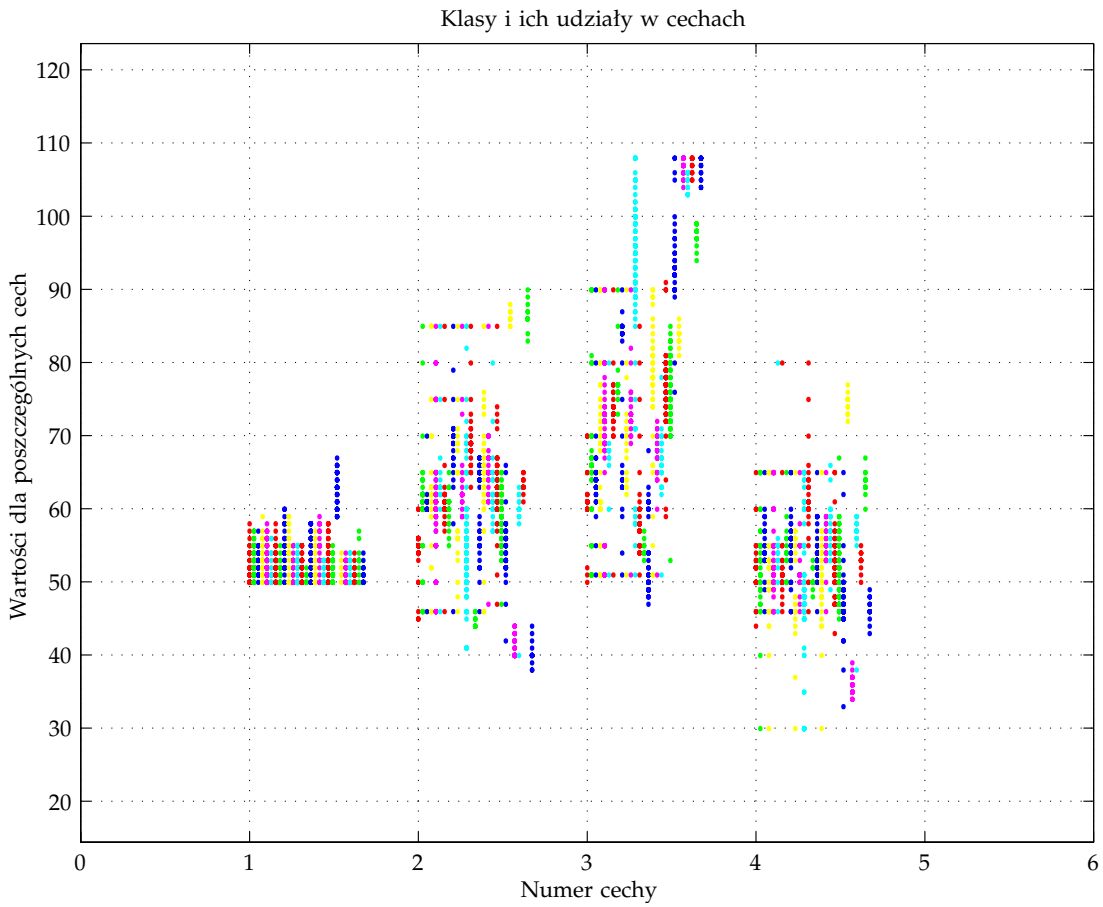
Ilustracje kolorowe



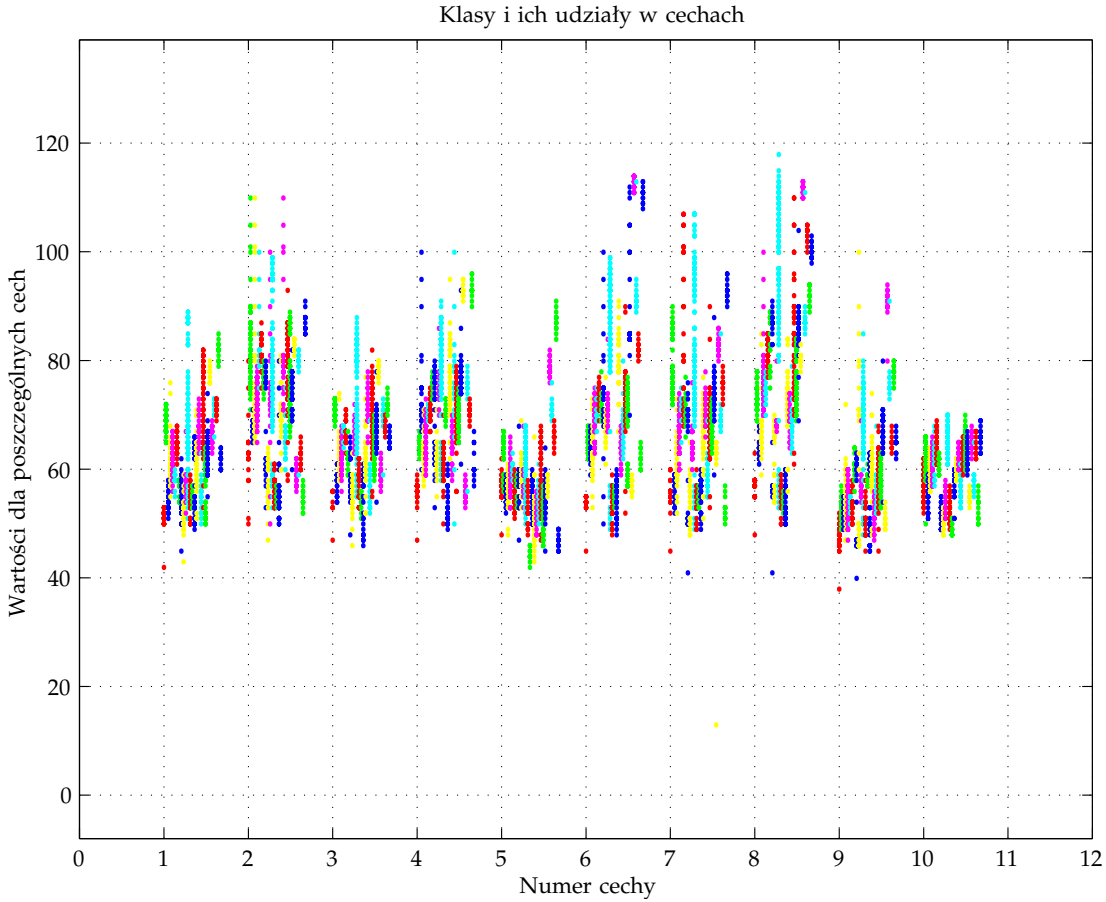
Rysunek .1: Gęstości: kryterium Meta-SVM, poprawności, liczby wektorów podpierających (SV) i liczby ograniczonych wektorów podpierających dla testu *wisconsin breast cancer*.



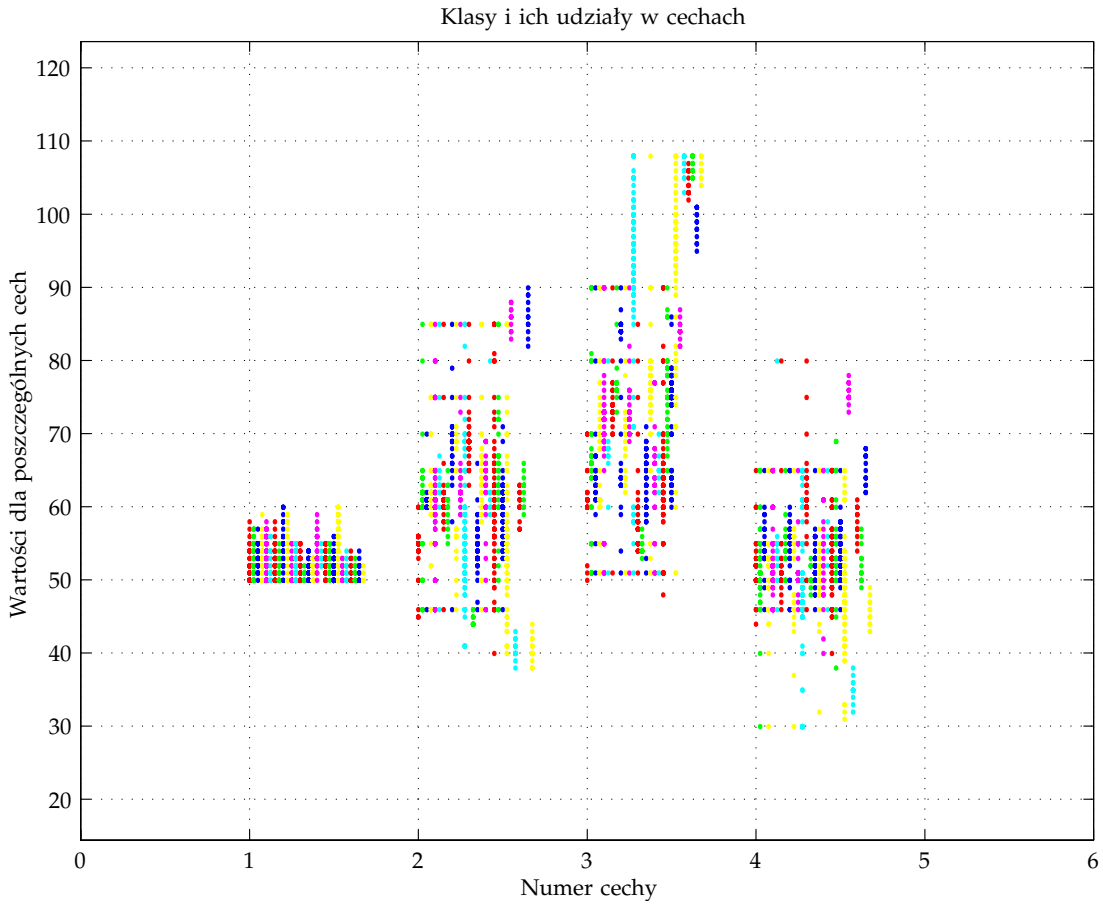
Rysunek .2: Gęstości: kryterium Meta-SVM, poprawności, liczby wektorów podpierających (SV) i liczby ograniczonych wektorów podpierających dla testu *glass*.



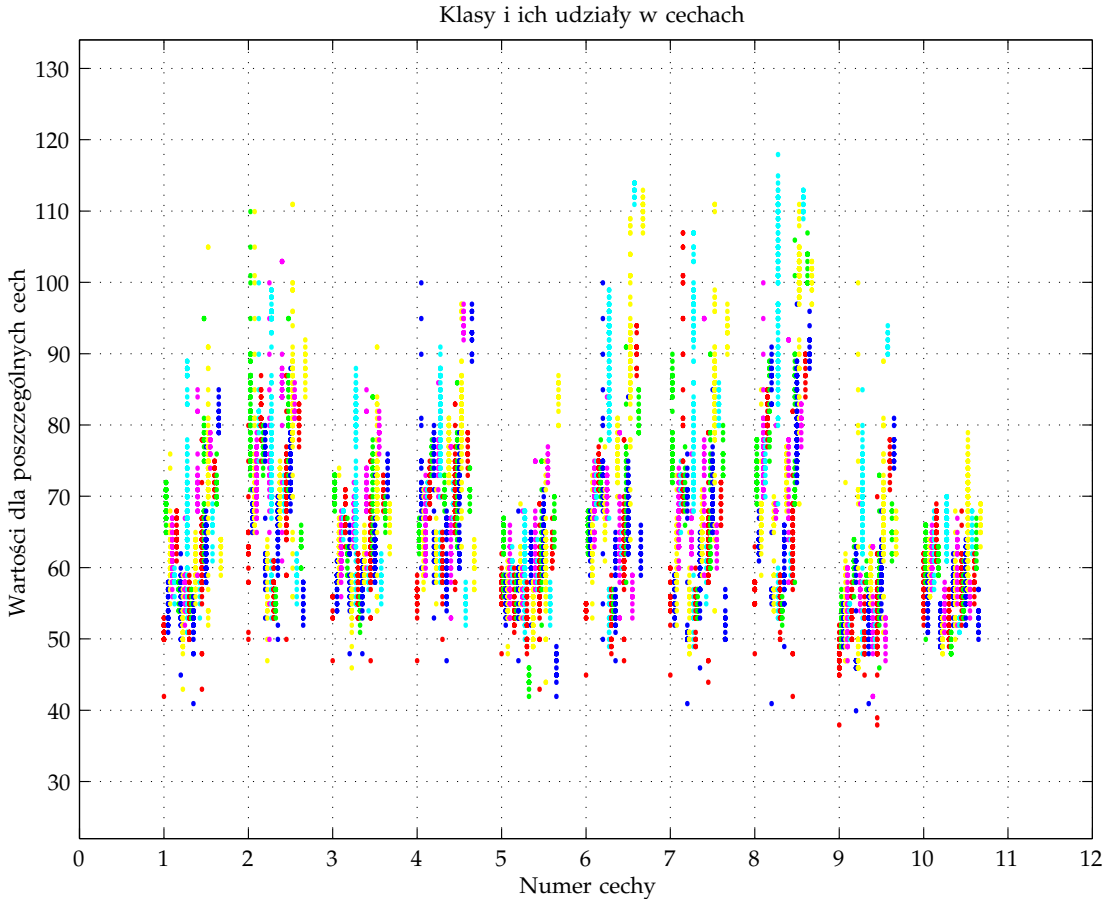
Rysunek .3: Pierwsza baza danych psychometrycznych (1027 wektorów, 27 klas, 14 wymiarowe wejście). Rysunek ukazuje pierwsze cztery cechy — skale kontrolne.



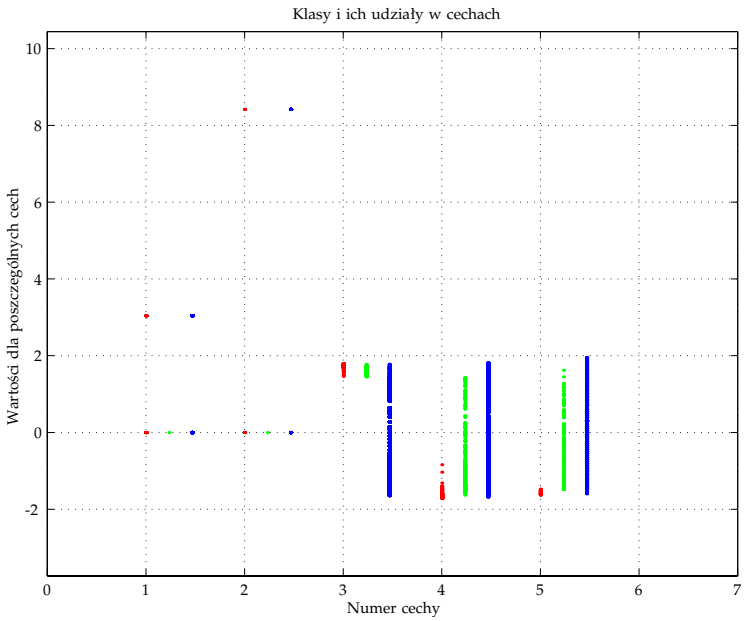
Rysunek .4: Pierwsza baza danych psychometrycznych (1027 wektorów, 27 klas, 14 wymiarowe wejście). Rysunek ukazuje 10 kolejnych cech — skale kliniczne.



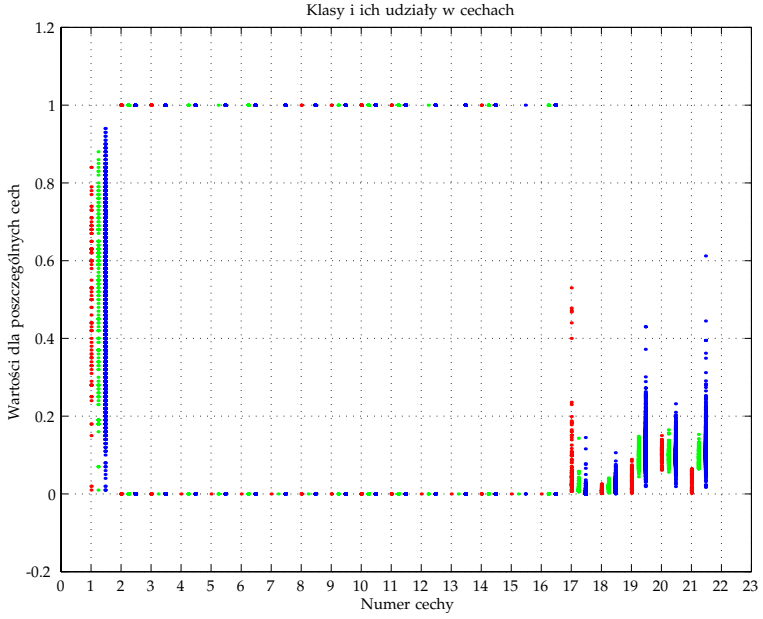
Rysunek .5: Druga baza danych psychometrycznych (1167 wektorów, 28 klas, 14 wymiarowe wejście). Rysunek ukazuje pierwsze cztery cechy — skale kontrolne.



Rysunek .6: Druga baza danych psychometrycznych (1167 wektorów, 28 klas, 14 wymiarowe wejście). Rysunek ukazuje 10 kolejnych cech — skale kliniczne.



Rysunek .7: Baza danych nadczynności i niedoczynności tarczycy po selekcji istotnych cech i transformacji.



Rysunek .8: Baza danych nadczynności i niedoczynności tarczycy.