Lecture Notes in Computer Science 7665, 390-397.
Springer, Heidelberg, 2012

Recursive Similarity-Based Algorithm for Deep Learning

Tomasz Maszczyk[1] and Włodzisław Duch[1,2]

[1] Department of Informatics, Nicolaus Copernicus University
Grudziądzka 5, 87-100 Toruń, Poland
[2] School of Computer Engineering, Nanyang Technological University, Singapore
{tmaszczyk,wduch}@is.umk.pl
http://www.is.umk.pl

Abstract. Recursive Similarity-Based Learning algorithm (RSBL) follows the deep learning idea, exploiting similarity-based methodology to recursively generate new features. Each transformation layer is generated separately, using as inputs information from all previous layers, and as new features similarity to the k nearest neighbors scaled using Gaussian kernels. In the feature space created in this way results of various types of classifiers, including linear discrimination and distance-based methods, are significantly improved. As an illustrative example a few non-trivial benchmark datasets from the UCI Machine Learning Repository are analyzed.

Key words: similarity-based learning, deep networks, machine learning, k nearest neighbors

1 Introduction

Classification is one of the most important area of machine learning. Similarity-based methods (SBL, [1, 2]), including many variants of the k-nearest neighbor algorithms, belong to the most popular and simplest methods used for this purpose. Although such methods have many advantages, including an easy handling of unlimited number of classes and stability of solutions against small perturbations of data, their applications are limited, because their computation time scales like $O(n^2)$ with the number of reference samples n. For large databases, especially in problems requiring real-time decisions, such "lazy approaches" relaying more on calculations performed at the time of actual classification rather than at the time of training are too slow. Training of all similarity-based methods, including kernel-based SVM approaches, also suffers from the same quadratic scaling problem. Fast methods for finding approximate neighbors can reduce this time to roughly $O(\log n)$ [3].

After decades of development simple predictive machine learning methods seem to have reached their limits. The future belongs to techniques that automatically compose many transformations, as it is done in meta-learning based on search in the model space [4–6], learning based on generation of novel features [7, 8], and deep learning [9] approaches. The recursive SBL (RSBL) approach presented here is inspired by recent successes of the deep learning techniques. Kernel-based approaches make only one step, replacing original features with similarity-based features and performing linear discrimination in this space. Deep learning in neural networks is based on learning in new feature spaces created by adding many network layers, in essence performing recursive transformations. Instead of sequentially performing input/output transformations, RSBL version considered here systematically expands the feature space using information from all previous stages of data transformation. In this paper only transformations based on similarities to the nearest k-samples scaled by Gaussian kernel features are explored, but any other similarity measures may be used in the same way [7, 8]. In essence this connects similarity-based methodology with deep learning techniques, creating higher-order k-nearest neighbors method with kernel features.

In the next two sections a distance-based and deep learning approaches are introduced. Short description of classification algorithms used here is given in the fourth section. RSBL algorithm is introduced in section 5. Illustrative examples for several datasets that require non-trivial analysis [10] are presented in section 6. Conclusions are given in the final section.

2 Similarity-Based Learning

Categorization of new points based on their distance to points in a reference (training) dataset is a simple and effective way of classification. There are many parameters and procedures that can be included in the data models M based on similarity. Such models are optimized to calculate posterior probability $p(C_i|\boldsymbol{x}; M)$ that a vector $\boldsymbol{x}$ belongs to class C_i [1, 2]. Optimization includes the type of distance functions, or the type of kernel $D(\boldsymbol{x}, \boldsymbol{y})$ that should be designed depending on the problem, selection of reference instances, weighting of their influence, and other elements. The most common distance functions include:

- Minkowski's metric $D(\boldsymbol{x}, \boldsymbol{y})^\alpha = \sum_{i=1}^{d} |x_i - y_i|^\alpha$, becoming Euclidean metric for $\alpha = 2$, the city block metric for $\alpha = 1$ and the Chebychev metric for $\alpha = \infty$.
- Mahalanobis distance $D(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(\boldsymbol{x} - \boldsymbol{y})' \mathbf{C}^{-1} (\boldsymbol{x} - \boldsymbol{y})}$ where $\mathbf{C}$ is the covariance matrix, taking into account scaling and rotation of data clusters.
- Cosine distance, equal to the normalized dot product $D(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \cdot \boldsymbol{y} / ||\boldsymbol{x}|| ||\boldsymbol{y}||$.
- Hamming distance is used for binary features $D(\boldsymbol{x}, \boldsymbol{y}) = \#(x_i \neq y_i)/d$.
- Correlation distance is also often used:

$$D(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{d} (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{d} (x_i - \overline{x})^2 \sum_{i=1}^{d} (y_i - \overline{y})^2}} \tag{1}$$

Heterogenous metric functions suitable for nominal data may be defined using conditional probabilities [1, 2], but will not be used in this paper.

3 Deep learning

Learning proceeds by reduction of information, starting from rich information at the input side and after a series of transformations creating an output sufficient for high-level decision, such as an assignment to a specific category. This information compression process can be presented as a network, a flow graph in which each node represents elementary data transformation. Flow graphs have different depth, i.e. the length of the longest path from an input to an output. Popular classification algorithms have low depth indices. For example, SVM algorithms, Radial Basis Function (RBF) networks, or the k-NN algorithms all have depth equal to two: one for the kernel/distance calculation, and one for the linear models producing outputs, or for selection of the nearest neighbors. Kernel SVM algorithms are basically linear discrimination algorithms in the space of kernel features [8], selected using the wide-margin principle. The depth of the Multilayer Perceptron (MLP) neural networks depends on the number of hidden layers and in most cases is rather small, but in deep learning it tends to be large [9].

Despite their low depth most classifiers are universal approximators, i.e. they can represent arbitrary function to a given target accuracy. In his book Bengio [9] shows examples of functions that can be represented in a simple way with deep architecture, but a shallow one may require exponentially large number of nodes in the flow graph, and may be hopelessly difficult to optimize. The most important motivation to introduce deep learning comes from signal processing by the brain. For example, the image processing by the retina, lateral geniculate nuclei and visual cortex is done in many areas, each of which extracts some features from the input, and communicates results to the next level. Each level of this feature hierarchy represents the input at a different level of abstraction, with more abstract features further up in the hierarchy, defined in terms of the lower-level ones. One may argue that this processing is in fact best approximated by a sequence of layers estimating similarity based on the lower-level similarity estimations. People organize ideas and concepts hierarchically, learning first simpler concepts and then composing them to represent more abstract ones. Engineers break-up solutions into multiple levels of abstraction and processing, using the divide-and-conquer approach at many levels. RSBL is inspired by such observations.

4 Classification algorithms

In this section short description of classification algorithms used in our tests is presented. All of them are well known and their detailed description may be found in classic textbooks [11].

4.1 Support Vector Machines (SVM)

Support Vector Machines (SVMs) are currently the most popular method of classification and regression [12]. They require two transformations: first is based on kernels that estimate similarity $K(\boldsymbol{x}; \boldsymbol{x}_i)$ comparing the current vector $\boldsymbol{x}$ to the reference vectors $\boldsymbol{x}_i$ selected from the training set. The second transformation is based on linear discrimination, selecting from the training vectors only those reference vectors (support vectors) that are close to the decision border, with regularization term added to ensure wide-margin solutions. Depending on the choice and optimization of kernel parameters SVM is capable of creating flexible nonlinear data models that, thanks to the optimization of classification margin, offer good generalization. The best solution maximizes the minimum distance between the training vectors $\boldsymbol{x}_i$ and the points $\boldsymbol{x}$ on the decision hyperplane $\boldsymbol{w}$:

$$\max_{\boldsymbol{w},b} \min \|\boldsymbol{x}_i - \boldsymbol{x}\| \ : \ \boldsymbol{w} \cdot \boldsymbol{x} + b = 0, \ i = 1, \ldots, n \tag{2}$$

The $\boldsymbol{w}$ weight vector and the bias b are rescaled in such a way that points closest to the hyperplane $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$ lie on one of the parallel hyperplanes defining the margin $\boldsymbol{w} \cdot \boldsymbol{x} + b = \pm 1$. This leads to the requirement that:

$$\forall_{\boldsymbol{x}_i} \ y_i[\boldsymbol{w} \cdot \boldsymbol{x}_i + b] \geq 1 \tag{3}$$

The width of the margin is then equal to $2/\|w\|$. The SVM algorithm is usually formulated for two classes, labeled by $y_i = \pm 1$, and presented as quadratic optimization, leading to the discriminant function of the form:

$$g(x) = \mathrm{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i \boldsymbol{x} \cdot \boldsymbol{x}_i + b \right) \tag{4}$$

where linear combination coefficients α_i are multiplied by the y_i. The dot product $\boldsymbol{x} \cdot \boldsymbol{x}_i$ is replaced by a kernel function $K(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}')$ where $\phi(\boldsymbol{x})$ represents an implicit transformation of the original vectors to the new feature space. For any $\phi(\boldsymbol{x})$ vector the part orthogonal to the space spanned by $\phi(\boldsymbol{x}_i)$ does not contribute to $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}')$ product, so it is sufficient to express $\phi(\boldsymbol{x})$ and $\boldsymbol{w}$ as a combination of $\phi(\boldsymbol{x}_i)$ vectors. The dimensionality d of the input vectors is frequently lower than the number of training patterns $d < n$, therefore $\phi(\boldsymbol{x})$ usually represents mapping into a high-dimensional space. Cover theorem [11] is frequently invoked to show advantages of increasing the dimension of the feature space. In some problems – for example the microarray data – dimensionality d may be much higher than the number of training patterns n, which is usually very small. In such cases dimensionality reduction helps to decrease noise inherent in some features. The discriminant function in the $\phi()$ space is:

$$g(\boldsymbol{x}) = \mathrm{sgn}\left(\sum_{i=1}^{n} \alpha_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b \right) \tag{5}$$

If the kernel function is linear the $\phi()$ space is simply the original space and the linear SVM discriminant function is based on cosine distances to the reference vectors $\boldsymbol{x}_i$ from the y_i class. The original features $\boldsymbol{x}_j, j = 1..d$ are replaced by new features $z_i(\boldsymbol{x}) = K(\boldsymbol{x}, \boldsymbol{x}_i), i = 1..n$ that evaluate how close (or how similar) the vector is from the reference vectors using cosine metric. Incorporating signs in the coefficient vector $A_i = \alpha_i y_i$ the binary discriminant functions is:

$$g(\boldsymbol{x}) = \mathrm{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i z_i(\boldsymbol{x}) + b \right) = \mathrm{sgn}\left(\boldsymbol{A} \cdot \boldsymbol{z}(\boldsymbol{x})) + b \right) \tag{6}$$

With the proper choice of non-zero α coefficients this function projects vectors in the kernel space on a line defined by $\boldsymbol{A}$ direction, with b defining the class boundary. In non-separable case instead of using cosine distance measures it is better to use localized similarity measures, for example scaling the distance with Gaussian kernel $K_G(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\beta\|\boldsymbol{x} - \boldsymbol{x}'\|^2)$, contributing to the stability of the SVM solutions.

4.2 k-Nearest Neighbours (kNN)

This is a one of the simplest classification algorithms used in patter recognition. The k-nearest neighbors algorithms classify new objects assigning them to the most common class among the k nearest neighbors (k is typically a small positive integer). If k=1, then the object is simply assigned to the class of its nearest neighbor. Such version of kNN are often called 1NN (one nearest neighbor). The accuracy of k-nearest neighbor classification depends significantly both on the k value (which can be easy optimized using crossvalidation), and the metric used to compute distances between different examples. For continuous variables Euclidean or cosine distance is usually taken as the metric. For nominal features other measures, such as the Hamming distance or probability-dependent metrics may be used.

5 Recursive Similarity-Based Learning (RSBL)

Deep learning methodology combined with distance-based learning and Gaussian kernel features can be seen as recursive supervised algorithm to create new features, and hence used to provide optimal feature space for any classification method. Implementation of RSBL used in this paper is based on Euclidean distance and Gaussian kernel features with fixed σ=0.1, providing new feature spaces at each depth level. Classification is done either by linear SVM with fixed C=2^5, or the 1NN algorithm. The Algorithm sketched below presents steps of the RSBL; in each case parameters $k_{\max} = 20$ and $\alpha = 5$ were used.

Algorithm 1 Recursive similarity-based learning

```
Require: Fix the values of internal parameters: $k_{\max}$, maximum depth $\alpha$, and $\sigma$ (dispersion).
 1: Standardize the dataset, $n$ vectors, $d$ features.
 2: Set the initial space $\mathcal{H}^{(0)}$ using input features $x_{ij}$, $i = 1..n$ vectors and $j = 1..d$ features.
 3: Set the current number of features $d(0) = d$.
 4: for $m = 1$ to $\alpha$ do
 5:    for $k = 1$ to $k_{\max}$ do
 6:       For every training vector $\boldsymbol{x}_i$ find $k$ nearest neighbors $\boldsymbol{x}_{j,i}$ in the $\mathcal{H}^{(m-1)}$ space.
 7:       Create $nk$ new kernel features $z_{j,i}(\boldsymbol{x}) = K(\boldsymbol{x}, \boldsymbol{x}_{j,i}), j = 1..k; i = 1..n$ for all vectors
          using kernel functions as new features.
 8:       Add new $nk$ features to the $\mathcal{H}^{(m-1)}$ space, creating temporary $\mathcal{H}^{(m,k)}$ space.
 9:       Estimate error $E(m,k)$ in the $\mathcal{H}^{(m,k)}$ space on the training or validation set.
10:    end for
11:    Choose $k'$ that minimizes $E(m,k')$ error and retain $\mathcal{H}^{(m,k')}$ space as the new $\mathcal{H}^{(m)}$
       space.
12: end for
13: Build the final model in the enhanced feature space $\mathcal{H}^{(\alpha)}$.
14: Classify test data mapped into the enhanced space.
```

In essence the RSBL algorithm at each level of depth transforms the actual feature space into the extended feature space $\mathcal{H}^{(m)}$, discovering useful information by creating new redundant features. Note that the initial space covers d original features x_j that are available at each depth, preserving useful information that kernel SVM may discard. The final analysis in the $\mathcal{H}^{(\alpha)}$ space (and optimization of parameters at each level of RSBL algorithm, including feature selection) may be done by various machine learning methods. Once useful information is extracted many classification methods may benefit from it. The emphasis is on generation of new features using deep-learning methodology rather than optimization of learning.

In this paper only the simplest models, 1NN and linear SVM with fixed C=2^5, are used for illustration. RSBL may be presented as a constructive algorithm, with new layers representing transformations and procedures to extract and add to the overall pool more features, and a final layer analyzing the image of data in the enhanced feature space.

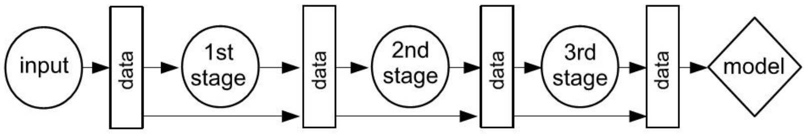

Fig. 1. RSBL method presented in graphical form for depth equal three.

6 Illustrative examples

Many sophisticated machine learning methods are introduced every year and tested on relatively trivial benchmark problems from the UCI Machine Learning Repository [13]. Most of these problems are relatively easy: simple and fast algorithms with $O(nd)$ complexity give results that are not statistically significantly worse than those obtained by the best known algorithms. Some benchmark problems are not trivial, they require complicated decision borders and may only be handled using sophisticated techniques. To distinguish dataset that should be regarded as trivial from more difficult cases simple methods with $O(nd)$ complexity have been compared with the optimized Gaussian SVM results [10].

New methods should improve results of simple low-complexity machine learning methods in non-trivial cases. Below RSBL results for a few non-trivial dataset are presented, i.e. data for which result obtained with low complexity methods are significantly worse than those obtained by kernel SVM. These datasets obtained from the UCI repository [13], are summarized in Tab. 1. In experiments 10-fold crossvalidation tests have been repeated 10 times, and the average results are collected in Tables 2-3, with accuracies and standard deviations given for each dataset.

The ORG column gives results of SVM or 1NN algorithm using the original data. RSBL(1) – RSBL(5) columns presents results in enhanced spaces at depth 1 to 5. In all cases RSBL combined with linear SVM (fixed parameters) gives results that are

Table 1. Summary of datasets used in experiments.

Dataset	#Vectors	#Features	#Classes
ionosphere	351	34	2
monks-problems-1	556	6	2
monks-problems-2	601	6	2
parkinsons	195	22	2
sonar	208	60	2

Table 2. 10 x 10 crossvalidation accuracy and standard deviation for RSBL combined with SVM. Additionally SVM with optimized Gaussian kernels (SVMG) results are presented for comparison.

Dataset	Method						
	ORG	RSBL(1)	RSBL(2)	RSBL(3)	RSBL(4)	RSBL(5)	SVMG
ionosphere	88.2±6.4	92.3±3.8	94.0±3.9	94.0±3.9	94.0±3.9	94.0±3.9	94.6±3.7
monks-problems-1	74.6±4.6	100±0.0	100±0.0	100±0.0	100±0.0	100±0.0	99.8±0.6
monks-problems-2	65.7±0.6	79.6±3.1	84.9±4.1	85.7±4.2	85.7±4.2	85.7±4.2	84.9±4.9
parkinsons	88.7±7.8	89.3±5.4	93.3±4.9	91.3±6.0	89.2±5.1	87.7±5.4	93.2±5.6
sonar	74.9±9.5	82.2±7.9	85.1±4.6	86.6±7.0	87.4±7.7	87.9±7.3	86.4±7.6

Table 3. 10 x 10 crossvalidation accuracy and standard deviation for RSBL combined with 1NN.

Dataset	Method					
	ORG	RSBL(1)	RSBL(2)	RSBL(3)	RSBL(4)	RSBL(5)
ionosphere	87.1±5.2	87.4±4.8	87.8±4.9	87.8±4.9	87.8±4.9	87.8±4.9
monks-problems-1	100±0.0	99.9±0.1	99.4±1.2	99.3±1.2	99.4±1.1	99.4±1.0
monks-problems-2	68.8±6.2	69.2±8.7	71.6±6.2	71.6±6.2	71.6±6.2	71.8±6.2
parkinsons	93.8±5.4	92.8±6.6	91.7±6.1	91.7±6.1	91.7±6.1	91.7±6.1
sonar	85.0±5.8	85.5±6.8	86.0±6.6	87.9±6.5	87.9±6.5	87.9±6.5

comparable to SVM with optimized Gaussian kernels. Additionally, increasing levels of depth provides an increase of classification accuracy (except for the *parkinsons* dataset).

The linear SVM results obtained in the RSBL enhanced feature space are almost always improved, although for this data improvement over RSBL(2) are not significant. Results of the 1NN do not improve in the enhanced space.

7 Conclusions

The most important goal of computational intelligence is to create methods that can automatically discover the best models for a given data. There is no hope that a single method will always be the best [11], therefore such techniques like deep learning, meta-learning or feature construction methodology should be used.

RSBL algorithm introduced in this paper is focused on hierarchical generation of new distance-based and kernel-based features rather than improvement in optimization and classification algorithms. Finding interesting views on the data by systematic addition of novel features is very important because combination of such transformation-based systems should bring us significantly closer to the practical applications that automatically create the best data models for any data. Expanded feature space may benefit not only from random projections, but also from the nearest neighbor methods.

Results on several non-trivial benchmark problems shows that RSBL creates explicitly feature spaces in which linear methods reach results that are at least as good as optimized SVM with Gaussian kernels. Further improvements to the RSBL algorithm will include the use of different distance measures, fast approximate neighbors, feature selection and global optimization of the whole procedure. Applications to more challenging datasets and to the on-line learning of non-stationary data will also be considered.

Acknowledgment This work was supported by the Nicolaus Copernicus University under research grant for young scientists no. 1141-F, and by the Polish Ministry of Education and Science through Grant no. N516 500539.

References

1. Duch, W.: Similarity based methods: a general framework for classification, approximation and association. Control and Cybernetics **29** (2000) 937–968
2. Duch, W., Adamczak, R., Diercksen, G.: Classification, association and pattern completion using neural similarity based methods. Applied Mathematics and Computer Science **10** (2000) 101–120
3. Arya, S., Malamatos, T., Mount, D.: Space-time tradeoffs for approximate nearest neighbor searching. Journal of the ACM **57** (2010) 1–54
4. Duch, W., Grudziński, K.: Meta-learning via search combined with parameter optimization. In Rutkowski, L., Kacprzyk, J., eds.: Advances in Soft Computing. Physica Verlag, Springer, New York (2002) 13–22
5. Maszczyk, T., Grochowski, M., Duch, W.: Discovering Data Structures using Meta-learning, Visualization and Constructive Neural Networks. In: Meta-learning in Computational Intelligence. Volume 262 of Studies in Computational Intelligence, Advances in Machine Learning II. Springer (2010) 467–484
6. Jankowski, N., Duch, W., Grąbczewski, K., eds.: Meta-learning in Computational Intelligence. Volume 358 of Studies in Computational Intelligence. Springer (2011)
7. Duch, W., Maszczyk, T.: Universal learning machines. Lecture Notes in Computer Science **5864** (2009) 206–215
8. Maszczyk, T., Duch, W.: Support feature machines: Support vectors are not enough. In: World Congress on Computational Intelligence, IEEE Press (2010) 3852–3859
9. Bengio, Y.: Learning deep architectures for AI. Foundations and Trends in Machine Learning **2** (2009) 1–127
10. Duch, W., Maszczyk, T., Jankowski, N.: Make it cheap: learning with o(nd) complexity. In: IEEE World Congress on Computational Intelligence, Brisbane, Australia. (2012) 132–135
11. Duda, R.O., Hart, P.E., Stork, D.: Patter Classification. J. Wiley & Sons, New York (2001)
12. Schölkopf, B., Smola, A.: Learning with Kernels. Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge, MA (2001)
13. Asuncion, A., Newman, D.: UCI machine learning repository. http://www.ics.uci.edu/∼mlearn/MLRepository.html (2007)