Uniwersytet Mikołaja Kopernika
Wydział Matematyki i Informatyki

Gracjan Wilczewski
nr albumu: 158162

Praca magisterska
na kierunku informatyka

InTrees: Modularne podejście do Drzew Decyzyjnych

Opiekun pracy dyplomowej:
prof. dr hab. Włodzisław Duch
Wydział Fizyki, Astronomii i
Informatyki Stosowanej

Toruń 2008

Pracę przyjmuję i akceptuję

.....

data i podpis opiekuna pracy

Potwierdzam złożenie pracy dyplomowej

.....

data i podpis pracownika dziekanatu

Serdeczne podziękowania dla promotora prof. dra hab. Włodzisława Ducha,
dra Krzysztofa Grąbczewskiego, dra Norberta Jankowskiego
za poświęcony czas i cenne uwagi podczas pisania pracy magisterskiej
oraz dla moich najdroższych rodziców za cierpliwość i wyrozumiałość.

Spis treści

Wstęp

Głównym celem tej pracy było stworzenie modułu InTrees współpracującego z powstającą na Uniwersytecie Mikołaja Kopernika w Toruniu platformą Intemi, realizującą zadania z zakresu inteligencji obliczeniowej. Zgodnie z przyjętymi założeniami i tendencjami obowiązującymi w nowoczesnej informatyce, aplikacja została zaprojektowana i zrealizowana w całości w postaci obiektowej, jest skalowalna i posiada budowę modularną, co daje szerokie możliwości jej przyszłej rozbudowy. Użytkownik znajdzie w niej szczegółowe implementacje najpopularniejszych i najskuteczniejszych algorytmów budowy drzew decyzji. Dzięki unikalnemu podejściu do tematu, korzystający z łatwością może łączyć poszczególne elementy znanych metod z własnymi implementacjami. Całość zintegrowana jest z platformą Intemi, dzięki czemu uzyskujemy dostęp do szeregu zasobów, takich jak walidacja krzyżowa, metody wizualizacji, repozytorium wyników oraz wiele innych.

Praca składa się z czterech głównych części. Pierwsza zawiera wprowadzenie do metod drzew decyzyjnych, wraz z omówieniem głównych elementów typowych algorytmów konstrukcji drzew klasyfikacji. Moim celem nie było wyczerpujące przedstawienie tematu, a jedynie omówienie jego najważniejszych aspektów oraz zarys podstawowych idei. Całość stworzona została z myślą o przyszłych użytkownikach modułu Intemi, ze szczególnym uwzględnieniem tych, którzy dopiero rozpoczynają swoją przygodę z inteligencją obliczeniową.

Kolejne rozdziały to przegląd najbardziej znanych i osiągających najlepsze wyniki algorytmów generujących drzewa klasyfikacji. Można traktować go także, jako uściślenie koncepcji nakreślonych w pierwszej części pracy, poprzez nadanie im konkretnej postaci i wskazanie zastosowanych w nich metod i rozwiązań. Większość przedstawionych algorytmów znalazło się w module InTrees, a ich implementacje zostały omówione w dalszej

części pracy. Jednym z głównych tematów jest zagadnienie reguł logicznych, a w szczególności proces ich generowania na podstawie gotowych drzew klasyfikacji.

Część poświęcona opisowi implementacji modułu InTrees przeznaczona jest dla programistów i zaawansowanych użytkowników aplikacji pragnących rozwijać zaprezentowane rozwiązanie. Może ona także posłużyć jako pomoc przy tworzeniu kolejnych modułów platformy Intemi.

Praca zakończona jest eksperymentami, które w ciekawy sposób porównują poszczególne elementy algorytmów konstruujących drzewa klasyfikacji.

Rozdział 1

Wprowadzenie do drzew decyzyjnych

Sposób reprezentowania wiedzy służącej do podejmowania decyzji przy pomocy drzew jest bardzo stary i nie wywodzi się ani z systemów ekspertowych ani z inteligencji obliczeniowej. Obecnie stał się on jednak podstawową metodą indukcyjnego uczenia się maszyn. Wpłynęła na to duża efektywność, możliwość prostej implementacji, a także duża intuicyjność i łatwość w rozumieniu drzew klasyfikacji przez człowieka.

1.1 Teoria drzew decyzyjnych

1.1.1 Struktura drzewa, podstawowe pojęcia i definicje

Rozważając pojęcie drzewa decyzyjnego mamy na myśli strukturę złożoną z węzłów, z których wychodzą gałęzie prowadzące do innych węzłów lub liści. Drzewo decyzyjne ma typowe właściwości drzewa, w znaczeniu, jakie temu pojęciu nadaje się w informatyce. Formalnie mianem drzewa określamy dowolny spójny graf acykliczny. Krawędzie grafu nazywane są gałęziami, wierzchołki z których wychodzi co najmniej jedna krawędź węzłami, a pozostałe wierzchołki liśćmi. [2]

Definicja 1.1 ***(Przestrzeń klasyfikacji)***
Przestrzeń klasyfikacji określa się jako zbiór własności q, które charakteryzują elementy

zbioru W.

Definicja 1.2 ***(Drzewo klasyfikacji)***

Drzewem klasyfikacji dla przestrzeni klasyfikacji *X* i zbioru klas C

nazywamy parę *D* spełniającą jeden z poniższych warunków:

1. $D = (W, ())$, gdzie *W* jest węzłem a () jest pustą listą drzew.

2. $D = (W, (D_1, \ldots, D_n))$, gdzie $W = (f, c)$ jest węzłem, n ∈ N, dla każdego i ∈ {1, . . .n}, $D_i = ((f_i, c_i), L_i)$ jest drzewem (które nazywamy *poddrzewem bezpośrednim* drzewa *D*) oraz

dla każdego $x \in X$, $\sum_{i=1}^{n} f_i(x) \leq 1$ (tzn., że każdy wektor wpada co najwyżej do jednego podwęzła).

Węzeł *W* nazywamy wówczas *węzłem głównym* lub *korzeniem* drzewa *D*. Węzły główne drzew $D_1, \ldots, D_n$ nazywamy *podwęzłami* albo *dziećmi* węzła *W*, a węzeł *W* ich *nadwęzłem* lub *rodzicem.*

Definicja 1.3 ***(Węzeł drzewa klasyfikacji)***

Węzłem *drzewa klasyfikacji dla przestrzeni klasyfikacji X i zbioru klas C nazywamy dowolną parę* $W = (f, c)$ *, gdzie* $f : X \to \{0,1\}$ *oraz* $c \in C$*. Funkcję f nazywamy wówczas funkcją przynależności do węzła W, a klasę c etykietą węzła W.*

Definicja 1.4 ***(Klasyfikator)***

Klasyfikatorem opartym na drzewie *D* nazywamy funkcję klasyfikującą, która każdemu wektorowi $x \in X$ przyporządkowuje klasę, która jest etykietą liścia zawierającego *x*.

Warto także uściślić kilka terminów używanych w ogólnej teorii drzew.

Definicja 1.5 ***(Rodzeństwo)***

Węzły mające wspólnego rodzica nazywamy rodzeństwem (każdy węzeł jest dla pozostałych *bratem*).

Definicja 1.6 ***(Poddrzewo)***

Poddrzewem drzewa *W* jest jego każde poddrzewo bezpośrednie, a także każde

poddrzewo dowolnego poddrzewa bezpośredniego.

Definicja 1.7 ***(Liść drzewa)***

Liściem drzewa *D* nazywamy każdy węzeł drzewa *W*, który jest węzłem głównym poddrzewa z pustą listą poddrzew właściwych.

Definicja 1.8 ***(Gałąź drzewa)***

Gałęzią drzewa *D* nazywamy dowolny ciąg węzłów ($W_1, \ldots W_n$) taki, że n ∈ N, W_1 jest węzłem głównym drzewa *D*, W_n jego liściem oraz dla każdego i ∈ {2, . . .,n} W_i jest podwęzłem węzła W_{i-1}.

Definicja 1.9 ***(Długość gałęzi drzewa)***

Długością gałęzi drzewa nazywamy liczbę węzłów ją stanowiących.

Definicja 1.10 ***(Głębokość drzewa)***

Głębokością drzewa nazywamy maksymalną długość gałęzi tego drzewa.

Definicja 1.11 ***(Drzewo binarne)***

Drzewem binarnym nazywamy drzewo, w którym każdy węzeł nie będący liściem posiada dokładnie dwa podwęzły.

Definicja 1.12 ***(Przynależność do węzła drzewa)***

Mówimy, że $x \in X$ należy (bądź *wpada*) do węzła $W=(f,c)$ drzewa *D* o ile $f(x)=1$ oraz, albo *W* jest węzłem głównym *D*, albo *x* należy do nadwęzła węzła *W* w drzewie *D*.

1.1.2 Przykładowe drzewo decyzyjne

Przykład 1.1

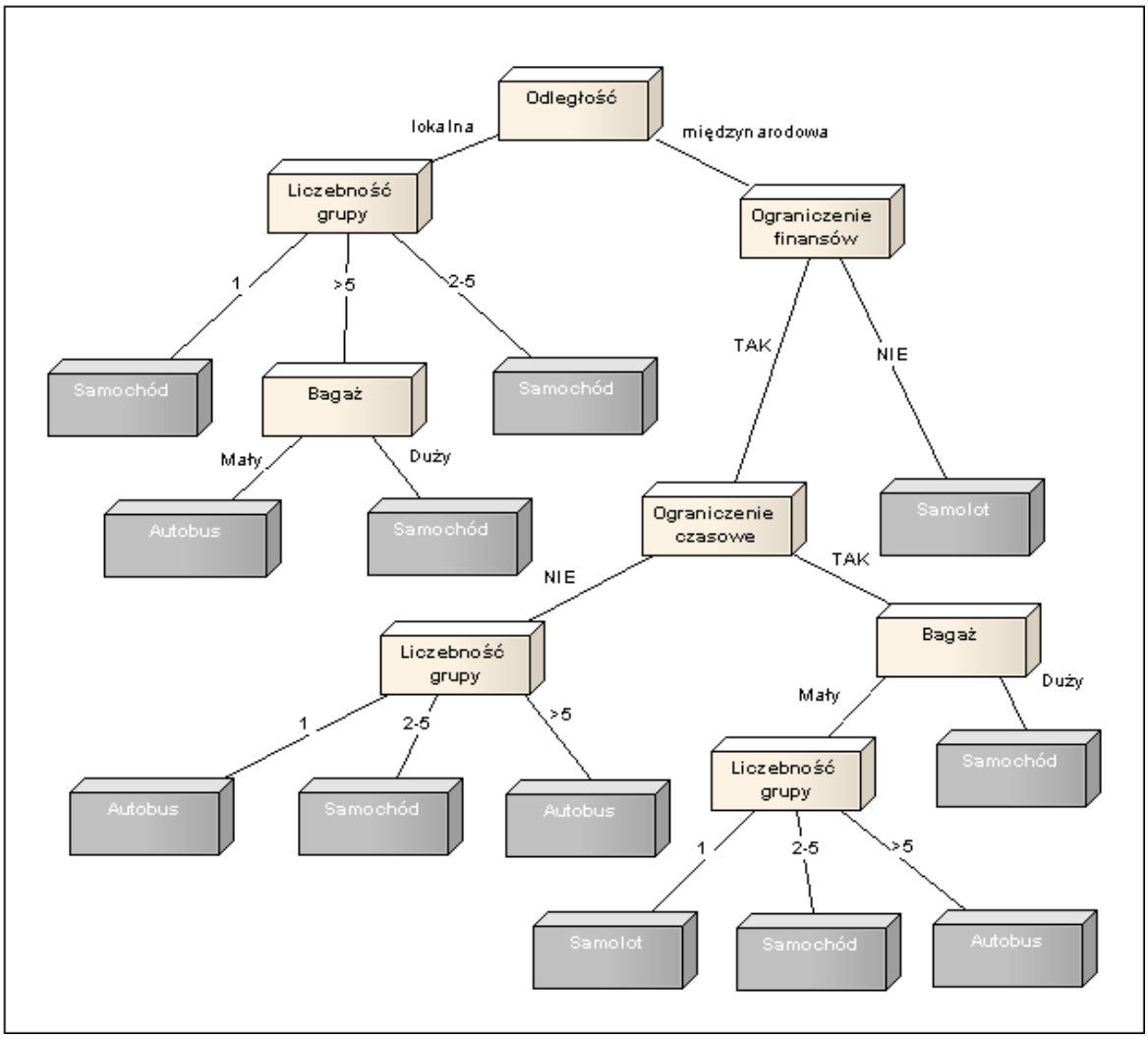
Załóżmy, że chcemy zautomatyzować proces podejmowania decyzji o wyborze środka lokomocji podczas podróży służbowej naszych pracowników.

Mamy do dyspozycji następujące parametry (cechy) opisujące każdą delegację:

- odległość: (lokalna, międzynarodowa)
- liczebność grupy (1, 2-5, >5)
- ograniczenie finansów (tak, nie) rozumiane jako: “Czy koszt odgrywa znaczącą rolę?”
- ograniczenie czasowe (tak, nie) rozumiane jako: “Czy czas odgrywa znaczącą rolę?”
- bagaż (mały, duży)

Przyjmijmy, że jeżeli liczebność grupy przekracza 5 osób, to podróż samochodem nadal jest możliwa. Zostanie użyty po prostu więcej niż jeden samochód.

Przykładowe drzewo decyzyjne wygenerowane dla tego problemu mogłoby wyglądać następująco (Rysunek 1.1).



Rysunek 1.1: Przykładowe drzewo decyzyjne

1.1.3 Zalety i ograniczenia drzew decyzyjnych

Drzewa decyzyjne mogą być budowane przy wykorzystaniu algorytmicznych technik "dziel i zwyciężaj". Metoda ta jest znana ze swej szybkości [25]. Dodatkowo reprezentacja wykorzystująca drzewa jest przy tym stosunkowo efektywna pamięciowo w porównaniu z innymi podejściami, chociaż obecnie nie jest to już tak istotne. Ważniejsze i godne podkreślenia jest to, że drzewa decyzyjne umożliwiają niezwykle efektywną implementację procesu klasyfikowania przykładów, czyli stosowania hipotezy uzyskanej w procesie uczenia się [1].

Drzewa mogą być używane zarówno do selekcji jak i ekstrakcji cech. Bardzo istotną zaletą drzew jest to, że ich modele są stosunkowo zrozumiałe dla człowieka, łatwe do wizualizacji i translacji na reguły logiczne. Dlatego też zalecane są w przypadkach, w których ważna jest nie tylko poprawność klasyfikacji, ale również możliwość logicznej oceny proponowanych przez system rozwiązań.

Drzewa klasyfikacji mogą reprezentować dowolne pojęcia [1], jeśli tylko ich definicje można wyrazić w zależności od atrybutów, należy jednak podkreślić fakt, że w przypadku niektórych złożonych hipotez odpowiednie drzewa mogą być równie złożone.[1] Znanymi przykładami nie posiadającymi naturalnej reprezentacji za pomocą drzew decyzji są sytuacje “większość jest za” oraz “prawdziwość n czynników z m możliwych” [4]. Ograniczenia drzew wynikają głównie z faktu, że stosowane testy sprawdzają wartości pojedynczych atrybutów, tracąc w ten sposób szansę na wykorzystanie ewentualnych zależności pomiędzy poszczególnymi wartościami atrybutu, co mogłoby zaowocować sformułowaniem prostszej hipotezy. Znamienny jest również fakt, że o ile koniunkcje atrybutów reprezentuje się w bardzo naturalny sposób poprzez sekwencję atrybutów wzdłuż pewnej ścieżki w drzewie, to próba reprezentacji ich alternatywy często prowadzi do znacznej komplikacji budowy drzewa.

Uciążliwy może być również brak stabilności drzew [3]. W skrajnych przypadkach zmiana pojedynczej wartości atrybutu może prowadzić do całkowitej przebudowy struktury drzewa. Wreszcie drzewa decyzyjne nie dają możliwości łatwej aktualizacji, czyli doskonalenia drzewa po dołączeniu do zbioru treningowego nowych przypadków, a nieliczne algorytmy oferujące takie możliwości, osiągają zwykle rezultaty gorsze niż w przypadku budowania drzewa od podstaw.

1.2 Konstruowanie drzewa metodą zstępującą

Istnieje wiele algorytmów uczenia się pojęć z wykorzystaniem drzew decyzyjnych do reprezentacji hipotez. Zgodnie z ogólnym celem uczenia się indukcyjnego, dążą one do uzyskania możliwie prostego drzewa decyzyjnego klasyfikującego przykłady ze zbioru treningowego z jak najmniejszym błędem oczekując, że takie drzewo będzie miało równie niewielki błąd podczas klasyfikacji dokonywanej na zbiorze testowym [1].

W większości przypadków algorytmy konstrukcji drzew decyzyjnych można sprowadzić do wspólnego schematu budowy drzewa metodą zstępującą. W schemacie takim wyróżnić można cztery podstawowe elementy: kryterium rozbudowy, kryterium stopu, test atrybutu, metoda przycinania drzewa (ang. pruning). Szczególne znaczenie dla kształtu drzewa powstałego w wyniku działania algorytmu mają dwa pierwsze elementy - kryterium rozbudowy oraz kryterium stopu. Metoda przycinania drzewa nie jest już tak ściśle związana z danym algorytmem i definiuje sposób radzenia sobie z nadmiernym dopasowaniem się modelu do danych treningowych (ang. overfitting)[10]. Rodzaje testów atrybutu określają natomiast typy danych, do klasyfikacji których drzewo może zostać wykorzystane. Część algorytmów ograniczonych jest wyłącznie do atrybutów nominalnych, inne radzą sobie świetnie także z atrybutami porządkowymi i ciągłymi.

1.2.1 Zarys algorytmu

Schemat konstrukcji drzewa decyzyjnego może zostać przedstawiony w postaci procedury rekurencyjnej *budujDrzewo*. Jako argumentów podawanych przy jej wywołaniu używamy trzech zmiennych:

P - zbiór etykietowanych przykładów pojęcia docelowego *c,*

d - domyślna etykieta kategorii przyporządkowywana do liścia, gdy nie możliwe jest określenie jego właściwej etykiety,

S - zbiór wszystkich możliwych testów,

Przykładowe wywołanie procedury może mieć następującą postać:

$$budujDrzewo(T, \arg\max_{d} |T^d|, S)$$

Przez T^d oznaczamy podzbiór zbioru treningowego T, którego przykłady etykietowane są kategorią *d*. Za domyślną etykietę kategorii, przyjęliśmy więc tutaj najliczniej reprezentowaną

kategorię w zbiorze treningowym.

W podanym algorytmie (Algorytm 1.1) o utworzeniu liścia decyduje kryterium stopu. Spełnienie kryterium stopu powoduje utworzenie liścia. Jeżeli kryterium stopu nie jest spełnione tworzony jest węzeł n - następuje wybór testu t_n ze zbioru *S,* a następnie dla każdego z jego możliwych wyników tworzona jest gałąź prowadząca do poddrzewa, które budowane jest poprzez rekurencyjne wywołanie tej samej procedury. Etykieta klasy (zwykle klasy większościowej) przypisywana jest na każdym poziomie budowy drzewa, zarówno do węzłów jak i liści. Ułatwia to późniejszy proces przycinania drzewa. W wywołaniu rekurencyjnym tworzącym poddrzewo, do którego prowadzi gałąź odpowiadająca wynikowi *r* testu t_n, przekazujemy zbiór tylko tych przykładów, dla których test t_n daje wynik *r* $(P_{t_n} r)$. Zbiór pozostających do wykorzystania testów *S* jest pomniejszany o t_n, gdyż jego ponowne wykorzystanie na niższym poziomie drzewa nie może przynieść żadnych korzyści [1].

funkcja *budujDrzewo(P,d,S)*
argumenty wejściowe: *P, d, S* - zgodnie z opisem powyżej
wynik: korzeń drzewa decyzyjnego reprezentującego hipotezę przybliżającą *c* na zbiorze *P*

```
budujDrzewo(P,d,S)
    jeśli kryteriumStopu(P,S) to
        utwórz liść k;
        d_k := kategoria(P,d);
        zwróć k;
    koniec jeśli
    utwórz węzeł n;
    t_n := wybierzTest(P,S);
    d_n := kategoria(P,d);
    dla wszystkich r ∈ R_{t_n} wykonaj
        n[r] := budujDrzewo (P_{t_n r}, d_n, S − {t_n});
    koniec dla
    zwróć n;
```

Algorytm 1.1: Schemat zstępującego konstruowania drzewa decyzyjnego

1.2.2 Test atrybutu. Rodzaje testów.

Operacja wyboru testu, nazywana przez nas kryterium rozbudowy drzewa, jest rdzeniem wszystkich algorytmów indukcji drzew decyzyjnych opartych na schemacie zstępującej budowy drzewa i to ona przede wszystkim decyduje o ich własnościach [1]. Istotne jest jednak nie tylko to jak wybieramy, ale także co tak naprawdę mamy do wyboru.

Rodzaje testów dla pojedynczych atrybutów zależą od typów tych atrybutów, spośród których wyróżniamy trzy najbardziej podstawowe: nominalne, porządkowe i ciągłe.
Dla atrybutów nominalnych o skończonej liczbie wartości stosuje się zwykle test polegający na sprawdzaniu wartości atrybutu. Wynikiem takiego testu jest po prostu wartość danego atrybutu. Taki test nazywamy *testem tożsamościowym.*

Definicja 1.13 ***(Test tożsamościowy)***

Test $t: X \to R_t$ jest testem tożsamościowym dla atrybutu $a: X \to A$, to $R_t = A$ i dla każdego przykładu $x \in X$ mamy $t(x) = t(a)$

Dla atrybutów nominalnych można stosować również *testy równościowe* oraz t*esty przynależnościowe* [1].

Definicja 1.14 ***(Test równościowy)***

Test $t: X \to R_t$ jest testem równościowym dla atrybutu $a: X \to A$, jeżeli jest zdefiniowany dla każdego przykładu $x \in X$ następująco:

$$t(x) = \begin{cases} 1 & jeśli\ a(x) = \upsilon, \\ 0 & jeśli\ a(x) \neq \upsilon, \end{cases}$$

przy czym $\upsilon \in A$ jest jedną z możliwych wartości atrybutu *a.* Wówczas oczywiście $R_t = \{0,1\}$

Definicja 1.15 ***(Test przynależnościowy)***

Test $t: X \to R_t$ jest testem przynależnościowym dla atrybutu $a: X \to A$, jeżeli jest zdefiniowany dla każdego przykładu $x \in X$ następująco:

$$t(x) = \begin{cases} 1 & jeśli\ a(x) \in V, \\ 0 & jeśli\ a(x) \notin V, \end{cases}$$

przy czym $V \subset A$ jest pewnym właściwym podzbiorem przeciwdziedziny atrybutu *a.*

Testy równościowe nie zwiększają jednak w sposób znaczący „siły wyrazu" drzew decyzyjnych, natomiast testy przynależnościowe pomimo, iż pozwalają czasami na reprezentację hipotezy w sposób znacząco prostszy, znacznie zwiększają obliczeniową złożoność problemu wyboru testu [1]. . Testów przynależnościowych może być bardzo dużo, a ich liczba wykładniczo zależy od rozmiaru przeciwdziedziny atrybutu, którego dotyczą.

Dla atrybutów porządkowych o skończonej liczbie wartości można stosować wszystkie testy, o których była mowa przy okazji atrybutów dyskretnych. Należy mieć jednak na uwadze, że rezygnacja z dodatkowej informacji jaką jest relacja porządku atrybutów, może pozbawić nas możliwości konstrukcji prostszych i bardziej dokładnych drzew [1].

W przypadku atrybutów ciągłych możliwe jest stosowanie testów przynależnościowych, w których podzbiorem przeciwdziedziny testowanego atrybutu jest pewien przedział. Uporządkowanie wartości ciągłych jest oczywiście uwzględniane i pozostaje jedynie wybór końców przedziałów. Najczęściej wykorzystuje się relację porządku, którą możemy oznaczać używając matematycznych symboli nierówności. Polega ona na porównaniu wartości atrybutu z wybraną wartością z jego przeciwdziedziny za pomocą tych nierówności. Taki test nazywamy *testem nierównościowym.*

Definicja 1.16 ***(Test nierównościowy)***

Test $t: X \to R_t$ jest testem przynależnościowym dla atrybutu $a: X \to A$, jeżeli jest zdefiniowany dla każdego przykładu $x \in X$ następująco:

$$t(x) = \begin{cases} 1 & jeśli\ a(x) \leqslant \theta, \\ 0 & jeśli\ a(x) > \theta, \end{cases}$$

przy czym $\theta \in A$ jest pewną wartością progową z przeciwdziedziny atrybutu.

W praktyce najczęściej stosowanymi testami są testy tożsamościowe i nierównościowe.

1.2.3 Kryterium rozbudowy

Centralnym punktem każdego algorytmu konstruującego drzewo metodą zstępującą jest kryterium wyboru testu (kryterium rozbudowy). Decyduje ono o złożoności drzewa, czyli po prostu o jego rozmiarze. Zadaniem dobrego kryterium rozbudowy jest wybór, spośród

testów kandydujących, testu generującego jak najkorzystniejszy podział próbek. Aby podjąć decyzję o wyborze testu potrzebna jest funkcja liczbowo oceniająca wartość testów. Wybór testu maksymalizuje wartość funkcji oceny lokalnie, czyli niezależnie dla każdego węzła, co niekoniecznie prowadzić musi do optymalnych drzew rozpatrywanych jako całość. Z drugiej strony konstrukcja drzew globalnie optymalnych byłaby nieporównywalnie bardziej kosztowna, dlatego też w praktyce stosuje się różne metody szukania heurystycznego. Poprawność klasyfikacji poszczególnych węzłów nie przekłada się wprost na poprawność klasyfikacji całych drzew, dlatego większość systemów budowania drzew klasyfikacji używa najczęściej kryteriów podziału (węzłów na podwęzły) wywodzących się z teorii informacji, uznając prostą jakość klasyfikacji jako kryterium niewystarczające [8]. Najczęściej stosowane kryteria zostaną przedstawione w dalszej części poświęconej przeglądowi popularnych algorytmów drzew decyzyjnych.

1.2.4 Kryterium stopu

Kryterium stopu procedury budowy drzewa odpowiada za jego dokładność. Podczas rekurencyjnego generowania drzewa klasyfikacji, zbiór przykładów przekazywanych do procedury budowy drzewa ulega zmniejszeniu. Pozostawiane są w nim jedynie przykłady, dla których test wybrany w poprzednio utworzonym węźle daje wynik odpowiadający wychodzącej z tego węzła gałęzi. Na coraz głębszych poziomach rekurencji stopniowo zmniejsza się liczba przykładów co pociąga za sobą zmniejszanie się liczby różnych kategorii. Automatycznie formułuje się więc naturalne kryterium stopu, które kończy konstrukcję, gdy zbiór przykładów jest pusty lub gdy wszystkie przykłady należą do tej samej kategorii.

$$P=\emptyset \vee |\{d' \in C | (\exists x \in P) c(x)=d'\}|=1,$$

Kiedy podjęta zostanie decyzja o zaprzestaniu podziałów i utworzeniu liścia, należy przydzielić do niego etykietę kategorii, wspólną dla wszystkich przykładów z tego zbioru. Na ogół stosuje się kategorię domyślną, wskazującą na najczęściej występującą klasę na bezpośrednio wcześniejszym poziomie rekursji, kiedy jeszcze zbiór przykładów nie był pusty. Jeżeli zakończenie procesu zachodzi na skutek tego, że wszystkie rozważane przykłady należą do tej samej klasy to oczywiście liść etykietujemy tą właśnie klasą.

Zwróćmy uwagę, że zbiór możliwych do wykorzystania testów także ulega zmniejszaniu w miarę postępowania procesu konstrukcji drzewa. Ponieważ jest to zbiór skończony, to na odpowiednio głębokim poziomie wywołania rekurencyjnego stanie się on

zbiorem pustym, chyba że wcześniej zadziała jedno z kryteriów opisanych powyżej. Tak sytuacja oznacza, że zestaw testów przekazanych algorytmowi jest niewystarczający do skonstruowania drzewa spójnego ze zbiorem trenującym. Obserwujemy to przede wszystkim wtedy, gdy zbiór trenujący nie jest poprawny i zawiera przekłamania lub gdy zestaw atrybutów nie opisuje przykładów w dostateczny sposób i w związku z tym przestrzeń hipotez jest zbyt uboga do reprezentowania pojęcia docelowego. W takim przypadku algorytm zgłasza brak możliwości konstrukcji drzewa klasyfikacji lub tworzony jest liść etykietowany klasą *większościową.*

Zdarzają się sytuacje, kiedy świadomie podejmujemy decyzję o zaprzestaniu dalszych podziałów pomimo, że węzeł, który przekształcamy w liść zawiera przykłady należące do różnych klas. Postępowanie takie ma nas uchronić od zbytniego dopasowywania się drzewa klasyfikacji do danych treningowych. Spośród najczęściej stosowanych w takiej sytuacji kryteriów stopu możemy wyróżnić kryteria uwzględniające:

- osiągnięcie zakładanej dokładności podziałów,
- osiągnięcie zakładanej ilości gałęzi,
- osiągnięcie ilości atrybutów mniejszej niż zakładany próg.

1.2.5 Przycinanie

W poprzednim podrozdziale omawiane były kryteria stopu, mające zapobiec zbytniemu dopasowaniu się modelu klasyfikacji do danych treningowych. Praktyka dowodzi jednak, że znacznie lepsze rezultaty, niż wczesne kończenie budowy drzewa, daje technika zwana przycinaniem (ang. pruning). Przycinanie to usuwanie nieistotnych z punktu widzenia działania klasyfikatora części w pełni rozrośniętego drzewa. Polega na zastąpieniu jego

f**unkcja** przytnijDrzewo(T,P)
argumenty wejściowe:
T - drzewo do przycięcia
P - zbiór przycinania
wynik: drzewo *T* po przycięciu
przytnijDrzewo(T,P)
dla wszystkich węzłów **n** drzewa *T* **wykonaj**
zastąp **n** liściem **k** z etykietą większościowej kategorii w zbiorze $P_{T,n}$ jeśli nie powiększy to szacowanego na podstawie *P* błędu rzeczywistego drzewa *T*;
koniec dla
zwróć *T*.

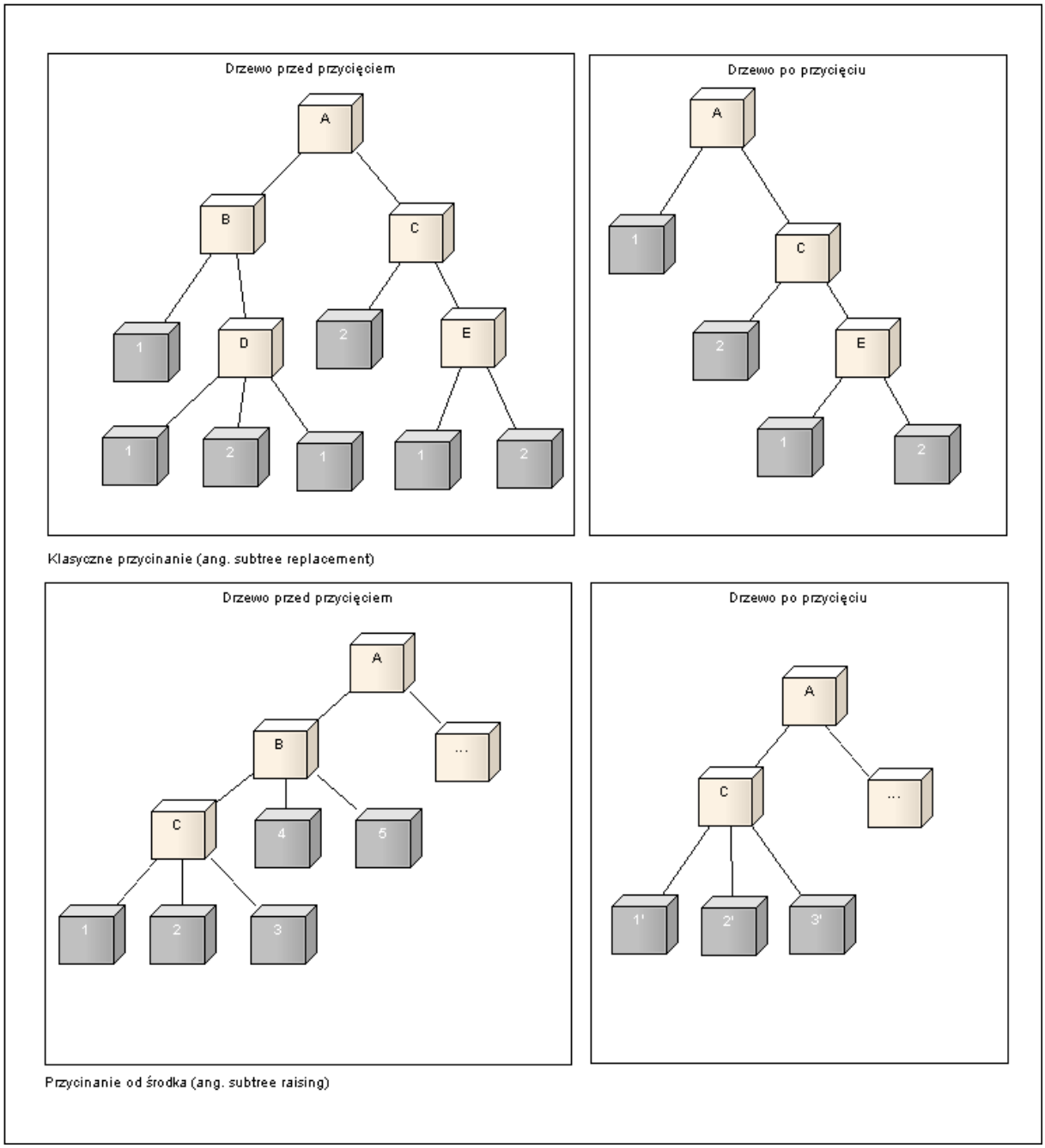
Algorytm 1.2: Schemat przycinania drzewa decyzyjnego

wybranych poddrzew przez liście, którym przypisuje się etykietę kategorii najczęściej występującej wśród związanych z nim przykładów. Zmniejszenie rozmiaru drzewa, które pociąga za sobą uproszczenie jego konstrukcji, pomimo że powoduje obniżenie jakości klasyfikacji na zbiorze treningowym, podczas docelowej klasyfikacji daje znacznie lepsze rezultaty. Drzewa uzyskane po przycięciu, poza ewentualnym zmniejszeniem błędu rzeczywistego, są czytelniejsze dla człowieka, wymagają mniejszej ilości pamięci i są efektywniejsze podczas procesu klasyfikacji. Algorytm przycinania drzewa decyzyjnego przedstawiony został na schemacie. (Algorytm 1.2).

Czasami stosuje się bardziej zaawansowaną metodę przycinania (tzw. subtree raising lub przycinanie od środka), w której przycięcie węzła nie oznacza zastąpienia go liściem a jednym z jego węzłów potomnych. Pociąga to bardzo często za sobą konieczność reorganizacji całej struktury poddrzewa. Taka operacja jest bardzo kosztowna i dlatego niezbyt często stosowana [5]. Działanie obu metod zostało zaprezentowane na schemacie (Rysunek 1.2).

Procesy przycinania drzewa potrzebują danych, na podstawie których oszacowany zostanie błąd rzeczywisty drzewa. Jednym z podejść jest szacowanie z użyciem oddzielnego zbioru przycinania, złożonego z przykładów z poza zbioru trenującego. Można stosować je wówczas, kiedy dostępna jest dostatecznie duża liczba przykładów. Rzadko jednak jesteśmy

w tak komfortowej sytuacji. Ocena kształtu drzewa dająca największe prawdopodobieństwo uogólniania problemu jest zwykle dokonywana przez uczenie kroswalidacyjne lub podobne techniki (kroswalidacja Monte Carlo, bagging itp.).



Rysunek 1.2: Dwie metody przycinania drzewa

Rozdział 2

Przegląd drzew decyzyjnych

2.1 ID3

Algorytm ID3 oparty jest na teorii informacji. Kryterium oceniającym podziały jest tutaj *zysk informacyjny* (ang. information gain)

$$\Delta I(S, W) = I_E(W) - \sum_i p_i I_E(W_i^S)$$

gdzie W_i^S są podwęzłami węzła W wynikającymi z podziału S oraz p_i jest stosunkiem liczby wektorów zbioru treningowego wpadających do węzła W_i do liczby wektorów wpadających do węzła W.

Miarą niejednorodności węzła w zysku informacyjnym jest *entropia*

$$I_E(W) = -\sum_{c \in C} P(c|W) \log_2 P(c|W)$$

W kolejnych rekurencyjnych przebiegach algorytmu węzły dzielone są na podwęzły, aż do uzyskania maksymalnego drzewa. Podczas każdego podziału tworzonych jest tyle węzłów potomnych ile wartości ma najbardziej informatywna cecha, czyli ta oferująca największą redukcję entropii. Niekorzystną konsekwencją takiej strategi jest tendencja do preferowania cech, które mają dużą w stosunku do innych liczbę wartości. Drugim bardzo dużym ograniczeniem jest wymóg dyskretności wszystkich cech przestrzeni klasyfikacji. Jeżeli chcemy użyć metody ID3 do problemów z przestrzeni cech ciągłych, musimy dokonać wstępnej dyskretyzacji. Sytuację dodatkowo komplikuje fakt, że metoda jest bardzo wrażliwa na jakość dyskretyzacji i może dawać bardzo różne rezultaty dla różnych jej odmian.

Wszystko to sprawia, że ID3 jest rzadko wykorzystywany do zastosowań praktycznych, gdzie czasami można go spotkać w rozbudowanej wersji NewID[14]. Dał on jednak solidne podstawy dla jednego z najpopularniejszych i najszerzej stosowanych algorytmów drzew decyzji - metody C4.5. [5]

2.2 C4.5

Metoda C4.5 podobnie jak jej poprzedniczka ID3 korzysta z teorii informacji. W dużej części oba algorytmy są identyczne, C4.5 rozbudowana została jednak o elementy, które nadają jej zupełnie nową jakość. Zostały wprowadzone następujące zmiany:

- modyfikacja miary nieczystości węzłów,
- możliwość wykorzystania ciągłych atrybutów,
- metoda oczyszczania (przycinania) drzewa,
- możliwość klasyfikacji niepełnych danych.

W C4.5 zamiast zysku informacyjnego stosuje się tzw. *względy zysk* (ang. gain ratio). Ma to zapobiegać niepożądanemu efektowi preferowania atrybutów o dużej liczbie możliwych wartości.

Względny zysk określamy wzorem:

$$\Delta I'(S,W) = \frac{\Delta I_E}{SI(S,W)} \quad ,$$

gdzie

$$SI(S,W) = -\sum_i p_i \log_2 p_i$$

W powyższym równaniu p_i określa stosunek liczby wektorów wpadających do i-tego podwęzła do liczby wektorów w węźle W, a zatem $SI(S,W)$ (ang. Split Information) jest entropią rozkładu wynikającego z podziału S.

W przypadku cech ciągłych rozpatrywane są wszystkie możliwe podziały na dwa zbiory $(-\infty, a]$ oraz (a, ∞). W przeciwieństwie do atrybutów dyskretnych, ciągłe mogą pojawić się na kilku poziomach tej samej gałęzi drzewa. Dla każdego podziału określa się wartość względnego zysku informacyjnego i wybiera ten, który wypada najlepiej.

Przycinanie drzewa oparte zostało na statystycznej ocenie istotności różnicy błędu klasyfikacji dla danego węzła i jego węzłów potomnych. Zakładając dwumianowy rozkład liczby błędów szacuje się prawdopodobieństwo zmniejszania błędu i obcina podziały, dla

których prawdopodobieństwo nie przekracza zadanego progu. Istnieje też wersja algorytmu przycinania, w której zamienia się poddrzewo o korzeniu w danym węźle jego najlepszym poddrzewem.

C4.5 radzi sobie z brakującymi wartościami w danych. Podczas budowania drzewa ocena zysku nie uwzględnia danych, dla których brakuje wartości badanej cechy, a wyliczony zysk skaluje się mnożąc przez częstość występowania wartości tej cechy w próbie treningowej.

Obecnie istnieje także zmodyfikowana wersja algorytmu C4.5 zwana C5 lub też See5. Ponieważ jest to jednak produkt komercyjny, popularność jego jest o wiele mniejsza, a rezultaty jego zastosowań nie są powszechnie dostępne. [8, 5, 11]

2.3 CART

Opracowana w 1984 roku metoda CART (Classification and regression trees) jest nadal bardzo popularna i szeroko stosowana. Drzewa decyzyjne tworzone przez ten algorytm są binarne i zawierają dokładnie dwie gałęzie w każdym z węzłów.

CART może być stosowane zarówno do danych ciągłych jak i dyskretnych. W przypadku danych ciągłych stosuje się technikę zbliżoną do tej z algorytmu C4.5 - rozpatruje się wszystkie podziały dwuprzedziałowe zdeterminowane punktem podziału *a*.

Jako kryterium oceniające jakość podziałów CART wykorzystuje przyrost czystości węzłów. Sugerowaną przez autorów miarą nieczystości jest tzw. *Gini index:*

$$I_G = 1 - \sum_{c \in C} P(c|W)^2$$

Można także stosować znaną z ID3 i C4.5 entropię.

Niezależnie od użytej miary, można stosować technikę binaryzacji. CART dokonuje wówczas podziału danych na dwie superklasy i następnie dla nich ocenia jakość podziałów. System próbuje tak połączyć klasy w grupy, aby były one w miarę równoliczne.

W algorytmie CART silnie rekomendowane jest niestosowanie kryteriów stopu, na rzecz przycinania w pełni rozrośniętego drzewa. [8, 11, 12]

2.4 FACT i QUEST

Tradycyjne algorytmy bazujące na algorytmach szukania zachłannego niosą ze sobą dwa dosyć istotne ograniczenia - sporą złożoność obliczeniową oraz dużą tendencyjność podczas wyboru zmiennych. Algorytmy FACT (Fast Algorithm for Classification Trees) i QUEST (Quick, Unbiased, Efficient, Statistical Tree) to algorytmy parametryczne, wykorzystujące statystyki bazujące na pewnych założeniach co do rozkładu danych. Kluczowym elementem tych algorytmów jest wybór cech podziału, który jest wyraźnie oddzielony od samego podziału. W algorytmie FACT wykorzystuje się do tego celu statystykę F, wybierając cechę dla której wartość tej statystyki jest maksymalna. Analiza dokonywana jest dla cech ciągłych, w przypadku cech dyskretnych dokonywana jest ich konwersja. W algorytmie QUEST dla cech ciągłych także wykorzystywana jest statystyka F, natomiast analiza cech dyskretnych odbywa się z użyciem testu χ^2 niezależności danej cechy i klasy.

Kiedy dana cecha zostanie wybrana, FACT używa liniowej dyskryminacji (LDA) do skonstruowania podziału. Takie rozwiązanie posiada dwie wady. Po pierwsze węzeł dzielony jest na tyle podwęzłów ile jest klas. Jeżeli ta liczba jest duża, może doprowadzić to do zbyt szybkiego podziału danych treningowych i otrzymane drzewo będzie zbyt krótkie aby uzyskać ciekawe informacje z danych. Drugim zauważalnym brakiem jest ignorowanie efektu nierówności wariancji poszczególnych klas.

Obie niedogodności zostały rozwiązane w metodzie QUEST. Aby zapewnić binarność podziałów, klasy grupowane są w dwie superklasy przed zastosowaniem procedury dyskryminacji. Rozwiązaniem drugiego problemu jest zastosowanie dyskryminacji kwadratowej (QDA) w miejsce dyskryminacji liniowej stosowanej w FACT. W wyniku QDA otrzymuje się dwa potencjalne punkty podziału, z których wybiera się ten, który jest bliższy średniej wartości jaką przyjmuje analizowana cecha w populacji wektorów należących do jednego z klastrów.

Kontrola złożoności drzew w algorytmach FACT i QUEST może odbywać się z wykorzystaniem kryteriów stopu. Dodatkowo QUEST potrafi także stosować kroswalidację na wzór systemu CART [8, 24]

2.5 CAL5

Algorytm Cal5 został opracowany z myślą o ciągłych lub dyskretnych uporządkowanych wartościach atrybutów. Istnieją jednak odmiany algorytmów potrafiące radzić sobie z nieuporządkowanymi dyskretnymi wartościami równie skutecznie.

Niech *E* będzie próbką ze zbioru przykładów opisanych przez *n* atrybutów. Cal5 przekształca n – wymiarowe przykłady w przestrzenie reprezentowane przez podzbiory $E_i \in E(i=1,...,n)$, gdzie klasa $c_j(j=1,...,m)$ występuje z prawdopodobieństwem $p(c_j)>\beta$, a $\beta \leqslant 1$ jest progiem decyzji.

Drzewo jest budowane sekwencyjnie. Począwszy od jednego atrybutu, rozwijane jest rekurencyjnie z użyciem następnych, tak długo, aż wystarczające rozróżnienie klas zostanie osiągnięte. Jeżeli w węźle nie może zostać podjęta decyzja o przydzieleniu klasy, gałąź utworzona z użyciem kolejnego atrybutu jest dołączana do drzewa. Kolejność wyboru atrybutu odbywa się na podstawie jednej z dwóch metod: statystycznej lub opartej na entropii. W przypadku metody entropowej stosuje się omawiany wcześniej (znany z takich drzew jak ID3) [19] współczynnik przyrostu informacji (*Information Gain*). Podejście statystyczne nie używa rezultatów przeprowadzanej dyskretyzacji. Dla atrybutów ciągłych współczynnik (*quotient)* [13]

$$quotient(N)=\frac{A^2}{A^2+D^2}$$

jest miarą dyskryminacji, gdzie *A* jest odchyleniem standardowym przykładów ze zbioru *N* od środka ciężkości wartości atrybutów i *D* jest średnią kwadratów wartości odległości pomiędzy klasami.

Bardzo istotną częścią algorytmu Cal5 jest, oparta na podejściu statystycznym, procedura dyskretyzacji. Wszystkie przykłady $m_l \in E$ w danym węźle *N* są porządkowane (rosnąco) względem wartości nowo wybranego atrybutu a_i . Przedział zawierający uporządkowane wartości atrybutu jest budowany rekursywnie z wartości a_i począwszy od wartości najmniejszych do największych. Nowe wartości dodawane są do przedziału tak długo, aż możliwe będzie podjęcie decyzji o przydzieleniu etykiety klasy, lub będzie pewne (zgodnie z przyjętym poziomem ufności), że taka decyzja nie będzie mogła być podjęta. Powyższe informacje uzyskujemy na drodze testów dwóch hipotez:

H1: Istnieje klasa c_i w przedziale *I* taka, że:

$$p(c_i|N) \geq \beta$$

H2: Dla każdej klasy c_i występującej w I spełniona jest nierówność:

$$p(c_i|N) < \beta$$

gdzie: $p(c_i|N)$ jest warunkowym prawdopodobieństwem w danym węźle *N*, β jest progiem definiowanym przez użytkownika lub w wyniku meta-nadzoru.

Testy hipotez otwierają nam drogę do następujących decyzji:

(a) Jeżeli istnieje klasa c_i dla której H1 jest prawdziwa, to klasa ta jest dominującą w *I.* Przedział I jest zamykany. Powiązana ścieżka w drzewie jest kończona – tworzony jest liść etykietowany klasą c_i

(b) Jeżeli hipoteza H2 jest prawdziwa, stwierdzany jest brak klasy dominującej. Przedział jest zamykany. Niezbędny jest nowy test z użyciem kolejnego atrybutu.

(c) Jeżeli nie może być podjęta ani decyzja (a), ani (b), przedział jest rozszerzany o kolejną wartość atrybutu, oraz wykonywany jest ponowny test hipotez. Jeżeli nie ma kolejnych próbek, przydziela się etykietę klasy większościowej, jako najbardziej prawdopodobną.

W następstwie tej procedury, obserwuje się dużą ilość małych przedziałów I_l z przydzielonymi etykietami klas, lub oznaczonych jako wymagających dalszej procedury. Sąsiadujące ze sobą przedziały I_l, I_{l+1} , etykietowane w ten sam sposób, są łączone. Tak utworzone przedziały, do których przypisane jest unikalna etykieta klasy, podczas budowania drzewa stają się liśćmi. Te, które wymagały dalszej procedury, są optymalnie (w sensie statystycznym) przygotowane do dalszej rozbudowy. [16,17]

Rozdział 3

Reguły logiczne

Spośród wszystkich metod reprezentacji wiedzy stosowanych w maszynowym uczeniu się nie ma takiej, która byłaby bliższa metodom stosowanym do zapisu wiedzy przez ludzi niż reprezentacja regułowa. Zastosowanie drzew decyzyjnych znacznie ułatwia proces generowania reguł. Obiekty są oceniane na podstawie wartości ich atrybutów zgodnie z kolejnością wynikającą ze struktury drzewa. Generowanie reguł na podstawie drzew decyzyjnych umożliwia ich zwarty zapis i znacznie skraca czas potrzebny do wnioskowania. Reguła składa się z *części warunkowej* i *części decyzyjnej***.** Wskazuje ona decyzję właściwą dla sytuacji, gdy spełniona jest część warunkowa [9]. Regułę możemy zapisać następująco:

JEŻELI ***warunki*** **TO** ***decyzja***

3.1 Logiczne podstawy reguł

Reguła w postaci przedstawionej powyżej jest nawiązaniem do logicznej implikacji. Oba pojęcia nie są jednak całkowicie jednoznaczne. W klasycznej logice implikacja tworzy z dwóch logicznych formuł A i B nową formułę $A \rightarrow B$ której przypisuje się wartość logiczną (prawda lub fałsz) na podstawie wartości logicznej A i B. W przypadku reguły logicznej, o ile część warunkową możemy interpretować jako logiczną formułę, to niekoniecznie jest to prawdziwe dla części decyzyjnej tej reguły. W tym drugim wypadku jest to zależne od rodzaju wiedzy reprezentowanej przez regułę i od sposobu jej wykorzystania. Czasami części decyzyjne mogą być interpretowane jako stwierdzenia prawdy lub fałszu, czasami jednak

polegają one na wykonaniu pewnych akcji, którym trudno przypisać jakąkolwiek wartość logiczną [9]. Dla nas najbardziej interesującym typem reguł będą reguły służące do klasyfikowania przykładów. Oznacza to, że część warunkowa określa pewne warunki, które mogą być spełnione przez przykłady lub nie, część decyzyjna określa etykietę kategorii, którą należy przypisać do przykładu. W tym szczególnym przypadku reguła będzie miała postać:

JEŻELI ***warunki*** **TO** ***kategoria***

lub też w wersji skróconej:

warunki $\rightarrow$ ***kategoria***

Obu częściom takiej reguły możemy przypisać interpretację logiczną. Aby móc regułę w tej postaci traktować jako logiczną implikację w sposób ścisły, zmienimy sposób zapisu w celu eliminacji wszelkich niejednoznaczności. Załóżmy, że na dziedzinie X określone są atrybuty $a_1, a_2, \ldots, a_n$. Wówczas warunki nakładane przez regułę na wartości atrybutów przykładu $x \in X$ można reprezentować przez pewną zawierającą te wartości formułę logiczną $\alpha[a_1(x), a_2(x), \ldots, a_n(x)]$. Cześć decyzyjna może być natomiast zapisana jako $h(x) = d$, gdzie $d \in C$, C jest zbiorem kategorii, h oznacza hipotezę reprezentowaną za pomocą reguły. Jeśli reguła taka ma być stosowana do wszystkich przykładów dziedziny, to jej zapis w logicznej notacji może przyjąć następującą postać:

$$(\forall x \in X)(\alpha[a_1(x), a_2(x), \ldots, a_n(x)] \rightarrow h(x) = d)$$

3.2 Reprezentacja warunków

W przypadku reguł logicznych wykorzystywanych do klasyfikacji danych szczególną rolę odgrywa sposób reprezentacji części warunkowej reguły, gdyż zakładamy, że część decyzyjna jest po prostu etykietą kategorii [1]. Do reprezentacji warunków wykorzystamy *kompleksy* - interpretowane jako koniunkcje prostych warunków dla wartości pojedynczych atrybutów, nazywanych *selektorami*.

3.2.1 Selektory

Rozważając cztery rodzaje selektorów: uniwersalny, pojedynczy, dysjunkcyjny i pusty, zdefiniujemy je w sposób pozwalający na jednolite ich traktowanie.

Definicja 3.1 ***(Selektor pojedynczy)***

Mówimy, że selektor pojedynczy jest spełniony dla przykładu *x*, jeśli atrybut a_i dla tego przykładu ma pewną wskazaną wartość z jego przeciwdziedziny $v \in A_i$, a więc warunek równoważny takiemu selektorowi można zapisać jako $a_i(x) = v$. Selektor taki zapisujemy jako tą właśnie wartość *v*.

Definicja 3.2 ***(Selektor dysjunkcyjny)***

Mówimy, że selektor dysjunkcyjny jest spełniony dla przykładu *x*, jeśli atrybut ma jedną z wymienionych wartości z jego przeciwdziedziny $v_{i1}, v_{i2}, \ldots, v_{ik} \in A_i$ przy czym $\{v_{i1}, v_{i2}, \ldots, v_{ik}\} \neq A$, co oznacza, że warunek równoważny takiemu selektorowi można przedstawić jako $a_i(x) = v_{i1} \vee a_i(x) = v_{i2} \vee \ldots \vee a_i(x) = v_{ik}$. Selektor dysjunkcyjny zapisujemy jako listę wartości $v_{i1}, v_{i2}, \ldots, v_{ik}$ oddzielonych symbolem alternatywy $v_{i1} \vee v_{i2} \vee \ldots \vee v_{ik}$.

Definicja 3.3 ***(Selektor uniwersalny)***

Mówimy, że selektor uniwersalny jest spełniony dla przykładu *x* przez dowolną wartość atrybutu a_i z jego przeciwdziedziny. Warunek równoważny takiemu selektorowi można zapisać jako $a_i(x) \in A_i$. Selektor taki będziemy reprezentowali za pomocą symbolu *?*.

Definicja 3.4 ***(Selektor pusty)***

Selektor pusty nie jest spełniony dla przykładu *x* przez żadną wartość atrybutu a_i z jego dziedziny. Selektor pusty będzie reprezentowany za pomocą symbolu $\emptyset$.

Z każdym selektorem możemy powiązać zbiór wartości *dozwolonych* odpowiadającego mu atrybutu.

Definicja 3.5 ***(Zbiór wartości dozwolonych związany z selektorem)***

Z dowolnym selektorem *s* odpowiadającym atrybutowi $a : X \to A$ jest związany zbiór wartości $V_s \subseteq A$, przy czym :

1. $V_s = \{v\}$ dla selektora pojedynczego *s* = *v*
2. $V_s = \{v_1, \ldots, v_m\} \subset A$ dla selektora dysjunkcyjnego $s = v_1 \vee \ldots \vee v_m$
3. $V_s = A$ dla selektora uniwersalnego *s* = ?

4. $V_s = \emptyset$ dla selektora pustego $s = \emptyset$

Z powyższej definicji wynika, że pomiędzy selektorem *s* a związanym z nim zbiorem dozwolonych wartości V_s istnieje jednoznaczna odpowiedniość, można więc mówić o nich wymiennie. Korzystając z tej obserwacji możemy w prosty sposób zdefiniować relację pokrywania przykładów przez selektory:

Definicja 3.6 ***(Pokrywanie przykładu przez selektor)***
Selektor odpowiadający atrybutowi $a : X \to A$ *pokrywa* przykład $x \in X$, jeśli $a(x) \in V_s$, przy czym V_s oznacza zbiór wartości dozwolonych dla selektora *s*. Piszemy wówczas $s \eta x$.

Selektory pojedyncze i dysjunkcyjne mają zastosowanie dla atrybutów nominalnych lub porządkowych o dyskretnych zbiorach wartości. Użycie tych selektorów dla atrybutów ciągłych teoretycznie jest możliwe, ale z oczywistych względów nie przyniesie pożytecznych rezultatów[1]. Dla tego typu atrybutów pożądane byłby zastosowanie selektorów *nierównościowych*, porównujących wartość danego atrybutu z wartością progową, oraz selektorów *przedziałowych*, określających przedział do którego musi należeć wartość atrybutu.

3.2.2 Kompleksy

Kompleks zapisujemy jako ujętą w trójkątne nawiasy listę selektorów, oddzielonych przecinakami, odpowiadających wszystkim atrybutom określonym na dziedzinie, zakładając, że są one uporządkowane i selektor na pozycji *i* odpowiada *i*-temu atrybutowi. Tak rozumiany kompleks nie jest niczym innym jak tylko wygodnym sposobem zapisu koniunkcji pewnej liczby warunków nakładanych na wartości atrybutów, dla których w kompleksie znajduje sie selektor inny niż uniwersalny. Relacja pokrywania przykładów przez kompleksy wygląda następująco:

Definicja 3.7 ***(Pokrywanie przykładu przez kompleks)***
Kompleks $k = \langle s_1, s_2, \ldots, s_n \rangle$ pokrywa przykład $x \in X$, jeśli selektor s_i dla i = 1,2,...,*n*

pokrywa przykład x. Piszemy wówczas $k \eta x$.

Definicja 3.8 ***(Kompleks sprzeczny)***

Każdy kompleks zawierający przynajmniej jeden selektor pusty $\varnothing$ będzie utożsamiany z zawierającym wyłącznie selektory puste kompleksem $\langle \varnothing, \varnothing, \ldots, \varnothing \rangle$, i nazywany kompleksem sprzecznym lub pustym oraz oznaczany przez $\langle \varnothing \rangle$.

Definicja 3.9 ***(Kompleks uniwersalny)***

Kompleks zawierający wyłącznie selektory uniwersalne *?* będzie nazywany kompleksem uniwersalnym i oznaczany przez $\langle ? \rangle$.

Definicja 3.10 ***(Kompleks atomowy)***

Każdy kompleks zawierający dokładnie jeden selektor pojedynczy lub dysjunkcyjny i prócz niego wyłącznie selektory uniwersalne będziemy nazywać kompleksem atomowym.

Z powyższych definicji wynika oczywista obserwacja: kompleks sprzeczny nie pokrywa żadnego przykładu, a kompleks uniwersalny pokrywa wszystkie przykłady. Kompleks atomowy ogranicza dopuszczalne wartości dokładnie jednego atrybutu.

Przydatna może być także relacja porównywania kompleksów:

Definicja 3.11 ***(Relacja porównywania kompleksów)***

Dla dziedziny X i danych dwóch kompleksów k_1 i k_2 mówimy, że k_1 jest bardziej ogólny niż k_2 (i równoważnie: k_2 jest mniej ogólny niż k_1, k_2 jest bardziej szczegółowy niż k_1, k_1 jest mniej szczegółowy niż k_2) wtedy i tylko wtedy gdy

$$\{x \in X | k_2 \eta x\} \subset \{x \in X | k_1 \eta x\}$$

Piszemy wówczas $k_1 > k_2$, $k_2 < k_1$.

Dla dowolnego zbioru przykładów $P \subseteq X$ i kompleksu k przez P_k będziemy oznaczać podzbiór P złożony z przykładów pokrywanych przez *k:*

$$P_k = \{x \in P | k \eta x\}.$$

Poprzez P_k^d będziemy oznaczali wszystkie przykłady ze zbioru *P,* które są pokrywane przez kompleks *k* i należą do kategorii *d* ustalonego pojęcia docelowego, czyli:

$P_k^d = P^d \cap P_k$, $d \in C$.

3.2.3 Koniunkcje kompleksów i selektorów

Spośród wszystkich operacji logicznych takich jak koniunkcja, alternatywa, negacja, implikacja, itp. w przypadku kompleksów interesować nas będzie jedynie pierwsza z wymienionych. Wynika to z faktu, że tylko ona może być przeprowadzona w taki sposób aby jej wynik dla argumentów będących kompleksami także pozostał kompleksem. Ponieważ każdy kompleks reprezentuje koniunkcję warunków, oczywiste jest, że koniunkcja kompleksów powinna być kompleksem reprezentującym koniunkcję warunków z obu tych kompleksów. Odpowiednią operację określ poniższa definicja:

Definicja 3.12 ***(Koniunkcja kompleksów)***
Koniunkcją kompleksów *k* i *l* jest kompleks, którego każdy selektor jest koniunkcją pary odpowiednich selektorów z tych kompleksów. Jeśli $k = \langle s_1^k, s_2^k, \ldots, s_n^k \rangle$ i $l = \langle s_1^l, s_2^l, \ldots, s_n^l \rangle$, to $k \wedge l = \langle s_1^k \wedge s_1^l, s_2^k \wedge s_2^l, \ldots, s_n^k \wedge s_n^l \rangle$.

Powyższa definicja odwołuje się do koniunkcji selektorów, której sposób realizowania zależy od rodzaju tych selektorów. W ogólnym przypadku możemy ją zdefiniować następująco:

Definicja 3.13 ***(Koniunkcja selektorów)***
Koniunkcją selektorów s_1 i s_2 odpowiadających atrybutowi $a : X \to A$ jest odpowiadający temu atrybutowi selektor *s* o zbiorze wartości dozwolonych $V_s = V_{s_1} \cap V_{s_2}$.

Operacja koniunkcji kompleksów jest wykorzystywana jako mechanizm ich specjalizacji przez dodanie dodatkowych warunków. W ogólnym przypadku specjalizacja będzie przeprowadzana dla zbioru kompleksów za pomocą operacji przecięcia takich zbiorów:

Definicja 3.14 ***(Przecięcie zbioru kompleksów)***
Przecięciem zbioru kompleksów *K* i *L* jest zbiór wszystkich koniunkcji dwóch kompleksów odpowiednio z tych dwóch zbiorów: $K \cap L = \{k \wedge l | k \in K, l \in L\}$.

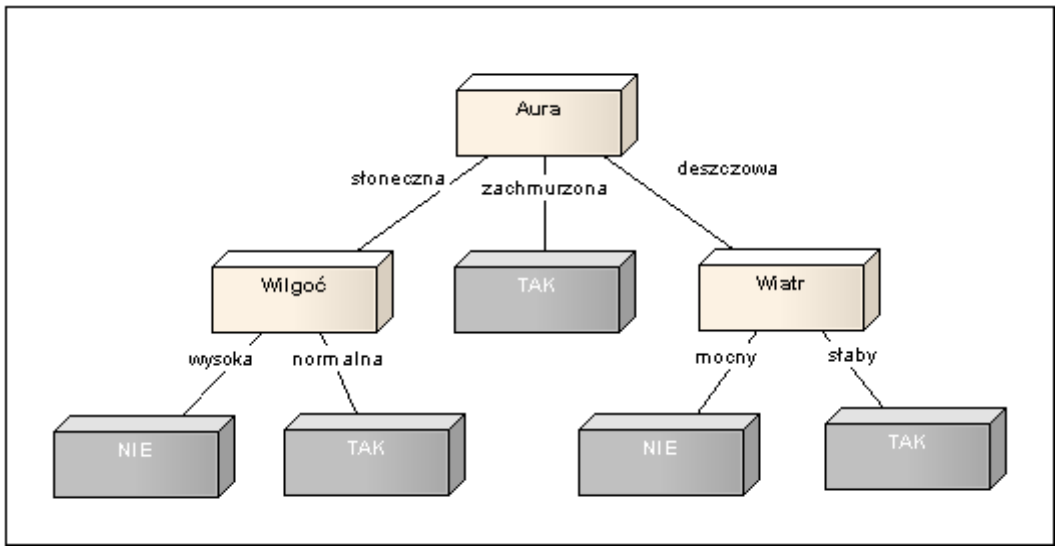
3.3 Konwersja drzew decyzji do postaci reguł logicznych

W tej części pracy przyjrzymy się problemowi budowy zbioru reguł logicznych na podstawie skonstruowanego wcześniej drzewa decyzyjnego. Wyselekcjonowany w ten sposób zestaw reguł jest w pełni zrozumiały dla człowieka, znajduje również zastosowania w różnego rodzaju aplikacjach, między innymi wykorzystywany jest przez systemy eksperckie.

3.3.1 Algorytm C4.5 Rule

Załóżmy, że w wyniku działania jednego z omawianych wcześniej algorytmów konstrukcji drzew decyzji otrzymaliśmy następujące drzewo (Rysunek 3.1).

Po przeprowadzeniu **prostej ekstrakcji** reguł z powyższego drzewa otrzymujemy:



Rysunek 3.1: Drzewo decyzyjne

JEŻELI (aura = „słoneczna”) **ORAZ** (wilgoć = „duża”) **TO** „Nie”
JEŻELI (aura = „słoneczna”) **ORAZ** (wilgoć = „normalna”) **TO** „Tak”
JEŻELI (aura = „zachmurzona”) **TO** „Tak”
JEŻELI (aura = „deszczowa”) **ORAZ** (wiatr = „silny”) **TO** „Nie”
JEŻELI (aura = „deszczowa”) **ORAZ** (wiatr = „słaby”) **TO** „Tak”

Następnym krokiem w działaniu algorytmu C4.5 Rule jest *generalizacja reguł* polegająca na usunięciu z nich warunków, które nie wpływają na jakość klasyfikacji. Osiąga się to poprzez pesymistyczną estymację dokładności (ang. accuracy) danej reguły. W każdym przebiegu algorytmu badany jest jeden warunek reguły. Oceny dokonujemy poprzez porównanie pesymistycznej estymacji danej reguły z pesymistyczną estymacją reguły, z której rozważany warunek został usunięty. W przypadku, gdy otrzymany rezultat jest nie gorszy niż zakładano, warunek zostaje usunięty. Gdy proces zostanie przeprowadzony dla wszystkich warunków reguły i pesymistyczna estymacja najlepszej z nowo powstałych reguł jest lepsza niż w przypadku wyjściowej, to reguła wyjściowa zastępowana jest nową regułą [18].

Pesymistyczny współczynnik błędu oparty jest na estymowanej dokładności reguły na zbiorze klasyfikacyjnym. Używamy pesymistycznego oszacowania mając na uwadze fakt, że drzewo, które dobrze radzi sobie na zbiorze treningowym, niekoniecznie osiągnie te same wynik w przypadku nowych próbek. C4.5 Rule wyznacza pesymistyczną estymację błędu licząc standardowe odchylenie błędu zarejestrowanego na zbiorze treningowym, zakładając dwumianowy rozkład danych. Wykorzystanie przedziałów ufności jako metody aproksymującej rzeczywisty błąd na podstawie błędu próbki treningowej daje zadowalające rezultaty [1].

N% przedział ufności dla danego parametru p, jest przedziałem, który z N% prawdopodobieństwem zawiera p.

Przedział ufności możemy obliczyć wykorzystując wzór:

$$error_D(h) = error_S(h) \pm Z_n \sqrt{\frac{(error_S(h))(1-error_S(h))}{n}}$$

gdzie: $error_D(h)$ jest pesymistycznym błędem

$error_S(h)$ jest błędem obserwowanym na zbiorze danych

Z_n jest stałą ufności

n jest liczbą przykładów w zbiorze

Pesymistycznym współczynnikiem błędu jest dolny zakres przedziału ufności:

$$error_D(h) = error_S(h) - Z_n \sqrt{\frac{(error_S(h))(1-error_S(h))}{n}}$$

Przedziały ufności dla rozkładu Gaussa:

Przedział ufności (N%)	50%	60%	80%	90%	95%	98%	99%
Stała Z_n	0,67	1,00	1,28	1,64	1,96	2,33	2,58

Przykład 3.1

Jeżeli obserwujemy 5 błędów klasyfikacji na zbiorze zawierającym 100 przykładów, możemy estymować pesymistyczny współczynnik błędu z N = 95

$$error_D(h) = \frac{5}{100} - 1{,}96\sqrt{\frac{(\frac{5}{100})(1-(\frac{5}{100}))}{100}}$$

$error_D(h) = 0{,}05 - 0{,}0427$

$error_D(h) = 0{,}00728$

Generalizacja reguł została przedstawiona na schemacie.(Algorytm 3.1)

Po przeprowadzeniu generalizacji, reguły są *grupowane* zgodnie z przeprowadzaną klasyfikacją.

Przykład 3.2

Zestaw reguł

JEŻELI (aura = „słoneczna”) ORAZ (wilgoć = „duża”) TO „Nie”

JEŻELI (aura = „słoneczna”) ORAZ (wilgoć = „normalna”) TO „Tak”

JEŻELI (aura = „zachmurzona”) TO „Tak”

JEŻELI (aura = „deszczowa”) ORAZ (wiatr = „silny”) TO „Nie”

JEŻELI (aura = „deszczowa”) ORAZ (wiatr = „słaby”) TO „Tak”

zostanie zgrupowane w następujący sposób

Tak:

JEŻELI (aura = „słoneczna”) ORAZ (wilgoć = „normalna”) TO „Tak”

JEŻELI (aura = „deszczowa”) ORAZ (wiatr = „słaby”) TO „Tak”

JEŻELI (aura = „zachmurzona”) TO „Tak”

Nie:

JEŻELI (aura = „słoneczna”) ORAZ (wilgoć = „duża”) TO „Nie”

JEŻELI (aura = „deszczowa”) ORAZ (wiatr = „silny”) TO „Nie”

funkcja generalizujReguły*(R,D)*

argumenty wejściowe:

R - zbiór reguł

D - zbiór danych

wynik: *zbiór reguł R po generalizacji*

generalizujReguły*(R,D)*

dla wszystkich $r \in R$ **wykonaj**

najlepszaReguła = null;

dopóki*(najlepszaReguła !=r)*

e=liczba przykładów ze zbioru D niepoprawnie sklasyfikowanych przez regułę r z R

p=pesymistyczna estymacja dokładności reguły (r,e)

najlepszaReguła=r

najlepszaDokładność=p

N = zbiór możliwych reguł wygenerowanych z reguły r poprzez usunięcie 1 warunku

dla wszystkich $n \in N$ **wykonaj**

j = liczba przykładów niepoprawnie sklasyfikowanych przez regułę n

k = pesymistyczna estymacja dokładności (n,j)

jeśli *(k<p)* **to**

najlepszaReguła = n

najlepszaDokładność = k

koniec jeśli

koniec dla

jeśli *(najlepszaReguła !=r)* **to**

zamień regułę r w zbiorze R najlepsząRegułą

koniec jeśli

koniec dopóki

koniec dla

zwróć R;

Algorytm 3.1: Generalizacja reguł

Rozdział 4

InTrees – moduł drzew decyzyjnych w systemie Intemi

4.1 Koncepcja platformy Intemi

Intemi to platforma dostarczająca rozwiązań realizujących zadania z zakresu inteligencji obliczeniowej. Jej unikalna, modułowa konstrukcja i rozbudowane środowisko programistyczne SDK sprawiają, że jest to doskonałe narzędzie do eksploracji nowych zagadnień oraz zastosowania meta - uczenia w jego pełnym wymiarze.

Struktura Intemi opiera się na *maszynach (machine/learning machine),* które reprezentują algorytmy adaptacyjne. Końcowy efekt procesu adaptacyjnego danego algorytmu w środowisku Intemi nazywamy *modelem (model).*

Model to nie tylko w pełni funkcjonalna jednostka przeprowadzająca zadanie aproksymacji lub klasyfikacji (np. sieć neuronowa, drzewo decyzyjne), ale także dowolna część algorytmu z dobrze określonym *wejściem i wyjściem (input, output).* W Intemi spotykamy modele testujące inne modele, przeprowadzające transformację danych, czy też ładujące dane z pliku lub innych źródeł [6].

Prawdziwa moc, opartej na maszynach, platformy Intemi drzemie w możliwościach łączenia poszczególnych maszyn w złożone struktury za pomocą odpowiednio zdefiniowanych wejść (input) i wyjść (output). Skonfigurowana maszyna może zostać uruchomiona (run) tworząc model. Jego rezultaty mogą zostać przekazane innym modelom lub zdeponowane za pomocą specjalnych mechanizmów w przygotowanym do tego celu repozytorium wyników. Podczas procesu adaptacyjnego maszyny mogą tworzyć instancje innych maszyn i korzystać z wykonanych przez nie obliczeń.

Czasami wygodniej jest traktować złożoną hierarchię maszyn jako pojedynczy obiekt z dobrze określonym wejściem i wyjściem. Taką enkapsulację możemy przeprowadzić używając schematów *(scheme)*. Doskonałym przykładem użycia schematów jest implementacja walidacji krzyżowej klasyfikatorów. Wykorzystujemy do tego celu wyspecjalizowaną maszynę *(repeater)*, która odpowiada za wielokrotny przebieg poszczególnych, często złożonych scenariuszy. Repeater bazuje na koncepcji t*ablic dystrybucji (distribution boards)* oraz *dystrybutorów (distributors)*. Każdy repeater używa zewnętrznej tablicy dystrybucji aby wygenerować wejścia dla poszczególnych przebiegów procesu powtarzania. Tablica dystrybucji generuje pewną liczbę kolekcji wejść, którą udostępnia pozostałym maszynom za pomocą dystrybutorów.

Działanie r*epeatera* przebiega następująco:

- tworzone są instancje tablic dystrybucji, proces przebiega zdefiniowaną w konfiguracji liczbę razy,
- dla każdej tablicy dystrybucji generowana jest zdefiniowana liczba dystrybutorów
- dla każdego dystrybutora konstruowana jest hierarchia maszyn, określona przez schemat, z wejściami kompatybilnymi z dystrybutorem [7].

Na bazie przedstawionych mechanizmów powstał moduł drzew decyzyjnych InTrees.

4.2 Struktura modułu InTrees

Moduł InTrees składa się z dwóch głównych części logicznych. W pierwszej z nich zaimplementowany został zestaw klas oraz maszyn platformy Intemi [6] i stanowi on swojego rodzaju ramy oraz silnik dla implementacji konkretnych algorytmów drzew decyzyjnych, zawartych w części drugiej. Taka konstrukcja modułu gwarantuje jego rozszerzalność i pozwala na szybkie dodawanie kolejnych algorytmów, bez konieczności zagłębiania się w szczegóły implementacji.

Aplikacja wykonana została w technologi .Net, a do jej napisania użyty został Microsoft Visual Studio 2005 [15]. Do sporządzenia projektu wykorzystano Enterprise Architect 7.0 [20].

4.3 Maszyny w module InTrees

Silnik modułu składa się z czterech maszyn systemu Intemi:

InTree

Implementowane interfejsy: IMachine, IClassifier

Deklarowane wejścia: Dataset (IDataTable, ITargets)

Deklarowane wyjścia: Classifier (IClassifier)

Główna maszyna modułu, podczas swojej pracy wykorzystuje dwie pod-maszyny: DT oraz DTValidator. Jej podstawowymi zadaniami są utrzymywanie spójności konfiguracji, dobór parametrów przycinania z użyciem walidacji krzyżowej oraz kontrola całego procesu uczenia się i klasyfikacji. Dla modeli zewnętrznych właśnie ta maszyna występuje w roli klasyfikatora.

DT

Implementowane interfejsy: IMachine

Deklarowane wejścia: Train dataset (IDataTable, ITargets)

Deklarowane wyjścia: Decision Tree (DecisionTree)

Output train dataset (IDataTable)

Maszyna odpowiedzialna za budowę drzewa decyzyjnego zgodnie z zadaną konfiguracją, działa pod kontrolą maszyny InTree.

DTValidator

Implementowane interfejsy: IMachine

Deklarowane wejścia: Train dataset (IDataTable, ITargets),

Test dataset (IDataTable, ITargets),

Decision Tree (DecisionTree)

Deklarowane wyjścia: Pruning table (IPruningTable)

Zadaniem tej maszyny jest zebranie wyników przycinania w procesie walidacji krzyżowej z automatycznym doborem parametrów. Na podstawie efektów jej działania InTree podejmuje decyzję o wyborze optymalnego stopnia przycinania drzewa.

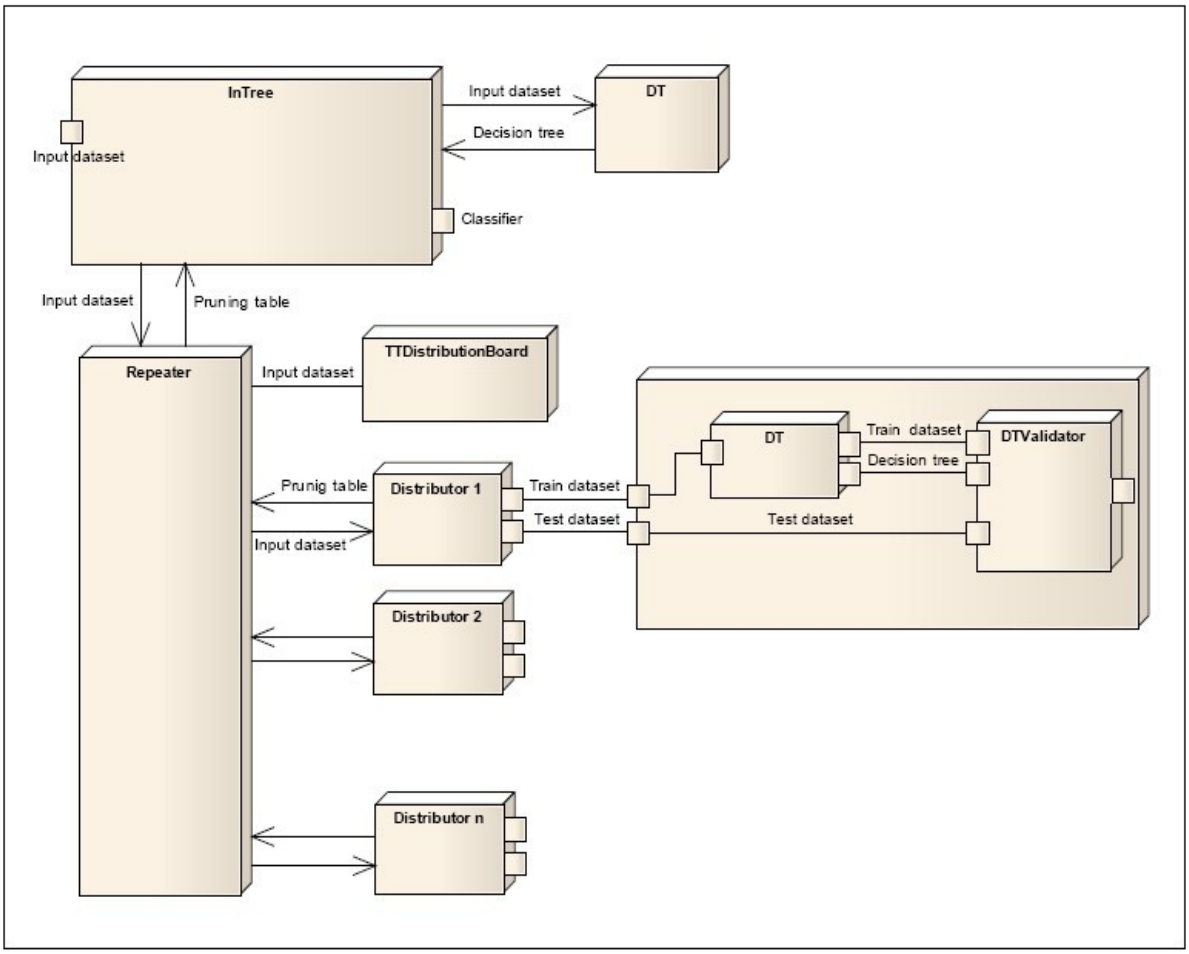
MetaTree

Implementowane interfejsy: IMachine

Deklarowane wejścia: Dataset (IDataTable, ITargets)

Deklarowane wyjścia: Classifier (IClassifier)

Maszyna przeznaczona do doboru elementów algorytmu budowy drzewa na poziomie meta. Z punktu widzenia użytkownika końcowego działa jak klasyfikator. Przed zbudowaniem końcowego drzewa wykonywana jest seria testów z użyciem walidacji krzyżowej w celu identyfikacji optymalnego zestawu elementów algorytmu oraz ich parametrów.



Rysunek 4.1: Schemat maszyn w module InTrees

4.4 Kluczowe interfejsy i klasy silnika modułu

Podczas tworzenia silnika InTrees zastosowano podejście modułowe opierając się na podstawowych elementach algorytmu budowy drzewa metodą TDIDT (Iteracyjna zstępująca budowa drzewa decyzyjnego). Przy takim założeniu w sposób naturalny dało się wyróżnić następujące obiekty:

INodeTestGenerator

Interfejs odpowiedzialny za wygenerowanie zbioru podziałów dla przykładów dostępnych na danym etapie budowy drzewa.

INodeTestSelector

Interfejs odpowiedzialny za wybór najlepszego (według określonego kryterium) testu i dokonaniu za jego pomocą podziału danych. Jest to najważniejszy element algorytmu budowy drzewa decyzyjnego, stanowiący niejako jego serce i w znacznej mierze decydujący o kształcie przyszłego drzewa.

IStoppingCriterium

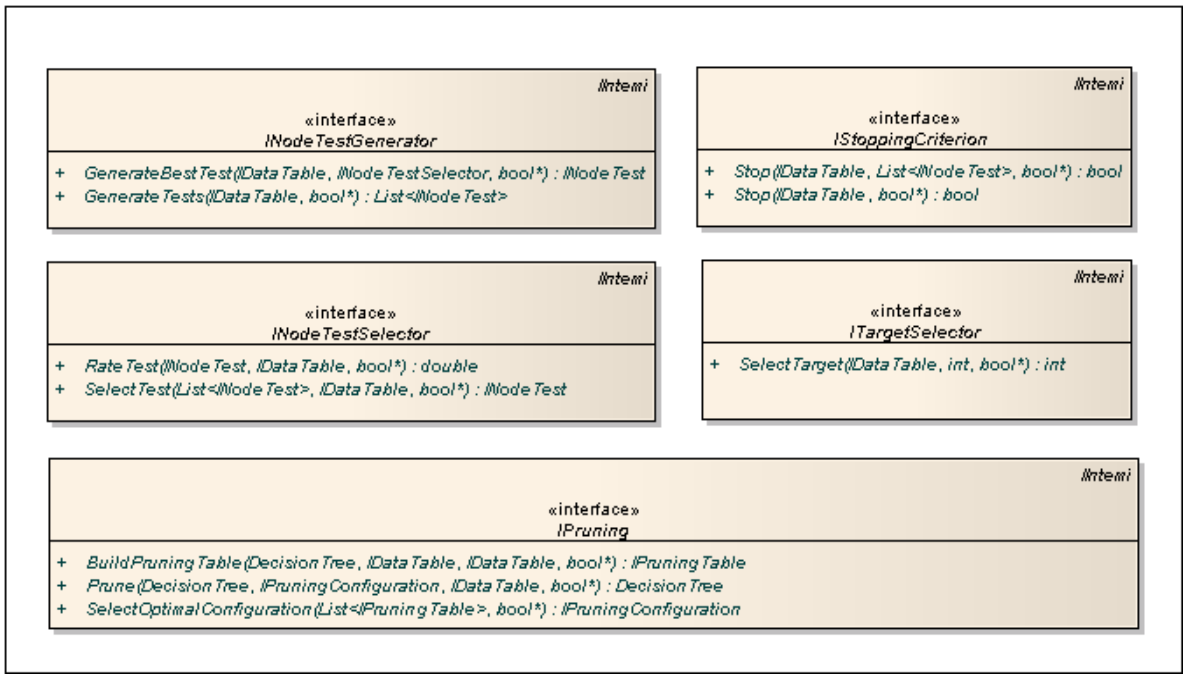
Interfejs definiujący kryterium (lub kryteria) stopu dla algorytmu budowy drzewa.

ITargetSelector

Interfejs odpowiedzialny za wybór etykiety domyślnej w danym zbiorze przykładów.

IPruning

W tym interfejsie określone zostały metody, używane w procesie przycinania drzewa.



Rysunek 4.2: Kluczowe interfejsy silnika modułu InTrees

Dodatkowo zdefiniowane zostały interfejsy:

INodeTest

Interfejs definiujący obiekt testu w danych węźle.

IPruningConfiguration

Interfejs pomocniczy używany przy przycinaniu drzewa – konfiguracja przycinania

Rysunek 4.3: Klasa INodeTest

A także klasy:

DTNode

Węzeł drzewa decyzyjnego.

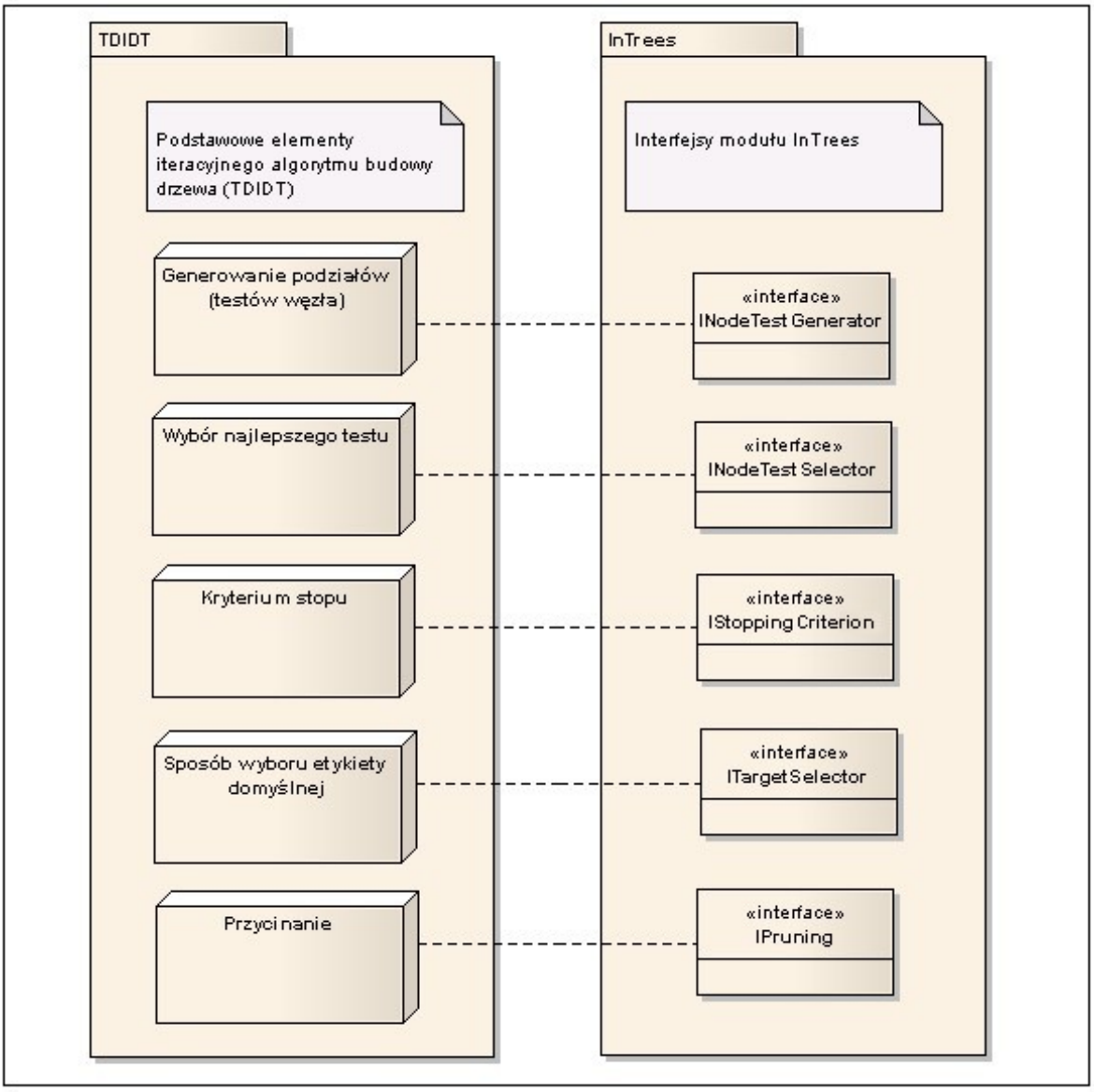
DecisionTree

Drzewo decyzyjne.

Rysunek 4.3: Klasy DecisionTree i DTNode

TDIDT

Klasa odpowiedzialna za proces budowy drzewa decyzyjnego, implementująca algorytm budowy drzewa.



Rysunek 4.4: Interfejsy modułu a algorytm TDIDT

Zastosowanie interfejsów oraz jednolita budowa pozwala użytkownikowi na swobodny dobór poszczególnych części algorytmu. Proces ten może odbywać się również na

drodze meta-uczenia. Niewątpliwą zaletą jest także stosunkowo łatwe tworzenie nowych rozwiązań i kompleksowe testy w różnych konfiguracjach.

4.5 Implementacje algorytmów w module InTrees

Kolejnym etapem budowy modułu, było stworzenie zestawu klas implementujących interfejsy InTrees.

Implementacje interfejsu INodeTestGenerator:

MultiSplitTestGenerator

Sposób generowania podziałów węzła charakterystyczny dla algorytmu C4.5. W przypadku atrybutów nieuporządkowanych tworzona jest oddzielna gałąź dla każdej jego wartości. W przypadku wartości uporządkowanych tworzone są przedziały wartości zgodne z definiowanym progiem tolerancji. [11]

BinaryTestGenerator

Klasa generująca binarne podziały zbioru przykładów wpadających do danego węzła drzewa. W przypadku atrybutów nieuporządkowanych o zbiorze wartości większym niż dwa, generowana jest seria podziałów według schematu (wybrany atrybut - pozostałe atrybuty), dla każdego kolejnego atrybutu. Jest to sposób podziału charakterystyczny dla algorytmu CART. [11]

Implementacje interfejsu INodeTestSelector:

GiniNTS

Klasa odpowiedzialna za wybór optymalnego testu i dokonanie za jego pomocą podziału zbioru przykładów przyporządkowanych do danego węzła drzewa. Jako kryterium wyboru podziałów wykorzystywany jest przyrost czystości węzłów. Sugerowaną miarą nieczystości jest *Gini index* [12]

InfoGainNTS oraz **InfoGainRatioNTS**

Kryterium oceniającym podziały jest tutaj *zysk informacyjny* (ang. information gain) wywodzący się z algorytmu ID3. Tego typu podejście charakteryzuje się jednak skłonnością do preferowania atrybutów o dużej liczbie możliwych wartości [8]. Aby zapobiec temu niepożądanemu zjawisku wprowadzone zostało kolejne kryterium – *względny zysk* (ang. gain ratio) szeroko stosowane w drzewach typu C4.5. [11]

Implementacje interfejsu ITargetSelector:

TargetSelector

W tej implementacji jako etykieta domyślna wskazywana jest etykieta większościowa.

Implementacje interfejsu IStoppingCriterion:

NaturalSC

Jest to klasa implementująca tzw. naturalne kryterium stopu. Decyzja o zaprzestaniu dalszej rozbudowy drzewa podejmowana jest w sytuacji gdy zachodzi przynajmniej jeden z warunków:

- zbiór potencjalnych testów jest pusty
- w zbiorze przykładów wpadających do danego węzła drzewa znajduje się nie więcej niż jeden element
- wszystkie przykłady w wpadające do danego węzła należą do tej samej klasy

Przytoczone warunki stopu stanowią swojego rodzaju bazę, którą można uzupełniać o dodatkowe, bardziej wyszukane kryteria:

MinNodeSizeSC

Kryterium określa minimalną wielkość węzła, rozumianą jako ilość przykładów wpadających do danego węzła drzewa. Jeżeli ilość przykładów osiągnie (lub spadnie poniżej) minimalną wielkość, podejmowana jest decyzja o zaprzestaniu rozbudowy drzewa.

NodePuritySC

Algorytm zaprzestaje rozbudowy drzewa gdy osiągnięty jest zdefiniowany w kryterium stopień czystości węzła.

ComiteeSC

Klasa implementująca komitet kryteriów stopu. Można wykorzystywać w niej wszystkie kryteria implementujące interfejs IStoppingCriterium. W zależności od przyjętej konfiguracji warunki mogą pracować jako alternatywa lub jako koniunkcja zadanych kryteriów stopu.

4.6 Implementacja przycinania drzewa

W module InTrees zaimplementowane zostało przycinanie drzewa automatycznie ustalające optymalne parametry przycinania z wykorzystaniem walidacji krzyżowej. Stworzone zostały interfejsy IPruning, IPruningConfiguration oraz IPruningTable, które umożliwiają zaawansowanym użytkownikom implementacje własnych metod przycinania drzewa bazujących na wspólnym schemacie ustalania parametrów przycinania na poziomie meta.

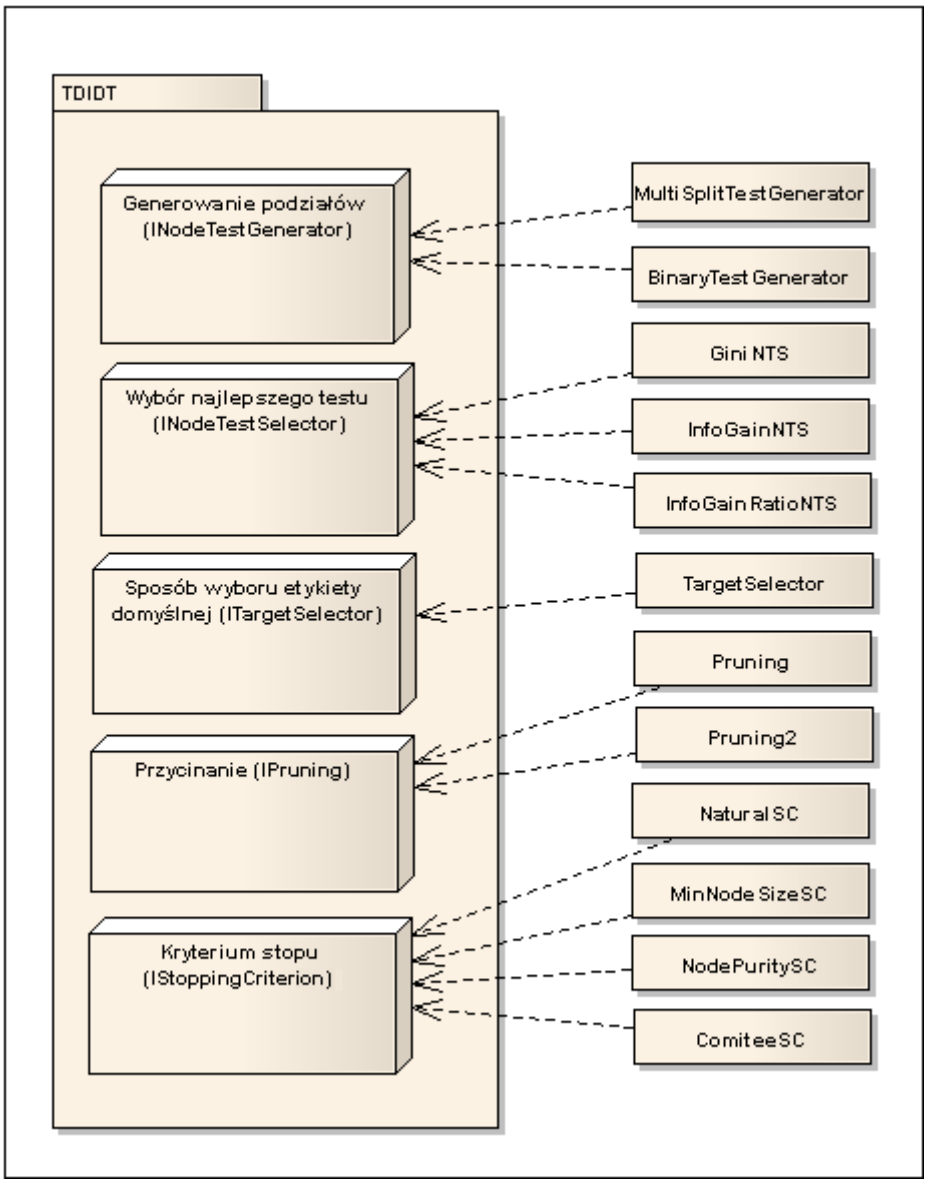
W obecnej wersji InTrees zaimplementowane zostało przycinanie drzewa z wykorzystaniem parametru optymalnej ilości węzłów drzewa dla danego zbiory danych. Właśnie ten parametr dobierany jest podczas procesu meta uczenia modelu z wykorzystaniem walidacji krzyżowej.

Proces ustalania optymalnej ilości węzłów przebiega w następujący sposób:

1. Budowane jest w pełni rozrośnięte drzewo. Zaleca się używanie najprostszego naturalnego kryterium stopu.
2. Wybierany jest węzeł ostatniego poziomu (wszystkie jego dzieci są liśćmi), który ma największą dokładność klasyfikacji (ang. accuracy) i jest on zastępowany liściem z etykietą klasy większościowej.
3. Sprawdzana jest dokładność klasyfikacji utworzonego w ten sposób drzewa i wynik zapisywany jest do tablicy przycinana (klasa implementująca interfejs IPruningTable).
4. Kroki 2 i 3 są powtarzane dopóki poziom drzewa jest większy od 1.
5. Otrzymywana jest tablica przycinania zawierająca zbiór par: (ilość węzłów, dokładność klasyfikacji na zbiorze testowym)

Cały procedura 1 - 5 jest powtarzana w procesie walidacji krzyżowej. Wyniki w postaci tablic przycinania dla każdego przebiegu walidacyjnego są konsolidowane i wybierana jest optymalna ilość węzłów w drzewie dla danego zbioru danych.

Budowane jest ostateczne drzewo decyzyjne, następnie jest ono przycinane do ustalonej, optymalnej ilości węzłów.



Rysunek 4.5: Implementacje podstawowych interfejsów silnika modułu InTrees

Rozdział 5

Eksperymenty

Nowe podejście do algorytmów budowy drzewa, traktujące je jako złożenie kilku powiązanych ze sobą elementów, w sposób naturalny rodzi następujące pytania:
- Które elementy algorytmu mają kluczowe znaczenie dla jakości klasyfikacji ?
- Jak wpłynie na klasyfikację zastąpienie elementu charakterystycznego dla danej metody, elementem zmodyfikowanym lub używanym w algorytmie konkurencyjnym ?
- Czy możliwe jest podniesienie jakości klasyfikacji na drodze automatycznego doboru poszczególnych elementów algorytmu na podstawie próbek zbioru danych ?

Celem tego eksperymentu jest znalezienie przynajmniej częściowych odpowiedzi na postawione pytania.

5.1 Procedura testowa

Do przeprowadzenia eksperymentów zostało wybrane sześć klasyfikatorów różniących się podstawowymi elementami algorytmu budowy drzewa takimi jak: kryterium generowania podziałów, kryterium wyboru testów, kryterium stopu i przycinani. Siódmym klasyfikatorem jest Meta – Drzewo, które na etapie uczenia automatycznie dobiera najbardziej pożądany zestaw elementów algorytmu. Dodatkowo dla części zbiorów danych użyto także wyników uzyskanych przez drzewo SSV, które jest jednym ze standardowych klasyfikatorów zaimplementowanych na platformie Intemi. Do badań wykorzystano ogólnie dostępne zbiory benchmarkowe [23]: Iris flower, Mushroom, Wisconsin Breast Cancer oraz Thyroid.

Klasyfikatory testowane są za pomocą 10 - krotnej walidacji krzyżowej, skonfigurowanej tak, aby każdy z algorytmów pracował na identycznych danych. Cała procedura powtarzania jest 10 razy (10x10CV). Istotność różnic osiąganych wyników oceniono za pomocą testów statystycznych dla prób zależnych: testu T [22] oraz testu kolejności par Wilcoxona [21].

Weryfikowane były następujące hipotezy:

H_0*: Testowane modele nie różnią się jakością klasyfikacji,*

wobec hipotezy alternatywnej

H_1*: Jakość klasyfikacji testowanych modeli jest różna.*

Na kartach wyników zaprezentowane zostały podstawowe rezultaty osiągane przez każdy z algorytmów (dokładność minimalna, średnia i maksymalna oraz odchylenie standardowe), zestawienie wyników dla każdej pary algorytmów przy użyciu trzech testów statystycznych oraz porównanie par algorytmów w świetle każdego z testów.

5.2 Zbiory danych

Iris flower

Zbiór danych zawierający opis 50 próbek każdego z trzech gatunków kwiatu Irys (Setosa, Versicolour, Virginica). Próbka została opisana za pomocą czterech atrybutów numerycznych (długość liścia, szerokość liścia, długość płatka, szerokość płatka) oraz etykiety klasy tożsamej z gatunkiem rośliny.

Rozkład klas:

33,3 % (50 przypadków) – klasa 1 “Iris Setosa”

33,3 % (50 przypadków) – klasa 2 “Iris Versicolour”

33,3 % (50 przypadków) – klasa 3 “Iris Virginica”

Mushroom

W zbiorze zawarty został opis 8124 próbek odnoszących się do 23 gatunków grzybów kapeluszowych. Każdy gatunek zdefiniowany została jako jadalny, trujący lub niepolecany. Dwie ostatnie kategorie połączone zostały w jedną klasę. Każda próbka opisana została za pomocą 22 nominalnych atrybutów.

Rozkład klas:
52% (4208 przypadków) – klasa 1 “jadalny”
48% (3916 przypadków) – klasa 2 “trujący”

Breast Cancer Wisconsin

Zbiór zawiera dane uzyskane podczas badania materiału cytopatologicznego pobranego z guza piersi. Zbiór zawiera 699 przypadków opisanych 9 atrybutami numerycznymi, podzielonych na dwie klasy odpowiadające postawionej diagnozie: łagodne i złośliwe.
Rozkład klas:
65,5 % (458) - klasa 1 “łagodne”
34,5 % (241) – klasa 2 “złośliwe”

Thyroid

W tym zbiorze opisanych zostało 3772 wyników badań gruczołu tarczycy opisanych za pomocą 21 atrybutów i podzielonych na 3 klasy. 15 atrybutów ma charakter binarny, pozostałe są uporządkowanymi atrybutami numerycznymi.
Rozkład klas:
2, 47 % (93) – klasa 1
5,06 % (191) – klasa 2
92,47 % (3488) – klasa 3

5.3 Testowane konfiguracje klasyfikatorów

W opisie konfiguracji klasyfikatorów użyto nazw kryteriów które zostały omówione w rozdziale 4.

Drzewo T1

Drzewo wzorowane na popularnym algorytmie CART, zarówno w przypadku wartości uporządkowanych jak i nieuporządkowanych zbiór przykładów dzielony jest na dwie części. Kryterium wyboru najlepszego podziału jest indeks Gini'ego. Zastosowano naturalne kryterium stopu, nadmiernemu dopasowaniu do danych ma zapobiegać przycinanie z

automatycznym doborem parametru.
Kryterium generowania podziałów: *BinaryTestGenerator*
Kryterium wyboru testów: *GiniNTS*
Kryterium stopu: *NaturalSC*
Przycinanie: *Z automatycznym doborem parametru przycinania*

Drzewo T2

W inspirowanym algorytmem C4.5 drzewie użyto algorytmu generowania podziałów, który dla cech nieuporządkowanych tworzy tyle potencjalnych gałęzi drzewa, ile wartości przyjmuje dana cecha. Kryterium wyboru optymalnego testu jest względny zysk informacyjny [11]. Kryterium stopu oraz przycinanie pozostaje niezmienione w odniesieniu do drzewa T1. Ewentualne różnice w jakości klasyfikacji drzew T1 i T2 wynikać więc będą z efektywności dwóch pierwszych elementów algorytmu.
Kryterium generowania podziałów: *MultiSplitTestGenerator*
Kryterium wyboru testów: *InfoGainRatioNTS*
Kryterium stopu: *NaturalSC*
Przycinanie: *Z automatycznym doborem parametru przycinania*

Drzewa T3 oraz T4 uzyskaliśmy z konfiguracji drzew T1 oraz T2 poprzez wzajemną zamianę najistotniejszego elementu algorytmu budowy drzewa – kryterium wyboru testów.

Drzewo T3

Kryterium generowania podziałów: *BinaryTestGenerator*
Kryterium wyboru testów: *InfoGainRatioNTS*
Kryterium stopu: *NaturalSC*
Przycinanie: *Z automatycznym doborem parametru przycinania*

Drzewo T4

Kryterium generowania podziałów: *MultiSplitTestGenerator*
Kryterium wyboru testów: *GiniNTS*
Kryterium stopu: *NaturalSC*
Przycinanie: *Z automatycznym doborem parametru przycinania*

Drzewo T5

W tym drzewie zrezygnowano z przycinania na rzecz bardziej restrykcyjnego kryterium stopu. Użyto kryterium czystości węzła na poziomie 80%. Pozostała konfiguracja jest zgodna z T2.

Kryterium generowania podziałów: *MultiSplitTestGenerator*

Kryterium wyboru testów: *InfoGainRatioNTS*

Kryterium stopu: *NodePuritySC (80%)*

Przycinanie: *brak*

Drzewo T6

Konfiguracja analogiczna do T1, zrezygnowano jednak z przycinania. Zastosowano stosunkowo wczesne kryterium stopu składające się z alternatywy kryterium czystości węzła na poziomie 70% oraz minimalnej wielkości węzła z parametrem 6.

Kryterium generowania podziałów: *BinaryTestGenerator*

Kryterium wyboru testów: *GiniNTS*

Kryterium stopu: *ComiteeSC (alternatywa): NodePuritySC (70%) lub MinNodeSizeSC (6)*

Przycinanie: *brak*

Meta – Drzewo

Ideą Meta – Drzewa było stworzenie klasyfikatora, automatycznie dobierającego dostępne kryteria algorytmu w zależności od charakteru i specyfiki danych. Aby obiektywnie ocenić jego przydatność w procesie klasyfikacji, zostało ono skonfigurowane tak, aby dla konkretnych danych automatycznie wybierało najbardziej optymalny algorytm spośród wymienionych powyżej drzew od T1 do T6. Spodziewanym rezultatem jest osiąganie wyników nieustępujących w znacznym stopniu tym, które w konkretnym przypadku były udziałem najlepiej dobranego (do danych) drzewa.

Drzewo oparte na kryterium SSV

Drzewo to nie wchodzi w skład pakietu InTrees, jest ono jednym z klasyfikatorów zaimplementowanych w platformie Intemi. Kryterium SSV to kryterium, które próbuje w

bardziej naturalny sposób (w porównaniu z metodami teorii informacji) oceniać separowalność obiektów z różnych klas. Szczegółowy opis kryterium i jego zastosowania w drzewie decyzyjnym można znaleźć w jednym z materiałów referencyjnych [8]. Wyniki uzyskane przez ten klasyfikator w czasie eksperymentów stanowią dobry materiał porównawczy i są punktem odniesienia dla pozostałych wyników.

5.4 Wyniki eksperymentów

Pierwszym zbiorem danych wykorzystanym podczas eksperymentów jest klasyczny zbiór *Iris Flowers***.** Dane zawarte w tym zbiorze są stosunkowo łatwe w klasyfikacji i zwykle nie sprawiają większych problemów klasyfikatorom. Przeprowadzone obserwacje potwierdziły ten fakt. Oba użyte testy statystyczne wykazały, że nie ma istotnych różnić pomiędzy testowanymi konfiguracjami i tylko w jednym przypadku T-test wskazał przewagę konfiguracji T2 nad Meta. W tabeli 5.5 przedstawiona została średnia ilość węzłów i liści w drzewach każdego typu, a także arkusz wyboru klasyfikatorów przez algorytm Meta – drzewa. Identyczne zestawienie zostało sporządzone dla wszystkich zbiorów danych.

	T1	T2	T3	T4	T5	T6	META	SSV
Dokładność:								
minimalna	91,33%	91,33%	91,33%	91,33%	91,33%	92,00%	91,33%	90,00%
średnia	93,07%	93,27%	93,07%	93,07%	93,20%	93,53%	93,00%	94,00%
maksymalna	94,67%	95,33%	95,33%	94,67%	95,33%	96,00%	94,67%	95,33%
Odchylenie std:	1,10%	1,24%	1,18%	1,10%	1,25%	1,22%	1,05%	1,54%
Dokł – Odch. std.	93,57%	94,10%	94,15%	93,57%	94,08%	94,78%	93,61%	93,79%
Dane:	**iris.dat**							

Tabela 5.1: Podstawowe statystyki klasyfikatorów na zbiorze Iris Flowers

Drzewo		T1	T2	T3	T4	T5	T6	Meta	SSV
T1	1		-1,96	0,00	0,00	-1,50	-1,25	1,00	-0,72
	2		0,08	1,00	1,00	0,17	0,24	0,34	0,49
T2	1			1,96	1,96	1,00	-0,63	**2,45**	-1,37
	2			0,08	0,08	0,34	0,54	**0,04**	0,20
T3	1				0,00	-1,50	-1,17	1,00	-1,28
	2				1,00	0,17	0,27	0,34	0,23
T4	1					-1,50	-1,25	1,00	-0,72
	2					0,17	0,24	0,34	0,49
T5	1						-0,81	1,96	-1,18
	2						0,44	0,08	0,27
T6	1							1,50	0,82
	2							0,17	0,44
Meta	1								-1,06
	2								0,32
SSV	1								
	2								
		1. t-Value	2. p-Value		**Sparowany T – test**		Dane:	**iris.dat**	

Tabela 5.2: Porównanie klasyfikatorów z użyciem Sparowanego T-testu

Drzewo		T1	T2	T3	T4	T5	T6	Meta	SSV
T1	1		3,00	0,00	0,00	2,00	2,13	0,33	-1,18
	2		0,08	1,00	1,00	0,16	0,14	0,56	0,12
T2	1			3,00	1,29	1,00	0,67	2,67	-0,94
	2			0,08	0,26	0,32	0,41	0,10	0,17
T3	1				0,00	2,00	1,96	0,20	-0,99
	2				1,00	0,16	0,16	0,65	0,16
T4	1					0,67	2,58	0,33	-1,18
	2					0,41	0,11	0,56	0,12
T5	1						1,09	1,80	-1,04
	2						0,30	0,18	0,15
T6	1							2,91	0,84
	2							0,09	0,20
Meta	1								-0,96
	2								0,17
SSV	1								
	2								
		1.z-Value	2. p-Value		**Test Wilcoxsona**		Dane:	**iris.dat**	

Tabela 5.3: Porównanie klasyfikatorów z użyciem testu Wilcoxona

Drzewo	Test	T1	T2	T3	T4	T5	T6	Meta	SSV
T1	Sparow any T-test		0	0	0	0	0	0	0
	Wilcoxon		0	0	0	0	0	0	0
T2	Sparow any T-test	0		0	0	0	0	**1**	0
	Wilcoxon	0		0	0	0	0	0	0
T3	Sparow any T-test	0	0		0	0	0	0	0
	Wilcoxon	0	0		0	0	0	0	0
T4	Sparow any T-test	0	0			0	0	0	0
	Wilcoxon	0	0			0	0	0	0
T5	Sparow any T-test	0	0	0	0		0	0	0
	Wilcoxon	0	0	0	0		0	0	0
T6	Sparow any T-test	0	0	0	0	0		0	0
	Wilcoxon	0	0	0	0	0		0	0
Meta	Sparow any T-test	0	**-1**	0	0	0	0		0
	Wilcoxon	0	0	0	0	0	0		0
SSV	Sparow any T-test	0	0	0	0	0	0	0	
	Wilcoxon	0	0	0	0	0	0	0	
Próg istotności (p – value):		**0,05**			Dane:	**iris.dat**			

Tabela 5.4: Podsumowanie porównania klasyfikatorów na zbiorze Iris Flowers

	T1	T2	T3	T4	T5	T6
Średnia ilość węzłów w drzewie	7	6	7	7	5	12
Użycie klasyfikatorów przez algorytm meta (10CV)	2	0	2	2	2	2
Dane:	**iris.dat**					

Tabela 5.5: Średnia ilość węzłów w drzewach i użycie klasyfikatorów przez Meta - drzewo

Badania wykorzystujące zbiór *Mushroom* pokazały, jak ważnym elementem algorytmów budowy drzew decyzyjnych jest proces generowania zbioru testów węzła, używanych następnie w procesie rekurencyjnego dzielenia zbioru danych. Z łatwością możemy zaobserwować jak w tym przypadku klasyfikatory generujące podziały (dla cech o wartościach dyskretnych) na tyle części, ile wartości dana cecha przyjmuje, zyskują przewagę nad tymi, które używają podziałów binarnych. Statystycznie istotna różnica została stwierdzona przez oba zastosowane testy. Ten eksperyment potwierdził także przydatność Meta – Drzewa. Klasyfikator Meta nie był statystycznie gorszy od żadnego innego użytego w testach, a od czterech (z sześciu testowanych) okazał się skuteczniejszy. Meta -algorytm sam dobrał najbardziej optymalne elementy ostatecznego klasyfikatora, odrzucając te, które przyczyniły się do słabych wyników konfiguracji T1, T3 i T6.

	T1	T2	T3	T4	T5	T6	META
Dokładność:							
minimalna	93,33%	99,36%	93,39%	99,91%	98,52%	93,65%	99,93%
średnia	93,69%	99,88%	93,66%	99,97%	98,52%	93,79%	99,96%
maksymalna	93,88%	**99,99%**	93,88%	**100,00%**	98,52%	93,88%	**99,99%**
Odchylenie std:	0,20%	0,19%	0,17%	0,03%	**0,00%**	0,07%	0,02%
dokł – odch. std.	93,69%	99,80%	93,71%	**99,97%**	98,52%	93,82%	**99,97%**
Dane:	**mushroom.dat**						

Tabela 5.6: Podstawowe statystyki klasyfikatorów na zbiorze Mushroom

Drzewo		T1	T2	T3	T4	T5	T6	Meta
T1	t-Value		**-64,72**	0,35	**-92,23**	**-77,65**	-1,92	**-100,34**
	p-value		**0,00**	0,74	**0,00**	**0,00**	0,09	**0,00**
T2	t-Value			**70,07**	-1,83	**22,75**	**100,20**	-1,55
	p-value			**0,00**	0,10	**0,00**	**0,00**	0,16
T3	t-Value				**-114,82**	**-90,17**	**-2,41**	**-115,65**
	p-value				**0,00**	**0,00**	**0,04**	**0,00**
T4	t-Value					**152,63**	**264,50**	1,50
	p-value					**0,00**	**0,00**	0,17
T5	t-Value						**229,71**	**-236,53**
	p-value						**0,00**	**0,00**
T6	t-Value							**-303,28**
	p-value							**0,00**
Meta	t-Value							
	p-value							
Test statystyczny:	**Sparowany T - test**				Dane:	**mushroom.dat**		

Tabela 5.7: Porównanie klasyfikatorów z użyciem Sparowanego T-testu

Drzewo		T1	T2	T3	T4	T5	T6	Meta
T1	z-Value		**-2,83**	1,55	**-2,83**	**-2,83**	1,45	**-2,83**
	p-value		**0,00**	0,06	**0,00**	**0,00**	0,07	**0,00**
T2	z-Value			**2,78**	**-2,22**	**2,78**	**2,78**	-1,55
	p-value			**0,00**	**0,01**	**0,00**	**0,00**	0,06
T3	z-Value				**-2,83**	**-2,83**	0,94	**-2,83**
	p-value				**0,00**	**0,00**	0,17	**0,00**
T4	z-Value					**2,78**	**2,78**	1,04
	p-value					**0,00**	**0,00**	0,15
T5	z-Value						**2,78**	**-2,83**
	p-value						**0,00**	**0,00**
T6	z-Value							**-2,83**
	p-value							**0,00**
Meta	z-Value							
	p-value							
Test statystyczny:	**Test Wilcoxsona**				Dane:	**mushroom.dat**		

Tabela 5.8: Porównanie klasyfikatorów z użyciem testu Wilcoxona

Drzewo	Test	T1	T2	T3	T4	T5	T6	Meta
T1	Sparowany T-test		-1	0	-1	-1	0	-1
	Wilcoxon		-1	0	-1	-1	0	-1
T2	Sparowany T-test	1		1	0	1	1	0
	Wilcoxon	1		1	-1	1	1	0
T3	Sparowany T-test	0	-1		-1	-1	-1	-1
	Wilcoxon	0	-1		-1	-1	0	-1
T4	Sparowany T-test	1	0	1		1	1	0
	Wilcoxon	1	1	1		1	1	0
T5	Sparowany T-test	1	-1	1	-1		1	-1
	Wilcoxon	1	-1	1	-1		1	-1
T6	Sparowany T-test	0	-1	1	-1	-1		-1
	Wilcoxon	0	-1	0	-1	-1		-1
Meta	Sparowany T-test	1	0	1	0	1	1	
	Wilcoxon	1	0	1	0	1	1	
Próg istotności (p – value):		**0,05**			Dane:	**mushroom.dat**		

Tabela 5.9: Podsumowanie porównania klasyfikatorów na zbiorze Mushroom

	T1	T2	T3	T4	T5	T6
Średnia ilość węzłów w drzewie	30	25	31	29	10	31
Użycie klasyfikatorów przez algorytm meta (10CV)	0	5	0	5	0	0
Dane:	**mushroom.dat**					

Tabela 5.10: Średnia ilość węzłów w drzewach i użycie klasyfikatorów przez Meta - drzewo

Analiza wyników uzyskanych na danych ze zbioru *Breast Cancer Wisconsin* ewidentnie wskazuje, że zrezygnowanie z przycinania połączone z podziałem danych na liczne podzbiory względem wartości cech oraz stosunkowo słabe kryterium stopu (czystość węzła na poziomie 80%) nie daje zadowalających rezultatów. W przypadku tego zbioru danych testy wykazały również niekwestionowaną wyższość kryterium separowalności nad klasycznymi kryteriami wywodzącymi się z teorii informacji. Meta klasyfikator po raz kolejny udowodnił swoją przydatność osiągając wyniki statystycznie nie odbiegające od pozostałych dobrych wyników, ustępując jedynie algorytmowi SSV.

	T1	T2	T3	T4	T5	T6	META	SSV
Dokładność:								
minimalna	93,13%	93,28%	93,41%	93,56%	91,56%	92,71%	93,13%	94,28%
średnia	94,02%	94,25%	94,09%	94,06%	92,06%	93,91%	94,02%	95,12%
maksymalna	94,71%	96,14%	95,14%	94,57%	92,71%	95,14%	94,71%	95,99%
Odchylenie std:	0,51%	0,84%	0,53%	0,33%	0,31%	0,65%	0,51%	0,58%
Dokł – Odch. std.	94,19%	95,30%	94,60%	94,24%	92,39%	94,49%	94,19%	95,41%
Dane:	**breast-cancer-wisconsin.dat**							

Tabela 5.11: Podstawowe statystyki klasyfikatorów na zbiorze Breast Cancer Wisconsin

Drzewo		T1	T2	T3	T4	T5	T6	Meta	SSV
T1	1		-0,88	-0,46	-0,34	**9,58**	0,57	1,04	**-2,73**
	2		0,40	0,65	0,74	**0,00**	0,58	0,32	**0,02**
T2	1			1,06	0,81	**8,14**	1,01	1,17	-3,75
	2			0,32	0,44	**0,00**	0,34	0,27	0,00
T3	1				0,22	**12,83**	0,69	0,97	**-4,05**
	2				0,83	**0,00**	0,51	0,36	**0,00**
T4	1					**13,54**	0,87	1,29	**-5,76**
	2					**0,00**	0,41	0,23	**0,00**
T5	1						**-6,64**	**-7,13**	**-17,25**
	2						**0,00**	**0,00**	**0,00**
T6	1							0,37	**-2,83**
	2							0,72	**0,02**
Meta	1								**-5,39**
	2								**0,00**
SSV	1								
	2								
1. t-Value		2. p-Value		**Sparowany T – test**			Dane:	**breast-cancer-wisconsin.dat**	

Tabela 5.12: Porównanie klasyfikatorów z użyciem Sparowanego T-testu

Drzewo		T1	T2	T3	T4	T5	T6	Meta	SSV
T1	1		0,13	0,43	0,03	**2,78**	**1,66**	0,43	-1,40
	2		0,45	0,33	0,49	**0,00**	**0,05**	0,33	0,08
T2	1			-0,18	0,64	**2,78**	1,25	0,54	-1,61
	2			0,43	0,26	**0,00**	0,11	0,30	0,05
T3	1				-0,08	**2,78**	1,25	1,45	**-1,91**
	2				0,47	**0,00**	0,11	0,07	**0,03**
T4	1					**2,78**	1,25	1,45	**-2,83**
	2					**0,00**	0,11	0,07	**0,00**
T5	1						**-2,83**	**-2,83**	**-2,83**
	2						**0,00**	**0,00**	**0,00**
T6	1							0,03	**-2,42**
	2							0,49	**0,01**
Meta	1								**-2,83**
	2								**0,00**
SSV	1								
	2								
		1.z-Value	2. p-Value		**Test Wilcoxsona**		Dane:	**breast-cancer-wisconsin.dat**	

Tabela 5.13: Porównanie klasyfikatorów z użyciem testu Wilcoxona

Drzewo	Test	T1	T2	T3	T4	T5	T6	Meta	SSV
T1	Sparow any T-test		0	0	0	**1**	0	0	**-1**
	Wilcoxon		0	0	0	0	**1**	0	0
T2	Sparow any T-test	0		0	0	**1**	0	0	0
	Wilcoxon	0		0	0	**1**	0	0	0
T3	Sparow any T-test	0	0		0	**1**	0	0	**-1**
	Wilcoxon	0	0		0	**1**	0	0	**-1**
T4	Sparow any T-test	0	0	0		**1**	0	0	**-1**
	Wilcoxon	0	0	0		**1**	0	0	**-1**
T5	Sparow any T-test	**-1**	**-1**	**-1**	**-1**		**-1**	**-1**	**-1**
	Wilcoxon	**-1**	**-1**	**-1**	**-1**		**-1**	**-1**	**-1**
T6	Sparow any T-test	0	0	0	0	**1**		0	**-1**
	Wilcoxon	-1	0	0	0	**1**		0	**-1**
Meta	Sparow any T-test	0	0	0	0	**1**	0		**-1**
	Wilcoxon	0	0	0	0	**1**	0		**-1**
SSV	Sparow any T-test	**1**	0	**1**	**1**	**1**	**1**	**1**	
	Wilcoxon	0	0	**1**	**1**	**1**	**1**	**1**	
Próg istotności (p – value):		**0,05**				Dane:		**breast-cancer-wisconsin.dat**	

Tabela 5.14: Podsumowanie porównania klasyfikatorów na zbiorze Breast Cancer Wisconsin

	T1	T2	T3	T4	T5	T6
Średnia ilość węzłów w drzewie	27	29	25	25	3	51
Użycie klasyfikatorów przez algorytm meta (10CV)	1	2	5	1	0	1
Dane:	**breast-cancer-wisconsin.dat**					

Tabela 5.15: Średnia ilość węzłów w drzewach i użycie klasyfikatorów przez Meta - drzewo

Testy przeprowadzone na zbiorze *Thyroid* pokazały, że zdecydowanie gorzej radzą sobie te klasyfikatory, w konfiguracji których brakuje przycinania z opcją automatycznego doboru parametru. Warto zaznaczyć, że w/w zbiór jest stosunkowo duży (3772 przypadki), więc brak odpowiedniej jakości przycinania daje wyraźne negatywne efekty. Najlepszym klasyfikatorem okazał się SSV, najgorszymi T5 i T6. Między pozostałymi klasyfikatorami nie zaobserwowano istotnych statystycznie różnic.

	T1	T2	T3	T4	T5	T6	META
Dokładność:							
minimalna	97,43%	97,75%	97,59%	97,43%	92,47%	96,90%	97,53%
średnia	97,82%	97,89%	97,84%	97,83%	92,47%	97,42%	97,85%
maksymalna	98,12%	98,14%	98,12%	98,04%	**92,47%**	97,69%	98,06%
Odchylenie std:	0,21%	0,12%	0,17%	0,18%	0,00%	0,22%	0,18%
Dokł – Odch. std.	97,91%	98,02%	97,95%	97,85%	92,47%	97,47%	97,89%
Dane:	**thyroid.dat**						

Tabela 5.16: Podstawowe statystyki klasyfikatorów na zbiorze Thyroid

Drzewo		T1	T2	T3	T4	T5	T6	Meta	SSV
T1	1		-0,92	-0,18	-0,14	**82,10**	**4,11**	-0,24	**-25,03**
	2		0,38	0,86	0,89	**0,00**	**0,00**	0,82	**0,00**
T2	1			1,18	1,01	**140,43**	**5,73**	0,79	**-31,36**
	2			0,27	0,34	**0,00**	**0,00**	0,45	**0,00**
T3	1				0,12	**99,35**	**3,76**	-0,16	**-26,50**
	2				0,91	**0,00**	**0,00**	0,88	**0,00**
T4	1					**91,62**	**4,20**	-0,19	**-25,86**
	2					**0,00**	**0,00**	0,85	**0,00**
T5	1						**-71,32**	**-95,13**	**-288,19**
	2						**0,00**	**0,00**	**0,00**
T6	1							**-4,94**	**-26,54**
	2							**0,00**	**0,00**
Meta	1								**-25,10**
	2								**0,00**
SSV	1								
	2								
		1. t-Value	2. p-Value		**Sparowany T – test**		Dane:	**thyroid.dat**	

Tabela 5.17: Porównanie klasyfikatorów z użyciem Sparowanego T-testu

Drzewo		T1	T2	T3	T4	T5	T6	Meta	SSV
T1	1		-1,10	-0,08	-0,28	**2,78**	**2,17**	-0,28	**-2,83**
	2		0,14	0,47	0,39	**0,00**	**0,02**	0,39	**0,00**
T2	1			1,35	-0,38	**2,78**	**2,78**	0,43	**-2,83**
	2			0,09	0,35	**0,00**	**0,00**	0,33	**0,00**
T3	1				0,03	**2,78**	**2,78**	-0,99	**-2,83**
	2				0,49	**0,00**	**0,00**	0,16	**0,00**
T4	1					**2,78**	**2,17**	-0,48	**-2,83**
	2					**0,00**	**0,02**	0,31	**0,00**
T5	1						**-2,83**	**-2,83**	**-2,83**
	2						**0,00**	**0,00**	**0,00**
T6	1							**-2,52**	**-2,83**
	2							**0,01**	**0,00**
Meta	1								**-2,83**
	2								**0,00**
SSV	1								
	2								
		1.z-Value	2. p-Value		**Test Wilcoxsona**		Dane:	**thyroid.dat**	

Tabela 5.18: Porównanie klasyfikatorów z użyciem testu Wilcoxona

Drzewo	Test	T1	T2	T3	T4	T5	T6	Meta	SSV
T1	Sparowany T-test		0	0	0	1	1	0	-1
	Wilcoxon		0	0	0	1	1	0	-1
T2	Sparowany T-test	0		0	0	1	1	0	-1
	Wilcoxon	0		0	0	1	1	0	-1
T3	Sparowany T-test	0	0		0	1	1	0	-1
	Wilcoxon	0	0		0	1	1	0	-1
T4	Sparowany T-test	0	0	0		1	1	0	-1
	Wilcoxon	0	0	0		1	1	0	-1
T5	Sparowany T-test	-1	-1	-1	-1		-1	-1	-1
	Wilcoxon	-1	-1	-1	-1		-1	-1	-1
T6	Sparowany T-test	-1	-1	-1	-1	1		-1	-1
	Wilcoxon	-1	-1	-1	-1	1		-1	-1
Meta	Sparowany T-test	0	0	0	0	1	1		-1
	Wilcoxon	0	0	0	0	1	1		-1
SSV	Sparowany T-test	1	1	1	1	1	1	1	
	Wilcoxon	1	1	1	1	1	1	1	
Próg istotności (p – value):		**0,05**			Dane:	**thyroid.dat**			

Tabela 5.19: Podsumowanie porównania klasyfikatorów na zbiorze Thyroid

	T1	T2	T3	T4	T5	T6
Średnia ilość węzłów w drzewie	9	6	6	14	2	95
Użycie klasyfikatorów przez algorytm meta (10CV)	4	3	3	0	0	0
Dane:	**thyroid.dat**					

Tabela 5.20: Średnia ilość węzłów w drzewach i użycie klasyfikatorów przez Meta - drzewo

Aby łatwiej dokonać oceny poszczególnych klasyfikatorów, stworzone zostało zestawienie zbiorcze (Tabela 5.21). Prezentowane dane pokazują, ile razy dany klasyfikator okazywał się lepszy, a ile razy gorszy, w porównaniu z konkurencją. Bardzo łatwo daje się zauważyć, że najlepiej poradziły sobie drzewa o konfiguracjach T2 i T4, tylko nieznacznie gorsza okazała się konfiguracja Meta. Przewaga tych drzew wynika z zastosowania testów węzła, które dla cech nieuporządkowanych dzielą zbiór na tyle elementów, ile wartości przyjmuje dane cecha (*MultiSplitTestGenerator*). Przypadek najgorszego klasyfikatora (T5) pokazuje, jak ważnym elementem algorytmu budowy drzewa decyzyjnego jest odpowiednio dobrane przycinanie. Zastąpienie tylko jednego elementu konfiguracji T2 (przycinania z automatycznym doborem parametrów), niezbyt wymagającym kryterium stopu (czystość

węzła na poziomie 80%) spowodował drastyczny spadek jakości klasyfikacji. Zgodnie z przewidywaniem nie zaobserwowano natomiast istotnych różnic pomiędzy kryterium Giniego *(GiniNTS)* a względnym zyskiem informacyjnym (*InfoGainRatioNTS)* podczas oceny podziałów.

	Test	T1	T2	T3	T4	T5	T6	Meta
Najlepsze	Sparowany T-test	3	**8**	3	**7**	**3**	3	**7**
	Wilcoxon	3	**7**	3	**8**	**3**	2	**7**
Najgorsze	Sparowany T-test	-4	**0**	-5	**0**	**-15**	-9	**-1**
	Wilcoxon	-4	**-1**	-4	**0**	**-15**	-9	**0**
Próg istotności (p – value):		**0,05**						

Tabela 5.21: Zbiorcze podsumowanie wyników

5.5 Podsumowanie

Jednym z celów przeprowadzonych doświadczeń było wykazanie, że istotny wpływ na jakość klasyfikacji ma odpowiedni dobór elementów i parametrów klasyfikatora. W toku eksperymentów zastosowano Meta – Drzewo, a jedną ze spodziewanych korzyści płynących z jego użycia było zniwelowanie różnic wynikających z różnorodności charakteru danych i nieoptymalną konfiguracją klasyfikatora. Założenie to zostało potwierdzone podczas przeprowadzonych eksperymentów. W przypadku zbiorów danych, w których widoczna jest spora różnica w jakości klasyfikacji, zastosowanie Meta - Drzewa pozwala na osiągniecie rezultatów bliskich optymalnym. Przeprowadzone z użyciem zbioru *mushroom* jednoznacznie pokazują, że jest on bardzo wrażliwy na kryterium generowania podziałów użytego w procesie budowy drzewa decyzyjnego. Klasyfikatory korzystające z podziałów binarnych (generowanych przez klasę *BinaryTestGenerator*) osiągały znacznie gorsze rezultaty od tych, które dzieliły zbiory danych na wiele części na każdym etapie budowy drzewa. Meta klasyfikator potrafił zidentyfikować tą prawidłowość, dokonał klasyfikacji z użyciem korzystniejszych mechanizmów, osiągając wyniki praktycznie nie ustępujące rozwiązaniu optymalnemu. Można stwierdzić, że wykorzystanie nawet tak prostej formy Meta – Drzewa, pozwala na wyeliminowanie niepożądanych efektów niedopasowania elementów algorytmu budowy drzewa do danych testowych. Zgodnie z ideą meta – uczenia umożliwia on dobór klasyfikatorów bez udziału użytkownika. Dalsze prace nad tym mechanizmem dają szanse na

poprawienie osiąganej przez niego dokładności, zmniejszenie odchylenia standardowego oraz znacznie skrócenie czasu budowy drzewa.

Literatura

[1] P. Cichosz. Systemy uczące się. Wydawnictwa Naukowo – Techniczne, Warszawa 2000.

[2] W. Duch. Computional Intelligence: Decision tress. SCE, NTU, Singapore 2005.

[3] W. Duch. Computional Intelligence: Methods and Applications. Pruning of decision tress. SCE, NTU, Singapore 2005.

[4] W. Duch. Inteligencja Obliczeniowa. Drzewa Decyzji. Wykład 23. Uniwersytet Mikołaja Kopernika, Toruń 2004.

[5] R. Duda, P. Hart, and D. Stork, "Chapter 8. Non-metric Methods," in *Pattern Classification*, 2nd ed: John Wiley & Sons, New Jersey 2000.

[6] K. Grąbczewski, N. Jankowski. Meta-Learning Architecture for Knowledge Representation and Management in Computational Intelligenc. Department of Informatics, Nicolaus Copernicus University, Toruń 2007.

[7] K. Grąbczewski, N. Jankowski. Versatile and Efficient Meta-Learning Architecture: Knowledge Representation and Management in Computational Intelligence. Department of Informatics, Nicolaus Copernicus University, Toruń 2007.

[8] K. Grąbczewski. Zastosowanie kryterium separowalności do generowania reguł klasyfikacji na podstawie baz danych. Instytut Badań Systemowych. Polska Akademia Nauk,

Warszawa 2003.

[9] D. Hand, H. Mannila, P. Smyth. Eksploracja danych. Wydawnictwa Naukowo – Techniczne, Warszawa 2005.

[10] Kolluru Venkata Sreerama Murthy. On Growing Better Decision Trees from Data, Baltimore, Maryland 1995.

[11] Danie T. Larose. Discovery Knowledge in Data. Chapter 6. “Decision trees”. John Wiley & Sons. Hoboken, New Jersey 2005.

[12] R. J. Lewis. An Introduction to Classification and Regression Tree (CART) Analysis. Department of Emergency Medicine, Harbor-UCLA Medical Center Torrance, California 2000.

[13] G. Meyer-Brotz, J.Schurmann. Methoden der automatischen Zeichenerkennung. Akademie-Verlag, Berlin1970.

[14] D. Michie, D.J. Spiegelhalter, C.C. Taylor. Machine Learning, Neural and Statistical Classification. Ellis Horwood Limited, New York 1994.

[15] Microsoft Corporation. Microsoft Visual Studio 2005. WEB: http://msdn.microsoft.com/vstudio.

[16] W. Muller, F. Wysotzki. Automatic construction of decision trees for classification. Fraunhofer Institut IITB, Berlin 1994.

[17] W. Muller, F. Wysotzki. The Decision Tree Algorithm CAL5 based on a Statistical Approach to Its Splitting Algorithm. Machine Learning and Statistics: The Interface, pp. 45-65, Wiley, New Jersey 1997.

[18] M. Pennington. C4.5 Rule Preceded by an Artificial Neural Network Ensemble for

Medical Diagnosis, Sheffield 2003.

[19] J.R. Quinlan. Induction of decision trees. Mach. Learn, 1986.

[20] Sparx Systems. Enterprise Architect 7.0. WEB: http://www.sparxsystems.com.au/

[21] A. Stanisz. Podstawy statystyki dla prowadzących badania naukowe Odcinek 12: Testy nieparametryczne. Medycyna Praktyczna 1999/10.

[22] StatSoft (2006). Elektroniczny Podręcznik Statystyki PL, Krakow, WEB: http://www.statsoft.pl/textbook/stathome.html.

[23] University of California, Irvine. UCI Machine Learning Repository, WEB: http://archive.ics.uci.edu/ml/

[24] Wei-Yin Loh, Yu-Shan Shih. Split Selection Methods for Classification Trees. Published in Statistica Sinica Vol 7, pp. 815-840, 1997.

[25] N. Wirth. Algorytmy + struktury danych = programy. Wydawnictwa Naukowo – Techniczne, Warszawa 2001.