sciendo

Int. J. Appl. Math. Comput. Sci., 2018, Vol. 28, No. 4, 719–733
DOI: 10.2478/amcs-2018-0055

amcs

COMPARISON OF PROTOTYPE SELECTION ALGORITHMS USED IN CONSTRUCTION OF NEURAL NETWORKS LEARNED BY SVD

NORBERT JANKOWSKI [a]

[a]Department of Informatics
Nicolaus Copernicus University, ul. Grudziądzka 5/7, 87-100 Toruń, Poland
e-mail: norbert@is.umk.pl

Radial basis function networks (RBFNs) or extreme learning machines (ELMs) can be seen as linear combinations of kernel functions (hidden neurons). Kernels can be constructed in random processes like in ELMs, or the positions of kernels can be initialized by a random subset of training vectors, or kernels can be constructed in a (sub-)learning process (sometimes by k-means, for example). We found that kernels constructed using prototype selection algorithms provide very accurate and stable solutions. What is more, prototype selection algorithms automatically choose not only the placement of prototypes, but also their number. Thanks to this advantage, it is no longer necessary to estimate the number of kernels with time-consuming multiple train-test procedures. The best results of learning can be obtained by pseudo-inverse learning with a singular value decomposition (SVD) algorithm. The article presents a comparison of several prototype selection algorithms co-working with singular value decomposition-based learning. The presented comparison clearly shows that the combination of prototype selection and SVD learning of a neural network is significantly better than a random selection of kernels for the RBFN or the ELM, the support vector machine or the kNN. Moreover, the presented learning scheme requires no parameters except for the width of the Gaussian kernel.

Keywords: radial basis function network, extreme learning machines, kernel methods, prototypes, prototype selection, machine learning, k nearest neighbours.

1. Introduction

This paper is focused on classification problems with a training dataset $\mathcal{D}$, which consists of learning vectors $\mathbf{x}_i$ ($\mathbf{x}_i \in \mathbb{R}^n, i \in [1,\dots,m]$) with the corresponding class labels y_i ($\mathbf{y} = [y_1,\dots,y_m]$).

Whenever we use support vector machines (Vapnik, 1995; Boser *et al.*, 1992), radial basis function networks (Broomhead and Lowe, 1988) or extreme learning machines (Huang *et al.*, 2004; 2006) the composed network has the form of a linear combination of kernels:

$$F(\mathbf{x};\mathbf{w}) = \sum_{j=1}^{l} w_j g_j(\mathbf{x}) + w_0, \tag{1}$$

where w_j are weights and $g_j(\mathbf{x})$ are kernel functions ($j \in [1,l]$), while l defines the number of kernels. The sigmoidal function

$$\sigma(\mathbf{x},\mathbf{b},\theta) = \frac{1}{1+e^{-(\mathbf{x}^T\mathbf{b}-\theta)}} \tag{2}$$

is the original kernel in the ELM, but in the RBFN it is usually the Gaussian function

$$h(\mathbf{x},\mathbf{b},\theta) = e^{-\gamma||\mathbf{x}-\mathbf{w}||^2}, \tag{3}$$

(sometimes also used in the ELM (Huang *et al.*, 2006)).

The learning of an RBFN or an ELM (the estimation of $\mathbf{w}$) can be defined by kernel selection and the minimization of the goal:

$$\begin{aligned} J(\mathbf{w}) &= ||\mathbf{G}\mathbf{w}-\mathbf{y}||^2 \\ &= \sum_{i=1}^{m}\Big(\sum_{j=1}^{l} w_j g_j(\mathbf{x}_i) + w_0 - y_i\Big)^2 \end{aligned} \tag{4}$$

over the kernels. The matrix G is defined as

$$G = \begin{bmatrix} 1 & g_1(\mathbf{x}_1) & \cdots & g_l(\mathbf{x}_1) \\ 1 & g_1(\mathbf{x}_2) & \cdots & g_l(\mathbf{x}_2) \\ \vdots & \vdots & & \vdots \\ 1 & g_1(\mathbf{x}_m) & \cdots & g_l(\mathbf{x}_m) \end{bmatrix}. \tag{5}$$

If we want to minimize the above error function, we can look for the minimum of $J(\mathbf{w})$ by solving $\nabla J(\mathbf{w}) = 0$. After some transformations we obtain

$$\mathbf{w} = (G^T G)^{-1} G^T \mathbf{y} = G^{\dagger} \mathbf{y}. \tag{6}$$

The pseudo-inverse matrix $G^{\dagger}$ can be efficiently computed by the singular value decomposition algorithm in $O(ml^2)$ time complexity. Note that there is no problem in using SVD for large datasets, as the complexity depends linearly on the number of vectors in the training set $\mathcal{D}$. For a thorough investigation of Moore–Penrose pseudo-inverse learning we refer the reader to the work of Górecki and Łuczak (2013).

The sigmoidal kernels in ELMs are constructed by randomizing their weights and thresholds. In the case of Gaussian kernels, they can be initialized by a subset of vectors of the training data $\mathcal{D}$. In both the cases the number of kernels has to be chosen manually. In the second instance we obtain the equivalence of the ELM with the original radial basis function network (Broomhead and Lowe, 1988). To keep the learning really time-efficient, we should try to use as few kernels as possible, because the complexity depends on the square of the number of kernels and linearly on the instance count. Conversely, the number of kernels should not be too small, as then we can end up with low accuracy. It can be noted that in nontrivial learning problems the Gaussian kernel in the ELM can be slightly more efficient (Chamara *et al*., 2013) than using sigmoidal functions. In the case of a support vector machine, the kernels (both in linear and non-linear cases) are defined by support vectors (Boser *et al*., 1992) extracted in the learning phase.

The advantage of the SVM is that the number of support vectors is selected during training (the QP optimization process), and therefore it is not chosen in a random manner. Although SVM learning is optimal (Vapnik, 1995) (optimal margin), it is not equivalent to the best generalization capability at all (see Section 4). However, it has been noticed that the SVM performs better than RBFN, as, for example in the works of Schölkopf *et al.* (1997; 1996) or Schwenker *et al*. (2001), where the SVM was compared to several versions of the RBFN (different kernel initializations, different phases of learning) and was slightly better than the best RBFN.

The main contribution of this article is the very combination of pseudo-inverse learning with selected prototype selection methods. The advantage of this strategy is that we no longer have to guess the number of kernels for ELMs or RBFNs. The only parameter of this combination is the width of the Gaussian kernel. There are several prototype selection methods, but in research we concentrate on the DROP2 and DROP4 algorithms, as proposed by Wilson and Martinez (2000), and those inspired by the encoding length principle (Cameron-Jones, 1995). Although initially prototype selection algorithms were set forth for lazy learning, the proposed combination of prototype selection and pseudo-inverse learning gives a much better accuracy than prototype selection methods alone.

How prototype selection methods work with some classifiers has been investigated before (Jankowski and Grochowski, 2004; Grochowski and Jankowski, 2004), but the results were not very promising, even for the combination of the SVM with prototype selection algorithms.

Additionally, Yousef and el Hindi (2005) presented an apparently wrongly investigated combination of some prototype selection methods with pseudo-inverse learning—it seems the authors obtained bad results by mistake (for more details, see the comments at the end of Section 4).

The following section presents a discussion on prototype selection methods and presents a chosen prototype selection algorithm for deeper analysis. Section 3 presents the main idea of the proposed algorithm and motivations. Section 4 is devoted to the analysis of the new algorithm on several data benchmarks and a comparison with best-known classification algorithms.

2. Prototype selection algorithms for pseudo-inverse learning

The problem of selecting instances from the original training set was investigated in many papers. Those methods can be divided into two groups: filters and prototype selections. We recommend some review articles concerning those methods (Garcia *et al*., 2012; Wilson and Martinez, 2000; Jankowski and Grochowski, 2004). The main goal of the filter group is to remove outliers or inconsistent instances from the original training data. Probably the most well-known algorithm in this group is edited nearest neighbours (Wilson, 1972) or the RNN (Gates, 1972). Methods from this group are characterized by a very small reduction of around 0–30% of instances.

The second group—prototype selection—is characterized (usually) by a much stronger reduction, mostly around 80–99%. However, some algorithms may have a reduction of around 50% too (although those could as well be described as filtering from a more practical point of view). The reduction strength is discussed thoroughly by Garcia *et al*. (2012) or Grochowski and Jankowski (2004). In the context of lazy learning, we can define an optimal instance selection as an algorithm which obtains both the highest accuracy and the highest reduction. Of course, generally, this is a hard problem. However, it does happen that some algorithms, like RMHC (Skalak, 1994) or Explore (Cameron-Jones, 1995), find very few prototypes whilst keeping very good accuracy.

Table 1. Results (accuracies, reduction rates and times) on large data sets from the work of Garcia *et al*. (2012).

method	acc	red.	time	N	method reference
RNG	0.82	**0.2**	14635	n	Sanchez *et al.*, 1997
SSMA	0.817	0.984	**45193**	n	Garcia *et al.*, 2008
RMHC	0.813	0.9	**77260**	n	Skalak, 1994
1NN	0.8				
HMNEI	0.801	**0.6**	80	n	Marchiori, 2008
DROP3	0.795	0.884	16899		Wilson and Martinez, 2000
CCIS	0.795	0.92	1349		Marchiori, 2010
FCNN	0.777	**0.695**	100	n	Angiulli, 2007

We should expect from a prototype selection algorithm to satisfy the following criteria:[1]

- It should not have too many configuration parameters. Too many parameters lead to problems with their optimization(s), which is usually very time-consuming.
- It should have possibly small (time) complexity.
- It should not finish with too few prototypes, as it could then compose an undersized kernel space and the final neural network may be of poor accuracy.
- The number of prototypes should not be too big, either. Since the complexity of the SVD algorithm depends on the squared number of the matrix columns, a growing number of kernels results in a quadratically longer learning time.
- Every nonlinear learning algorithm has to learn the borders of class regions. That is why the selected prototypes should also be smoothly placed around class borders.

In conclusion, we should avoid using filter methods or prototype selections whose time complexity is too high. Additionally, in Section 4 we show that using an excessively strong reduction leads to a lower accuracy of the classifier (compare results for the Explore algorithm).

To decide which prototype selection algorithm we should consider, recall the review presented by Garcia *et al*. (2012), an analysis on three types of data benchmarks: small, average and using big datasets. The results for small datasets in the context of kernel construction are obviously of minor importance. Results on average[2] and big data[3] as obtained by Garcia *et al*. (2012) are summarized in Tables 1 and 2.

Column *acc* shows averaged accuracies on test portions from cross-validation, column *red* shows average reduction strengths of a given method, column *time* is the learning time in seconds. For full details, see the work of Garcia *et al*. (2012). As discussed above, a prototype selection algorithm should not be too slow or result in an insufficient reduction. To simplify the analysis, we added column *N* with a value of 'n' in the rows where the learning is too slow or in the case of an insufficient reduction of the training set. It can be seen that in many cases the reduction rate[4] is lower than 80% or the learning time is impractically huge. Even among the plentiness of the reviewed methods, only in rare cases we can find methods with both a satisfactory reduction rate and low execution time.

It was not obvious that a combination of prototype selection for the initialization of kernel positions with pseudo-inverse learning of a neural network will be fruitful, since previously (Grochowski and Jankowski, 2004) we showed that combining prototype selection with the SVM (among others) leads to a degradation in accuracy.

Based on the above discussion and conclusions, we decided to analyze combinations of DROP algorithms (Wilson and Martinez, 2000) and those based on information coding (Cameron-Jones, 1995).

DROP2 algorithm. The family of DROP algorithms selects a relatively small and reasonable amount of instances. The research reported below concentrates on DROP2 and DROP4. DROP2 performs significantly better than DROP1, and DROP4 performs better than the previous versions, but is a little more computationally expensive (although of the same complexity). The main idea of the DROP2 algorithm lies in the definition of the set $\mathcal{A}(\mathbf{x}, k)$, which consists of the vectors for which $\mathbf{x}$ is one of their k nearest neighbours:

$$\mathcal{A}(\mathbf{x}, k) = \{\mathbf{x}' : \mathbf{x} \in N^k(\mathbf{x}')\}, \tag{7}$$

where $N^k(\mathbf{x}')$ is the set of the k nearest neighbours of $\mathbf{x}'$. The main concept of DROP2 is to delete all vectors whose removal does not change the classification of the remainder of the set $\mathcal{D}$. This idea produces the definition

[1] In the context of using prototypes as selection of placements for kernels.

[2] Cardinality of the dataset between 2001 and 20000 instances.

[3] Cardinality of the dataset greater than 20000 instances.

[4] By the reduction rate we mean the ratio of the number of instances removed to the cardinality of the original dataset.

Table 2. Results (accuracies, reduction rates and times) on average data sets from the work of Garcia *et al*. (2012).

method	acc	red.	time	N	method reference
RMHC	0.83	0.9	**12028**	n	Skalak, 1994
SSMA	0.829	0.98	**6306**	n	Garcia *et al.*, 2008
RNG	0.823	**0.116**	**1866**	n	Sanchez *et al.*, 1997
HMNEI	0.818	**0.535**	28.98	n	Marchiori, 2008
ModelCS	0.816	**0.065**	15.46	n	Brodley, 1995
CHC	0.809	0.991	**6803**	n	Cano *et al.*, 2003
GGA	0.808	0.908	**21262**	n	Kuncheva, 1995
1NN	0.806				
AllKNN	0.805	**0.21**	24.6	n	Aha *et al.*, 1991
POP	0.803	**0.08**	0.17	n	Riquelme *et al.*, 2003
RNN	0.802	0.945	**24480**	n	Gates, 1972
IB3	0.801	**0.767**	6.61	n	Aha *et al.*, 1991
MSS	0.801	**0.573**	7.9	n	Barandela *et al.*, 2005
FCNN	0.796	**0.76**	3.2	n	Angiulli, 2007
CNN	0.791	**0.737**	1.1	n	Hart, 1968
MENN	0.784	**0.314**	37	n	Hattori and Takahashi, 2000
Cpruner	0.76	0.889	35.3		Zhao *et al.*, 2003
Reconsistent	0.75	**0.68**	**1621**	n	Lozano *et al.*, 2003
DROP3	0.743	0.89	160		Wilson and Martinez, 2000
CCIS	0.713	0.95	12.4		Marchiori, 2010
MCNN	0.68	0.991	4.4		Devi and Mury, 2002
ICF	0.678	0.8	93		Brighton and Mellish, 2002

of the set $\mathcal{A}$, which simplifies the testing of the changes in classification to the elements of $\mathcal{A}$, contrary to testing on the whole of $\mathcal{D}$.

The algorithm model of DROP2 is as follows:

1: function DROP2($\mathcal{D}$,k)
2: **repeat**
3: **for** $\mathbf{x}_i$ in $\mathcal{D}$ in dist-order **do**
4: delete $\mathbf{x}_i$ if it does not change the
5: classification of instances from $\mathcal{A}(\mathbf{x}_i, k)$
6: **end for**
7: **until** no changes in $\mathcal{D}$
8: **return** $\mathcal{D}$

The 'dist-order' above defines a descending order of instances (in $\mathcal{D}$) with respect to the distance to their nearest enemy (the nearest instance from an opposite class). The previous version of DROP1 did not use the 'dist-order', and its accuracy was significantly worse on average. The outer loop usually iterates a few times. The inner loop iterates for each instance in $\mathcal{D}$. The time complexity is $O(m^3n)$. The reduction of $\mathcal{D}$ is quite strong; for details, please see the works of Wilson and Martinez (2000) or Grochowski and Jankowski (2004).

DROP4. The next version of the DROP algorithm begins with eliminating inconsistent instances. An *inconsistent* instance is one whose neighbours are mostly from a different class, but additionally the deletion of this instance would not decrease classification accuracy. The test of inconsistency is performed for each instance.

The algorithm model of DROP4 is as follows:

1: function DROP4($\mathcal{D}$,k)
2: **for** $\mathbf{x}_i$ in $\mathcal{D}$ **do**
3: delete $\mathbf{x}_i$ if $\mathrm{kNN}(\mathbf{x},k) \neq y_i$ and it will not
4: change classification of instances
5: from $\mathcal{A}(\mathbf{x}_i, k)$
6: **end for**
7: **repeat**
8: **for** $\mathbf{x}_i$ in $\mathcal{D}$ in dist-order **do**
9: delete $\mathbf{x}_i$ if it will not change
10: classification of instances from $\mathcal{A}(\mathbf{x}_i, k)$
11: **end for**
12: **until** no changes in $\mathcal{D}$
13: **return** $\mathcal{D}$

Here $\mathrm{kNN}(\mathbf{x},k)$ is the result of kNN classification of $\mathbf{x}$.

Owing to the deletion of inconsistent prototypes, the main part of DROP4 is somewhat smoother, as it does not depend on inconsistent instances.

DROP3 is just slightly different from DROP4—in the condition of deletion in the first loop, the right-hand side of the conjunction is dropped.

Encoding length. The next three algorithms are based on the concept of the encoding length (Cameron-Jones, 1995). The heart of the idea is Cameron's criterion below, which should be minimized through the extraction of

unnecessary instances from the original dataset $\mathcal{D}$:

$$J(m,m',q) = F(m',m) + m' \log_2 K + F(q, m-m') + q \log_2(K-1), \quad (8)$$

where m is the number of instances in the original dataset $\mathcal{D}$, m' is the number of instances in the prototype set S, and K is the number of classes, q defines the number of badly classified instances in $\mathcal{D} \setminus S$, where $F(m,n)$ is defined by

$$F(m,n) = \log^* \left(\sum_{i=0}^{m} \frac{n!}{i!(n-i)!} \right), \quad (9)$$

$$\log^* n = \arg\min_k F'(k) \geq n, \quad (10)$$

where $F'(0) = 1, F'(i) = 2^{F'(i-1)}$.

It can be easily seen that Cameron's criterion is smaller if the reduction is stronger and does not increase the error on $\mathcal{D} \setminus S$.

The (original) encoding length algorithm starts with all instances of $\mathcal{D}$ as prototypes, and iteratively tries to remove each instance, if only this reduces Cameron's criterion. Its scheme is as follows:

```
1: function EncLen(D, startS = D, R = D)
2: S = startS
3: m' = m
4: q = numerOfErrors(D \ S, S)
5: j = J(m, m', q)
6: for x_i in R do
7:    S = S \ {x_i}
8:    m' = m' - 1
9:    q = numerOfErrors(D \ S, S)
10:   j' = J(m, m', q)
11:   if j' ≤ j then
12:      j = j'
13:   else
14:      S = S ∪ {x_i}
15:      m' = m' + 1
16:   end if
17: end for
18: return S
```

Here $numerOfErrors(\mathcal{D} \setminus S, S)$ is defined as the number of classification errors obtained on instances from $\mathcal{D} \setminus S$ using the current set of prototypes S. The first argument of the EncLen $\mathcal{D}$ defines the learning set, the second argument $startS$ defines the initial set of prototypes, the third argument R of EncLen defines which vectors will be analyzed for possible removal. The call to start the original EncLen is `EncLen(D,D,D)`. What this means is that the training dataset is $\mathcal{D}$, the algorithm starts with the whole of $\mathcal{D}$ as the prototype set, and in the main loop all vectors will be checked for possible removal. Please compare it to the calls of EncLen in the algorithms EncLenGrow and DEL (below), where EncLen will be called with different arguments in other contexts.

Explore. The next algorithm, Explore, also uses Cameron's criterion, but in a more sophisticated manner. It mixes a few goals in one, bigger scheme. The first part of Explore is the EncLenGrow sub-procedure, which starts from an empty set of prototypes S and tries, for each instance in $\mathcal{D}$, to add it as a prototype, if only this reduces Cameron's criterion. In contrast to EncLen, this procedure tries to add, not to remove. However, after this phase the EncLen procedure is called to remove prototypes from S, should it reduce Cameron's criterion (as before). Such a strategy prevents it from retaining unnecessary prototypes. Its scheme is as follows:

```
1: function EncLenGrow(D)
2: S = ∅
3: m' = 0
4: q = numerOfErrors(D \ S, S)
5: j = J(m, m', q)
6: for x_i in D do
7:    S = S ∪ {x_i}
8:    m' = m' + 1
9:    q = numerOfErrors(D \ S, S)
10:   j' = J(m, m', q)
11:   if j' ≤ j then
12:      j = j'
13:   else
14:      S = S \ {x_i}
15:      m' = m' - 1
16:   end if
17: end for
18: S = EncLen(D, S, S)
19: return S
```

Having finished the EncLenGrow part, the Explore algorithm tries to tune S by several iterations of attempting actions randomly chosen between addition or removal of a single instance, or substitution of one instance in S with an instance from $\mathcal{D} \setminus S$, but the actions are only executed if they result in decreasing Cameron's criterion. Finally, the scheme of Explore can be given as follows:

```
1: function Explore(D, p)
2: S = EncLenGrow(D)
3: for i=1 to p do
4:    switch (random action 1 of 3)
5:    case 1:
6:       try to add a random instance from D \ S
7:          only when J will decrease
8:    case 2:
9:       try to remove random instance from S
10:         only when J will decrease
```

```
    case 3:
      try to
        add random instance from D \ S
      AND
        remove random instance from S
      only when J will decrease
    end switch
end for
return S
```

A typical value of p is 1000.

DEL. The last prototype selection algorithm analyzed in this article also uses Cameron's criterion, although in a different way compared to the previous ones. First, we construct a set R of instances from $\mathcal{D}$ whose classes are inconsistent with their neighbours' labels (i.e., the instances badly classified by the kNN). After that, each instance from R is removed if only this reduces Cameron's criterion. This is another form of reducing inconsistent instances (compare with DROP4). In the next phase, the instances from S are sorted in descending order with respect to the distance to their nearest enemy (cf. DROP2). Using such an order, the EncLen procedure is called repeatedly until no instance is removed during the procedure. The final scheme of the DEL procedure is presented below:

```
function DEL(D)
R = set of badly classified instances from D
S = EncLen(D, D, R)
S = sort(S) in descending order by distance
      to nearest enemy
repeat
  S' = S
  S = EncLen(D, S, S)
until S = S'
return S
```

All of the prototype selection algorithms presented in the above review will be used as elements of the entire learning algorithm introduced in the next section.

The complexities of the above algorithms are $O(m^3n)$ and $\Omega(m^2n)$. All algorithms except Explore provide an instance reduction rate close to 0.9. The reduction rate of the Explore algorithm is around 0.98.

3. Prototype-based kernels for extreme learning machines and radial basis function networks

Whenever we use RBFN or ELM learning, the model is defined by a linear combination ($\mathbf{w}$) of kernels $g_j(\mathbf{x}, \mathbf{x}_j)$:

$$F(\mathbf{x}) = \sum_{j=1}^{l} w_i g_j(\mathbf{x}, \mathbf{x}_j) + w_0. \tag{11}$$

The main goal of this article is to propose learning algorithms that automatically choose kernel placements and the number of kernels for the RBFN and the ELM, contrary to their original versions, as learning algorithms with manual selection of the number of kernels and their placements. Additionally, the complexity of the learning algorithms should be as small as possible.

The proposed algorithms are combinations of SVD learning with prototype selection. Currently, such combinations' complexity is $O(ml^2 + m^3n)$, as so are the costs of SVD and prototype selection (where l is the number of kernels). In the case of manual selection of the number of kernels, a validation process must be used, e.g., cross-validation, and finally, such learning uses much more CPU time (being a multi-learning and testing process).

To eliminate this disadvantage, we can first start with one of the prototype selection algorithms, and the selected prototypes can define the placements of the new kernels (typically Gaussian) for the RBFN or the ELM. Based on the selected prototypes, the kernels are defined. Next, using the aforementioned kernels, we move from the original space of the data $\mathcal{D}$ to a new kernel space with a data matrix G, obtained by computing each kernel for each of the data instance in $\mathcal{D}$ (cf. Eqns. (4) and (5)). The last two steps are the computation of the pseudo-inverse H by SVD and the multiplication of H by the vectors of class labels. The scheme of this algorithm is as follows:

```
function ProtoLearning(D, PrototypeSelection)
[p_1, ..., p_l] = PrototypeSelection(D)
G_ij = g_j(x_i, p_j)        ∀ i∈[1,...,m], j∈[1,...,l]
G' = [1  G]
H = svd_pseudo_inv(G')
w = Hy
return weights w and kernels g_*
```

Here $g_j(\mathbf{x})$ is the (Gaussian) kernel placed at $\mathbf{x}'_j$, G is a matrix with one column per kernel, each kernel being evaluated on every instance in the dataset. The matrix G' has an additional column of $1'$s with respect to w_0 in Eqn. (11).

In the case of the original RBFN, the first two instructions of the above scheme are substituted with random selection of instances from the dataset and g_j is a (Gaussian) kernel, while in the case of the ELM, originally g_j is a sigmoidal function with random projection (random weights), or a Gaussian kernel.

Now, it is clear that the number of kernels l has strong influence on the computational costs, as the complexity of SVD is $O(ml^2)$. The combination of prototype selection with SVD amounts to a total pessimistic complexity of $O(ml^2 + m^3n)$. But, currently, we are working on faster versions of DROP algorithms and encoding length-based ones, and we are close to obtaining an estimated

complexity close to $O(nm \log_2 m)$. Such complexity can be obtained using locality-sensitive hashing supported by additional data structures used to decrease recalculations. This is very important in the context of huge datasets.

4. Result analysis of pseudo-inverse learning with prototype selection-based kernels

The goal of this section is to present a comparative analysis of the presented `ProtoLearning` algorithm with other known algorithms.

To make a comparison of different algorithms, we use around 40 datasets from the UCI machine learning repository (Merz and Murphy, 1998) devoted to classification problems. The datasets differ in the origin, goals, the number of instances, features and classes, to present an objectively realistic behaviour of the new algorithms proposed. A summary of the datasets' properties is presented in Table 3.

All tests were conducted on the basis of 10-fold stratified cross-validation repeated 10 times. For each test the dataset was standardized. All experiments were made in the data mining framework called Anemon written in C#/.Net. Anemon is our own framework with many algorithms including neural networks, machine learning and statistics. Should the reader be interested in the details of the presented algorithm, we can share source codes.

The general `ProtoLearning` algorithm was combined with all the prototype selection algorithms described above. In this way, six combinations were obtained: Drop2-NN, Drop3-NN, Drop4-NN, Explore-NN, EncLen-NN, Del-NN, each being a neural network with kernel placements defined by the prototype algorithm respective to the combination name. Additionally, we present results achieved by a 1-NN classifier using the results of prototype selection algorithms.

The aforementioned networks were compared with the following learning machines: a linear discriminant learned by SVD (LDA), extreme learning machines with a sigmoidal kernel function (ELM), a radial basis function network with a Gaussian kernel (RBFg), k nearest neighbours ($k = 5$), a support vector machine with a linear kernel (L-SVM) and with a Gaussian kernel (SVM).

We formed three tables to compare several configurations of learning machines. The first one compares all combinations of prototype selection for kernel construction with neural networks (see Table 4). Two best prototype-based kernel selection methods—DROP2 and EncLen—were compared with known classifiers in Tables 6 and 7.

In Table 5 we present the results of prototype selection algorithms, without using them as the source of kernel positions. The comparison of results in Table 4 with those in Table 5 clearly shows that the combination of prototype selection methods with neural networks leads to much better results than using prototype selection algorithms *alone* (in the 1-NN scenario, as previously mentioned).

Each learning algorithm was always used with the same learning parameters for each benchmark dataset (no manual parameter tuning was done). ELMs and RBFNs were learned with 160 random kernels.[5] The placements of the RBFN's kernels were vectors randomly selected from the given benchmark dataset. ELMs use random weights ($\mathbf{b}$) and thresholds (θ), as in Eqn. (2). RBFg, DROP*-NN, Explore-NN and Del-NN were used with the Gaussian function (Eqn. (3)) with $\gamma = 2^{-5}$. The kNN was used with $k = 5$ and the Euclidean metric. L-SVM and SVM were used with $C = 1$. SVM was used with a Gaussian kernel with $\gamma = 0.1$[6]

To visualize the performance of all algorithms, we present the average accuracy and the rank for each benchmark dataset and learning machine. For each benchmark and each machine we used the same seed for randomization, which enabled us to employ paired t-tests to provide more trustful analysis.

Notice that each cell of the main part of Tables 4–7 is in the form

$$acc + std(rank), \tag{12}$$

where acc is the average test accuracy (for a given dataset and a given learning machine), std is its standard deviation and $rank$ is the rank as described below. If a given cell of the table is in bold, this means that the result is the best for a given dataset or not significantly worse than the best one (rank equal to 1 = winners).

Cumulative results of the analysis are presented by the number of wins (and unique wins), with the mean rank and mean accuracy as complementary information.

The *ranks* are calculated for each machine for a given dataset $\mathcal{D}$ as follows. First, for a given benchmark dataset $\mathcal{D}$ the averaged accuracies of all learning machines are sorted in descending order. The machine with the highest average accuracy is ranked 1. Then, the following machines in the accuracy order whose accuracies are not statistically different from the result of the first machine are ranked 1, until a machine with a statistically different result is encountered. That machine starts the next rank group (2, 3, and so on), and an analogous process is repeated on the remaining (yet unranked) machines.

A meta-code of the above procedure is given below. The `rank` function computes ranks for comparing learning machines based on the array of

[5]This is 'not too small' and 'not too big' for the analyzed benchmarks. The idea behind choosing this value was to keep the number of kernels constant, but not too big, as with the growth in the number of kernels the complexity grows quadratically.

[6]Note that on the average the SVM prefers (in terms of achieving good accuracy) a different γ than RBF neural networks.

Table 3. Summary of data set properties used in the analysis of learning algorithms.

data set	# classes	# instances	# features	# ordered f.
arrhythmia	11	63	279	206
autos	6	159	25	15
balance-scale	3	625	4	4
blood-transfusion-service-center	2	748	4	4
breast-cancer-wisconsin-diagnostic	2	569	30	30
breast-cancer-wisconsin-original	2	683	9	9
breast-cancer-wisconsin-prognostic	2	194	33	33
breast-tissue	6	106	9	9
car-evaluationNOM	4	1728	21	0
cardiotocography-1	10	2126	21	21
cardiotocography-2	3	2126	21	21
chess-king-rook-vs-king-pawn	2	3196	38	38
cmc01	3	1473	24	24
congressional-voting-records	2	232	16	16
connectionist-bench-sonar-mines-vs-rocks	2	208	60	60
connectionist-bench-vowel-recognition-deterding	11	528	10	10
cylinder-bands	2	277	39	18
dermatology	6	358	34	1
ecoli	8	336	7	7
glass	6	214	9	9
habermans-survival	2	306	3	3
hepatitis	2	80	19	6
ionosphere	2	351	34	34
iris	3	150	4	4
libras-movement	15	360	90	90
liver-disorders	2	345	6	6
lymph	4	148	18	3
monks-problems-1	2	556	15	15
monks-problems-2	2	601	15	15
monks-problems-3	2	554	15	15
parkinsons	2	195	22	22
pima-indians-diabetes	2	768	8	8
sonar	2	208	60	60
spambase	2	4601	57	57
spect-heart	2	267	22	22
spectf-heart	2	267	44	44
statlog-australian-credit	2	690	38	38
statlog-german-credit	2	1000	24	24
statlog-heart	2	270	20	7
statlog-vehicle-silhouettes	4	846	18	18
teaching-assistant-evaluation	3	151	54	54
thyroid-disease	3	7200	21	21
vote	2	232	16	0
wine	3	178	13	13
zoo	7	101	17	1

their accuracy vectors A_i for the given dataset. We are using the paired t-test: the `pttest` function that returns true if two chains of accuracies are not statistically different (with threshold α). Its scheme is as follows:

1: function ranks($A_1, \ldots, A_b, \alpha$)
2: **for** i=1 **to** b **do**
3: m_i = mean_acc(A_i)
4: **end for**
5: $[m_{k_1}, \ldots, m_{k_b}]$ = sort($[m_1, \ldots, m_b]$)
6: $r = 1$
7: $i = j = 1$
8: **while** $i \leq b$ **do**
9: **if not** pttest(A_{k_j},A_{k_i},α) **then**
10: $r = r + 1$
11: $j = i$
12: **end if**
13: $r_{k_s} = r$;

14: $s = s + 1$
15: **end while**
16: **return** $[r_1, \ldots, r_b]$

Thanks to the concept of the rank, we recognize not only the winners and the defeated, but more groups depending on really significant differences. This helps us to see how strongly a given machine defeats another in the meaning of statistical differences.

The last two rows of Tables 4–7 present cumulative results. The *mean rank* row presents the most significant information about the average ranks of the machines—for each machine, its average rank over all datasets is presented with standard deviation. The third row presents the numer of wins (how many times the given machine was the best or was not significantly worse than the best) for each machine, and in brackets, the number of unique wins. By a unique machine win we mean the case when all other machines are significantly worse. More simply, if the winner machine was the only one to achieve rank 1 (as described above), it is a unique winner.

From the *mean rank* row in Table 4, we can find that the best mean rank[7] 1.38 is assigned to EncLen-NN and 1.4 is assigned to DROP2-NN. Also, the number of wins[8] is the biggest for DROP2-NN (31) and for EncLen-NN (30). DROP4 came very close to the previous results, with a mean rank of 1.49 and 28 wins. The worst kernels were provided by Explore and DEL. We should remember, however, that those algorithms keep only a very small amount of original vectors as prototypes (Grochowski and Jankowski, 2004).

Analyzing Tables 4 and 5, we can find the differences between RBF/ELM neural networks with kernels initialized by prototypes and prototype methods alone. It is quite clear that the proposed method yields a significantly better classification.

In Tables 6 and 7 we present a comparison of the two best kernel providers (EncLen-NN and DROP2) with known learning algorithms. In Table 6 the best learning machine is the proposed EncLen-NN. Its mean rank is 1.87 and the win number is 18, while six of them are unique. The highest number of wins and the smallest value of the average rank means that the EncLen-NN is significantly better than the other algorithms.

The second-best is RBFg with a mean rank of 2.04 and 16 wins. You can see that the third result was LDA with a significantly smaller mean rank 2.8.

In Table 7, the best algorithm is DROP2-NN with a mean rank of 1.98 and 19 wins (6 of them unique). The second-best result was obtained by RBFg with a mean rank of 2.07 and 16 wins (3 of them unique).

In all cases the proposed learning machines were significantly better than other learning machines. Please note that the proposed methods are significantly better than the SVM or the ELM.

What is more, we can easily see that the proposed classification algorithms perform even better than sophisticated committees of learning machines, for example those proposed by Woźniak and Krawczyk (2012). The reader can compare the averaged accuracies presented in the above tables with those given by Woźniak and Krawczyk (2012). This shows that trustful learning can be now much (computationally) simpler and more accurate.

As mentioned earlier, in our articles (Jankowski and Grochowski, 2004; Grochowski and Jankowski, 2004), we proposed and analyzed a combination of prototype selection with a neural network or an SVM, but the results were not very promising. Later, a similar idea was presented by Yousef and el Hindi (2005). However, their results are, in our opinion, erroneously bad—the results on several datasets are much worse than ours, as presented in the above tables. One of the biggest differences lies in the construction of the Gaussian function. Yousef and el Hindi claim that they use individual standard deviation σ_j per attribute instead of γ^{-1}, as in Eqn. (3):

$$\sigma_j = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_{ij} - \bar{x}_j)^2}, \tag{13}$$

where $\mathbf{x}_{ij}$ is the value of the j-th attribute for the i-th instance and $\bar{x}_j$ is the mean of the attribute j. After adapting our own code to use a Gaussian kernel as presented by Yousef and el Hindi (2005), we obtained worse results than presented in this article, yet they were somewhat different from those by Yousef and el Hindi (2005). This suggests that the authors of the cited article may have made some other mistakes. Finally, the idea of combining prototype methods with neural networks has been erroneously viewed to be ill-advised so far.

5. Summary

The proposed learning algorithms, instead of randomly selecting the kernels for the RBFN or the ELM, use a prototype selection algorithm, and after that, the selected prototypes compose the kernels for the RBFN or the ELM, learned by a pseudo-inverse matrix (by SVD). Test results clearly show that such new algorithms are significantly better than learning machines such as the SVM, RBFN, ELM or kNN, and are additionally more stable.

Thanks to this concept, the new algorithms automatically select the kernels and their number. There is no need for manually tuning the number of kernels. What is more, there is no need for manually tuning any other parameter in the new algorithm, which is a big advantage, as it implies no need for inner cross-validation. It is a good alternative for the costly manual or automatic (for

[7] Smaller value means better.
[8] Bigger means better.

Table 4. Comparison of neural networks with prototype-based kernel selection by DROP2, DROP3, DROP4, Explore, EncLen and DEL.

	Drop2-NN	Drop3-NN	Drop4-NN	Explore-NN	EncLen-NN	Del-NN
arrhythmia	32±8.9(2)	**33.6±9.4(1)**	**34±8.9(1)**	**33.7±9.3(1)**	**34±9.6(1)**	31.7±7.7(2)
autos	72.2±12(2)	67.6±12(3)	69.4±12(3)	47.9±12(4)	**75.1±12(1)**	73.1±11(2)
balance-scale	**90.8±1.9(1)**	90.4±1.7(2)	90.3±1.8(2)	89.3±2.5(3)	90.2±1.9(2)	**90.9±1.4(1)**
blood-transfusion	79.1±3.6(2)	**79.5±3.7(1)**	**79.4±3.7(1)**	76.1±1.2(4)	**79.4±3.7(1)**	76.7±2.2(3)
breast-cancer-diagnostic	**96.6±2.4(1)**	**96.3±2.5(1)**	**96.6±2.4(1)**	92.3±6.6(2)	**96.3±2.6(1)**	93.6±3.8(2)
breast-cancer-original	96.8±2(2)	96.7±2(2)	96.7±2.2(2)	**97±1.9(1)**	**96.8±2.1(1)**	**96.9±1.8(1)**
breast-cancer-prognostic	**77.4±7(1)**	**78.4±6.1(1)**	**77.3±7.1(1)**	76.3±2.2(2)	**78.3±7.2(1)**	76.3±2.2(2)
breast-tissue	61.7±14(2)	**63.5±12(1)**	61.2±15(2)	**65.9±11(1)**	60.3±11(2)	**64±13(1)**
car-evaluation	90.9±2.2(2)	91.2±2.1(2)	91.4±2(2)	71.2±3.9(4)	**94.4±1.7(1)**	84.9±2.9(3)
cardiotocography-1	**84.5±2.2(1)**	**84.2±2.1(1)**	**84.5±2.3(1)**	76±10(3)	**84.2±2.2(1)**	83±2.1(2)
cardiotocography-2	**92.7±1.6(1)**	92.6±1.7(2)	**92.9±1.7(1)**	83.4±3(4)	92.6±1.9(2)	89±2.2(3)
chess-rook-vs-pawn	**98.9±0.64(1)**	98.6±0.67(2)	**98.9±0.62(1)**	84.1±12(5)	98.2±0.74(3)	92.7±1.7(4)
cmc	49±4.3(2)	**51±3.5(1)**	49.3±4(2)	49±5.3(2)	48.4±4.3(3)	48.1±4.2(3)
congressional-voting	**96.5±3.7(1)**	**96.6±3.7(1)**	**96.5±3.7(1)**	91±6.1(3)	95.3±4.8(2)	90.3±6.3(3)
connectionist-bench-sonar	**84.5±6.6(1)**	81.8±7.4(2)	82.8±7(2)	67.2±9.2(4)	**83.6±6.6(1)**	70.6±9(3)
connectionist-bench-vowel	**96.7±2.5(1)**	**96.7±2.7(1)**	**96.6±2.6(1)**	68.1±9.4(3)	96±2.9(2)	96.2±2.6(2)
cylinder-bands	**68.9±4.4(1)**	67.8±4.6(2)	**68.3±4.9(1)**	64.3±0.96(3)	68±5(2)	64.3±0.96(3)
dermatology	**95.7±3.1(1)**	94.9±3.3(2)	95.1±3.3(2)	87.6±6.1(4)	**95.5±3(1)**	93.9±3.9(3)
ecoli	**86.5±5.4(1)**	86.1±5.2(2)	**86.6±5(1)**	85.3±5.2(3)	86.3±5.1(2)	**87.1±4.9(1)**
glass	**66±9.8(1)**	**66.5±9.2(1)**	**66±9.7(1)**	62±9(2)	**65.3±9.3(1)**	**66.8±9.1(1)**
habermans-survival	**74±5.5(1)**	**73.8±5.8(1)**	72.8±5.9(2)	**73.5±2.7(1)**	73.1±5.9(2)	**73.4±1.9(1)**
hepatitis	**84.4±12(1)**	**84.5±11(1)**	**85.3±11(1)**	82.8±7.3(2)	**85.4±11(1)**	82.8±7.1(2)
ionosphere	**93.2±3.9(1)**	91.6±4.5(3)	92.4±4.1(2)	79.4±9(4)	**92.7±4.3(1)**	91.1±5(3)
iris	**96.9±4.2(1)**	**96.5±4.8(1)**	**96.5±4.8(1)**	91.7±7.7(3)	**96.5±4.7(1)**	95.7±4.9(2)
libras-movement	**86±5.7(1)**	82.7±5.5(3)	85±5.8(2)	61.1±7.8(4)	**86.1±5.1(1)**	84.9±5.8(2)
liver-disorders	**68.4±7.2(1)**	**68.5±7.3(1)**	**68.1±7(1)**	58.9±5.7(3)	**67.3±8.5(1)**	61.6±7.3(2)
lymph	84.3±8.9(2)	**85.7±8.2(1)**	**86.4±9(1)**	74.1±11(4)	**85.1±8.6(1)**	81.7±9.8(3)
monks-problems-1	**99.9±0.69(1)**	**99.9±0.44(1)**	**99.9±0.69(1)**	68.8±7.1(3)	**99.9±0.44(1)**	77.5±7.6(2)
monks-problems-2	59.5±6.9(3)	61.7±6.6(2)	61.3±6.2(2)	**65.7±0.85(1)**	60.4±7(2)	**65.7±0.82(1)**
monks-problems-3	**98.8±1.6(1)**	**98.8±1.5(1)**	**98.8±1.5(1)**	88.8±9.3(4)	98.3±2(2)	95.4±3.5(3)
parkinsons	**89.3±6.3(1)**	**89.1±6.6(1)**	**89.1±6.3(1)**	79.7±7.4(3)	**89.5±6.8(1)**	82.2±7.7(2)
pima-indians-diabetes	72.5±4.6(3)	74±5(2)	72.7±4.9(3)	**74.8±4.9(1)**	73.3±4.9(2)	**75±5.2(1)**
sonar	**84.5±6.6(1)**	81.8±7.4(2)	82.8±7(2)	67.2±9.2(4)	**83.6±6.6(1)**	70.6±9(3)
spambase	**91.2±1.2(1)**	**91.2±1.2(1)**	**91.3±1.3(1)**	82.9±8.3(4)	91.1±1.3(2)	87.6±1.6(3)
spect-heart	**82.8±6.1(1)**	**82.5±6.1(1)**	**82.6±6.2(1)**	79.7±3.1(3)	**82.7±5.7(1)**	81.2±4.8(2)
spectf-heart	**79.9±6.8(1)**	**79.6±7.1(1)**	**79.1±7(1)**	**79.3±2(1)**	**79.8±7(1)**	**79.4±1.7(1)**
statlog-australian-credit	**84.2±4(1)**	**84.8±4.4(1)**	**84.4±4.2(1)**	75.9±7.4(3)	**84.8±4.3(1)**	83.6±4.2(2)
statlog-german-credit	74.2±4.3(2)	**75±4(1)**	74.2±3.8(2)	71.1±2.6(3)	**74.7±3.8(1)**	70.6±2.2(3)
statlog-heart	81±7.4(2)	**82.3±7.9(1)**	**81.8±8(1)**	80.3±7.7(2)	**82.6±7.6(1)**	79.9±7.8(2)
statlog-vehicle	**83.4±3.8(1)**	**83.2±3.7(1)**	**83.2±3.9(1)**	60.8±12(3)	**83.2±4.1(1)**	79.9±3.7(2)
teaching-assistant	51±11(3)	46.3±12(4)	50.3±11(3)	41.4±11(5)	**56.5±12(1)**	53.8±12(2)
thyroid-disease	**96.2±0.51(1)**	95.8±0.52(3)	**96.2±0.53(1)**	93.5±0.35(5)	96±0.54(2)	94.1±0.51(4)
vote	**96.8±3.2(1)**	**96.5±3.5(1)**	**96.4±3.9(1)**	92±5.5(2)	**96.3±3.9(1)**	92.2±5.3(2)
wine	**97.8±3.5(1)**	**98.1±3.1(1)**	**98±3.1(1)**	97.1±3.6(2)	**97.7±3.3(1)**	96.9±3.7(2)
zoo	61.8±12(3)	48±11(5)	54.8±11(4)	40.4±2.4(6)	**70.5±13(1)**	67.5±12(2)
Mean Rank	1.4±0.099	1.6±0.14	1.49±0.11	2.98±0.19	1.38±0.087	2.2±0.12
Wins[unique]	31[0]	27[1]	28[0]	7[0]	30[4]	9[0]

Table 5. Comparison of prototype selection algorithms: DROP2, DROP3, DROP4, Explore, EncLen and DEL.

	Drop2	Drop3	Drop4	Explore	EncLen	Del
arrhythmia	49.67±21(2)	**52.9±21(1)**	**54.31±23(1)**	**55.4±22(1)**	48.83±23(2)	49.05±23(2)
autos	67.74±11(3)	56.23±10(5)	63.86±12(4)	48.87±9.9(6)	**73.1±12(1)**	70.6±12(2)
balance-scale	74.64±5.4(4)	80.4±4.1(2)	79.82±4.3(2)	**81.75±5.3(1)**	72.18±6.5(5)	78.83±4.8(3)
blood-transfusion	69.28±6.2(4)	75.11±4.6(2)	71.3±5.8(3)	**76.02±1.2(1)**	67.68±5.8(5)	**75.83±4.2(1)**
breast-cancer-diagnostic	91.9±3.3(2)	**93.62±2.8(1)**	**93.44±3.2(1)**	**94.25±3.9(1)**	86.94±4.6(4)	89.08±5.8(3)
breast-cancer-original	93.46±2.8(3)	95.19±2.5(2)	94.72±2.8(2)	**96.68±2(1)**	90.5±4.7(4)	**96.3±2.1(1)**
breast-cancer-prognostic	68.87±10(3)	72.39±9.9(2)	66.96±10(3)	**76.17±2.7(1)**	68.49±9.8(3)	**76.32±2.2(1)**
breast-tissue	**66.22±12(1)**	63.15±14(2)	**65.68±14(1)**	61.02±12(2)	**66.4±13(1)**	**64.71±13(1)**
car-evaluation	80.23±2.8(2)	79.46±2.8(3)	79.79±2.8(2)	70.26±1.1(5)	**85.98±2.7(1)**	75.57±2.7(4)
cardiotocography-1	70.67±3.3(2)	70.91±2.9(2)	**71.1±3(1)**	63.13±7.2(4)	**71.63±3.1(1)**	67.36±3.2(3)
cardiotocography-2	86.8±2.4(3)	**87.84±1.8(1)**	87.34±2(2)	83.4±3.1(5)	86.32±2.4(3)	84.64±2.6(4)
chess-rook-vs-pawn	90.46±1.7(2)	90.54±1.6(2)	**91.04±1.6(1)**	76.64±6.2(5)	82.23±2.5(4)	83.24±2.4(3)
cmc	42.95±3.9(2)	**45.23±3.4(1)**	43.01±3.9(2)	43.31±4(2)	42.27±4.1(3)	41.99±4(3)
congressional-voting	81.31±9.3(4)	**91.15±5.5(1)**	89.35±7.3(2)	**90.93±6.5(1)**	84.83±9.1(3)	87.97±6(2)
connectionist-bench-sonar	**81.96±8.6(1)**	78.69±8.9(3)	80.53±7.9(2)	70.52±10(4)	**81.74±8.3(1)**	66.78±10(5)
connectionist-bench-vowel	**96.33±2.8(1)**	94.45±3.7(3)	**96.02±3(1)**	51.92±9.1(4)	95.32±3.3(2)	95.55±3.1(2)
cylinder-bands	**65.43±8.1(1)**	60.91±8.3(3)	62.68±8.6(2)	**64.26±0.96(1)**	**63.88±8.3(1)**	**64.26±1.2(1)**
dermatology	**88.02±5(1)**	**87.34±4.3(1)**	**87.6±4.7(1)**	81.85±6.1(3)	85.84±5.2(2)	85.72±6.1(2)
ecoli	79.85±6.6(3)	**84.78±5.2(1)**	**84.13±4.8(1)**	81.47±6.5(2)	77.66±7.3(4)	82.42±6.3(2)
glass	66.53±9.5(2)	**68.33±8.4(1)**	67.22±9.4(2)	60.21±9.6(3)	**69±8.4(1)**	65.93±9.1(2)
habermans-survival	65.62±8.9(4)	69.49±7(2)	67.78±6.9(3)	**73.14±2.2(1)**	66.58±7.8(3)	**73.14±3.1(1)**
hepatitis	**82.25±11(1)**	**81.38±13(1)**	**82.38±13(1)**	**83.75±5.8(1)**	79.63±15(2)	**83.25±7.4(1)**
ionosphere	81.12±6.9(2)	**83.96±5(1)**	80.48±7.4(2)	78.06±7.4(3)	**84.89±7.3(1)**	81.08±7.5(2)
iris	92.6±6.7(2)	**93.87±6.2(1)**	**93.87±6.2(1)**	**93.47±7.6(1)**	90.87±7.5(3)	88.33±8.2(4)
libras-movement	**81.75±6.7(1)**	76.58±6.2(3)	**81.5±6.4(1)**	56.86±8.4(4)	**80.83±6.8(1)**	79.75±7.1(2)
liver-disorders	**61.66±8.7(1)**	59±8(2)	**60.29±8.2(1)**	58.66±5.7(2)	**61.71±7.9(1)**	57.69±7.6(2)
lymph	**76.7±9.6(1)**	75.49±11(2)	**77.19±11(1)**	70.5±12(3)	**76.58±11(1)**	68.7±12(3)
monks-problems-1	**94.66±2.9(1)**	**94.69±2.9(1)**	**94.62±2.9(1)**	70.42±7.1(4)	88.82±5.1(2)	74.41±6.6(3)
monks-problems-2	55.96±6.5(4)	58.67±5(2)	57.42±6.2(3)	**65.72±0.79(1)**	52.32±6.7(5)	**65.72±0.79(1)**
monks-problems-3	**93.18±3.5(1)**	**93.39±3.5(1)**	**93.39±3.5(1)**	84.59±6.8(2)	85.77±4.6(2)	82.52±6.6(3)
parkinsons	**87.73±7.1(1)**	**86.89±7.6(1)**	**87.82±7.4(1)**	80.83±7(3)	85.51±7.8(2)	83.41±7.7(2)
pima-indians-diabetes	68.71±5.6(3)	**72.08±5.5(1)**	70.41±5(2)	**72.59±5.7(1)**	67.93±5.6(3)	69.89±5.6(2)
sonar	**81.96±8.6(1)**	78.69±8.9(3)	80.53±7.9(2)	70.52±10(4)	**81.74±8.3(1)**	66.78±10(5)
spambase	86.34±1.6(3)	**88.01±1.8(1)**	87.64±1.6(2)	82.46±4.3(5)	82.64±2.2(5)	84.14±2(4)
spect-heart	77.5±7.5(2)	75.48±8.4(3)	77.61±7.7(2)	**79.42±1.7(1)**	72.02±8.9(4)	**79.45±1.8(1)**
spectf-heart	67.4±8.5(3)	69.46±7.8(2)	68.22±8.9(2)	**79.42±1.7(1)**	69.08±8.4(2)	**79.19±2.7(1)**
statlog-australian-credit	75.38±5.7(3)	77.49±4.9(2)	77.72±5.7(2)	76.87±6.5(2)	74.06±6(4)	**80.64±5.5(1)**
statlog-german-credit	65.47±4.9(4)	68.1±4(3)	66.23±4.4(4)	**70.91±3(1)**	65.29±4.4(5)	69.69±3.2(2)
statlog-heart	74.56±7.6(3)	76.37±7.6(2)	75.78±7.5(2)	**78.78±6.8(1)**	73.07±7.8(3)	76.37±8.8(2)
statlog-vehicle	66.22±4.3(2)	**68.18±4.8(1)**	**67.4±4.4(1)**	51.41±8.5(4)	65.54±4.1(2)	64.01±4.5(3)
teaching-assistant	**49.92±12(1)**	40.86±11(4)	45.46±13(3)	40.72±12(4)	**52.08±12(1)**	49.75±11(2)
thyroid-disease	87.99±1.3(3)	90.72±2.3(2)	90.65±1.5(2)	**93.13±7.1(1)**	87.2±1.7(3)	88.3±10(3)
vote	85.24±9.1(3)	**90.46±7.2(1)**	**90.72±6.6(1)**	**89.26±7.1(1)**	84.81±9.1(3)	87.82±8.9(2)
wine	**93.08±6.5(1)**	**93.37±5.9(1)**	**93.54±5.7(1)**	**93.29±6.7(1)**	91.66±6.4(2)	89.93±7.2(3)
zoo	**53.98±13(1)**	44.97±13(3)	49.33±13(2)	39.55±5.8(4)	**52.77±14(1)**	**52.35±13(1)**
Mean Rank	2.178±0.16	1.889±0.14	1.778±0.12	2.422±0.23	2.511±0.2	2.289±0.17
Wins[unique]	16[0]	19[3]	19[1]	20[4]	14[2]	12[1]

Table 6. Comparison of neural networks with kernels from EncLen prototype selection with LDA, ELM, RBFg, kNN, L-SVM and SVM.

	EncLen-NN	LDA	ELM	RBFg	kNN	L-SVM	SVM
arrhythmia	34±9.6(4)	**53±20(1)**	26.1±17(5)	44.4±16(3)	**52.4±16(1)**	49.7±20(2)	0±0(6)
autos	75.1±12(2)	64.1±10(4)	67.2±9.6(3)	**81.7±11(1)**	62.6±12(4)	53±11(6)	57±12(5)
balance-scale	90.2±1.9(2)	86.6±2.7(5)	86.6±2.7(5)	**90.8±1.9(1)**	87.6±2.9(4)	84.5±3.1(6)	89.6±2(3)
blood-transfusion	**79.4±3.7(1)**	77.3±1.9(2)	77±2.1(2)	**79.5±3.7(1)**	76.3±4.2(3)	76.1±0.62(4)	76.8±2(3)
breast-cancer-diagnostic	96.3±2.6(2)	95.7±2.7(3)	94.9±2.9(4)	**97.4±2.2(1)**	**97±2.1(1)**	**97.3±2.2(1)**	96.2±2.5(2)
breast-cancer-original	96.8±2.1(2)	96±2(4)	96.3±2(3)	96.1±2.2(3)	96.7±1.9(2)	96.6±2(2)	**97±2.1(1)**
breast-cancer-prognostic	78.3±7.2(2)	**80±8.1(1)**	78.3±8.6(2)	72.7±8.9(4)	76.2±6.3(3)	**80.5±8.3(1)**	76.6±3.7(2)
breast-tissue	60.3±11(3)	66.2±13(2)	**68±12(1)**	54.6±16(4)	65.9±13(2)	43.6±8.5(5)	42.3±8.4(6)
car-evaluation	**94.4±1.7(1)**	84.2±2(5)	84.2±2(5)	92.4±1.7(3)	93.1±1.4(2)	82.2±3.5(6)	88±2(4)
cardiotocography-1	**84.2±2.2(1)**	66.4±2.8(6)	67.2±3.1(5)	80.9±2.4(2)	75.1±2.7(3)	58.2±2.7(7)	70.5±2.8(4)
cardiotocography-2	**92.6±1.9(1)**	86.5±1.8(7)	86.8±2(6)	91.4±1.9(2)	90.8±1.8(3)	87.4±1.9(5)	90.4±1.8(4)
chess-rook-vs-pawn	**98.2±0.74(1)**	94.1±1.4(5)	94±1.5(5)	95±1.2(3)	94.6±1.2(4)	96.8±0.98(2)	**98.3±0.76(1)**
cmc	48.4±4.3(3)	50.4±3.6(2)	50.4±3.9(2)	**53.4±4.1(1)**	46.8±4(4)	18.7±2.8(6)	30.6±3(5)
congressional-voting	95.3±4.8(3)	**97±3.6(1)**	**97±3.6(1)**	95.3±4.4(3)	92.1±5.1(4)	95.4±4.7(3)	96.3±3.9(2)
connectionist-bench-sonar	**83.6±6.6(1)**	75.2±9.7(4)	74.1±10(4)	**84.9±7.5(1)**	81.3±7.6(2)	74.6±9(4)	78.4±6.9(3)
connectionist-bench-vowel	**96±2.9(1)**	47.6±5.5(5)	47.7±5.5(5)	95.4±3.2(2)	93.4±3.4(3)	25.7±4.1(6)	60.9±4.9(4)
cylinder-bands	68±5(3)	**74.5±7.1(1)**	64.5±8.1(5)	70.3±5.9(2)	62±8(6)	**75.1±6.9(1)**	66.7±3(4)
dermatology	**95.5±3(1)**	95±3.4(2)	**95±3.5(1)**	**95.7±3(1)**	92.5±3.6(4)	93.4±3.9(3)	86.7±4.9(5)
ecoli	**86.3±5.1(1)**	84.8±5.1(2)	84.8±5.1(2)	**86.1±5.2(1)**	**85.6±4.7(1)**	76.1±6.2(4)	83.1±5.3(3)
glass	**65.3±9.3(1)**	60.8±9.6(3)	62.1±9.7(2)	**65±9.2(1)**	**65.8±8(1)**	36.4±7(5)	56.8±7.9(4)
habermans-survival	73.1±5.9(2)	**74.2±4.2(1)**	**74.2±4.2(1)**	**73.6±5.7(1)**	71.1±6.5(4)	72.6±2.5(3)	73.4±3.8(2)
hepatitis	85.4±11(2)	83.1±11(3)	83.1±11(3)	**89.9±10(1)**	87±11(2)	81.6±10(3)	**88±8.3(1)**
ionosphere	92.7±4.3(2)	86.4±4.1(4)	86.4±4.6(4)	93.2±3.8(2)	84.5±4.6(5)	88.4±4.6(3)	**94.7±3.6(1)**
iris	**96.5±4.7(1)**	83±8.2(3)	83±8.2(3)	95±6.5(2)	95±5.6(2)	78.1±8.6(4)	**96.2±5.3(1)**
libras-movement	**86.1±5.1(1)**	57.7±7.6(5)	63.1±7.3(4)	84.4±6(2)	75.8±5.8(3)	49.5±6.4(6)	46.7±7.6(7)
liver-disorders	67.3±8.5(2)	68.9±6.9(2)	69±7(2)	67.8±7(2)	61.8±8.1(3)	69.1±7.3(2)	**71±7.2(1)**
lymph	**85.1±8.6(1)**	**83.7±8.7(1)**	**83.8±8.8(1)**	82.1±9.7(2)	80.1±9.7(2)	80.4±9.3(2)	79±9.4(3)
monks-problems-1	99.9±0.44(2)	74.6±4.5(4)	74.6±4.5(4)	99.9±0.35(2)	99.6±0.95(3)	74.6±4.5(4)	**100±0(1)**
monks-problems-2	60.4±7(3)	63.1±2.9(2)	63.1±2.9(2)	62.2±7.3(2)	54.5±5.6(4)	**65.7±0.79(1)**	60.5±4.2(3)
monks-problems-3	98.3±2(3)	96.4±2.5(4)	96.4±2.5(4)	98.8±1.5(2)	**98.9±1.5(1)**	**98.9±1.5(1)**	**98.9±1.5(1)**
parkinsons	89.5±6.8(2)	88.6±6.9(2)	88.4±7(2)	**92.1±6.4(1)**	**91.3±6.3(1)**	86.9±7.5(3)	89±5.7(2)
pima-indians-diabetes	73.3±4.9(3)	**77.3±4.6(1)**	**77.3±4.5(1)**	73.2±4.6(3)	74±4.8(3)	**77±4.5(1)**	76.1±4.6(2)
sonar	**83.6±6.6(1)**	75.2±9.7(4)	74.1±10(4)	**84.9±7.5(1)**	81.3±7.6(2)	74.6±9(4)	78.4±6.9(3)
spambase	91.1±1.3(3)	88.7±1.4(6)	89.9±1.4(5)	90.6±1.2(4)	90.9±1.4(3)	**92.9±1.1(1)**	91.6±1.4(2)
spect-heart	**82.7±5.7(1)**	**83.4±5.3(1)**	**83.2±5.5(1)**	82.1±6.9(2)	81.8±6.6(2)	81.7±6.1(2)	82.4±6.1(2)
spectf-heart	**79.8±7(1)**	77.2±5.8(2)	76.9±7.3(2)	**79±7.7(1)**	72.8±6.5(3)	**79.1±7.5(1)**	78±4.4(2)
statlog-australian-credit	84.8±4.3(2)	**85.6±4.1(1)**	85.1±4.3(2)	84.8±4.7(2)	79.6±5.4(4)	84.8±4(2)	82.9±4.2(3)
statlog-german-credit	74.7±3.8(3)	**76.9±3.8(1)**	76.5±3.9(2)	75.2±4(3)	72.4±4(4)	**76.6±3.9(1)**	75.3±3.3(3)
statlog-heart	82.6±7.6(2)	**84.3±7(1)**	**84.3±7(1)**	76.4±9.1(3)	82.1±7.1(2)	**83.7±7.1(1)**	82.9±6.9(2)
statlog-vehicle	**83.2±4.1(1)**	75.6±4.2(3)	77.1±4.2(2)	**83.3±3.8(1)**	73±4.1(4)	68.2±4.5(5)	65.8±3.8(6)
teaching-assistant	56.5±12(2)	**59.5±12(1)**	**60.1±12(1)**	**61.3±13(1)**	42.4±12(4)	53.9±11(3)	39.7±11(5)
thyroid-disease	**96±0.54(1)**	93.3±0.29(6)	93.6±0.26(5)	95.5±0.56(2)	94.9±0.46(3)	93.7±0.35(4)	95.4±0.41(2)
vote	96.3±3.9(2)	**97±3.1(1)**	**97±3.1(1)**	94.3±4(3)	92.1±5.4(4)	**96.9±3.2(1)**	**96.9±3.1(1)**
wine	97.7±3.3(2)	**98.9±2.4(1)**	**98.9±2.4(1)**	93.7±8(4)	96.7±3.7(3)	96.4±3.8(3)	**98.3±2.8(1)**
zoo	70.5±13(3)	**94.8±5.7(1)**	92.8±7.3(2)	71.6±13(3)	40.3±9.4(4)	**93.8±5.9(1)**	35.2±12(5)
Mean Rank	1.87±0.13	2.8±0.27	2.84±0.24	2.04±0.15	2.93±0.18	3.13±0.28	2.96±0.25
Wins[unique]	18[6]	15[1]	11[1]	16[3]	6[0]	12[2]	10[4]

Table 7. Comparison of neural networks with kernels from DROP2 prototype selection with LDA, ELM, RBFg, kNN, L-SVM and SVM.

	Drop2-NN	LDA	ELM	RBFg	kNN	L-SVM	SVM
arrhythmia	32±8.9(4)	**53±20(1)**	26.1±17(5)	44.4±16(3)	**52.4±16(1)**	49.7±20(2)	0±0(6)
autos	72.2±12(2)	64.1±10(4)	67.2±9.6(3)	**81.7±11(1)**	62.6±12(4)	53±11(6)	57±12(5)
balance-scale	**90.8±1.9(1)**	86.6±2.7(4)	86.6±2.7(4)	**90.8±1.9(1)**	87.6±2.9(3)	84.5±3.1(5)	89.6±2(2)
blood-transfusion	79.1±3.6(2)	77.3±1.9(3)	77±2.1(3)	**79.5±3.7(1)**	76.3±4.2(4)	76.1±0.62(5)	76.8±2(4)
breast-cancer-diagnostic	96.6±2.4(2)	95.7±2.7(3)	94.9±2.9(4)	**97.4±2.2(1)**	**97±2.1(1)**	**97.3±2.2(1)**	96.2±2.5(3)
breast-cancer-original	96.8±2(2)	96±2(4)	96.3±2(3)	96.1±2.2(3)	96.7±1.9(2)	96.6±2(2)	**97±2.1(1)**
breast-cancer-prognostic	77.4±7(2)	**80±8.1(1)**	78.3±8.6(2)	72.7±8.9(4)	76.2±6.3(3)	**80.5±8.3(1)**	76.6±3.7(2)
breast-tissue	61.7±14(3)	66.2±13(2)	**68±12(1)**	54.6±16(4)	65.9±13(2)	43.6±8.5(5)	42.3±8.4(6)
car-evaluation	90.9±2.2(3)	84.2±2(5)	84.2±2(5)	92.4±1.7(2)	**93.1±1.4(1)**	82.2±3.5(6)	88±2(4)
cardiotocography-1	**84.5±2.2(1)**	66.4±2.8(6)	67.2±3.1(5)	80.9±2.4(2)	75.1±2.7(3)	58.2±2.7(7)	70.5±2.8(4)
cardiotocography-2	**92.7±1.6(1)**	86.5±1.8(7)	86.8±2(6)	91.4±1.9(2)	90.8±1.8(3)	87.4±1.9(5)	90.4±1.8(4)
chess-rook-vs-pawn	**98.9±0.64(1)**	94.1±1.4(6)	94±1.5(6)	95±1.2(4)	94.6±1.2(5)	96.8±0.98(3)	98.3±0.76(2)
cmc	49±4.3(3)	50.4±3.6(2)	50.4±3.9(2)	**53.4±4.1(1)**	46.8±4(4)	18.7±2.8(6)	30.6±3(5)
congressional-voting	96.5±3.7(2)	**97±3.6(1)**	**97±3.6(1)**	95.3±4.4(3)	92.1±5.1(4)	95.4±4.7(3)	96.3±3.9(2)
connectionist-bench-sonar	**84.5±6.6(1)**	75.2±9.7(4)	74.1±10(4)	**84.9±7.5(1)**	81.3±7.6(2)	74.6±9(4)	78.4±6.9(3)
connectionist-bench-vowel	**96.7±2.5(1)**	47.6±5.5(5)	47.7±5.5(5)	95.4±3.2(2)	93.4±3.4(3)	25.7±4.1(6)	60.9±4.9(4)
cylinder-bands	68.9±4.4(3)	**74.5±7.1(1)**	64.5±8.1(5)	70.3±5.9(2)	62±8(6)	**75.1±6.9(1)**	66.7±3(4)
dermatology	**95.7±3.1(1)**	95±3.4(2)	95±3.5(2)	**95.7±3(1)**	92.5±3.6(4)	93.4±3.9(3)	86.7±4.9(5)
ecoli	**86.5±5.4(1)**	84.8±5.1(2)	84.8±5.1(2)	**86.1±5.2(1)**	85.6±4.7(2)	76.1±6.2(4)	83.1±5.3(3)
glass	**66±9.8(1)**	60.8±9.6(3)	62.1±9.7(2)	**65±9.2(1)**	**65.8±8(1)**	36.4±7(5)	56.8±7.9(4)
habermans-survival	**74±5.5(1)**	**74.2±4.2(1)**	**74.2±4.2(1)**	**73.6±5.7(1)**	71.1±6.5(4)	72.6±2.5(3)	73.4±3.8(2)
hepatitis	84.4±12(3)	83.1±11(3)	83.1±11(3)	**89.9±10(1)**	87±11(2)	81.6±10(4)	**88±8.3(1)**
ionosphere	93.2±3.9(2)	86.4±4.1(4)	86.4±4.6(4)	93.2±3.8(2)	84.5±4.6(5)	88.4±4.6(3)	**94.7±3.6(1)**
iris	**96.9±4.2(1)**	83±8.2(3)	83±8.2(3)	95±6.5(2)	95±5.6(2)	78.1±8.6(4)	**96.2±5.3(1)**
libras-movement	**86±5.7(1)**	57.7±7.6(5)	63.1±7.3(4)	84.4±6(2)	75.8±5.8(3)	49.5±6.4(6)	46.7±7.6(7)
liver-disorders	68.4±7.2(2)	68.9±6.9(2)	69±7(2)	67.8±7(2)	61.8±8.1(3)	69.1±7.3(2)	**71±7.2(1)**
lymph	**84.3±8.9(1)**	**83.7±8.7(1)**	**83.8±8.8(1)**	82.1±9.7(2)	80.1±9.7(2)	80.4±9.3(2)	79±9.4(3)
monks-problems-1	99.9±0.69(2)	74.6±4.5(4)	74.6±4.5(4)	99.9±0.35(2)	99.6±0.95(3)	74.6±4.5(4)	**100±0(1)**
monks-problems-2	59.5±6.9(3)	63.1±2.9(2)	63.1±2.9(2)	62.2±7.3(2)	54.5±5.6(4)	**65.7±0.79(1)**	60.5±4.2(3)
monks-problems-3	98.8±1.6(2)	96.4±2.5(3)	96.4±2.5(3)	98.8±1.5(2)	**98.9±1.5(1)**	**98.9±1.5(1)**	**98.9±1.5(1)**
parkinsons	89.3±6.3(2)	88.6±6.9(2)	88.4±7(2)	**92.1±6.4(1)**	**91.3±6.3(1)**	86.9±7.5(3)	89±5.7(2)
pima-indians-diabetes	72.5±4.6(4)	**77.3±4.6(1)**	**77.3±4.5(1)**	73.2±4.6(3)	74±4.8(3)	**77±4.5(1)**	76.1±4.6(2)
sonar	**84.5±6.6(1)**	75.2±9.7(4)	74.1±10(4)	**84.9±7.5(1)**	81.3±7.6(2)	74.6±9(4)	78.4±6.9(3)
spambase	91.2±1.2(3)	88.7±1.4(7)	89.9±1.4(6)	90.6±1.2(5)	90.9±1.4(4)	**92.9±1.1(1)**	91.6±1.4(2)
spect-heart	**82.8±6.1(1)**	**83.4±5.3(1)**	**83.2±5.5(1)**	82.1±6.9(2)	81.8±6.6(2)	81.7±6.1(2)	82.4±6.1(2)
spectf-heart	**79.9±6.8(1)**	77.2±5.8(2)	76.9±7.3(2)	**79±7.7(1)**	72.8±6.5(3)	**79.1±7.5(1)**	78±4.4(2)
statlog-australian-credit	84.2±4(3)	**85.6±4.1(1)**	85.1±4.3(2)	84.8±4.7(2)	79.6±5.4(5)	84.8±4(2)	82.9±4.2(4)
statlog-german-credit	74.2±4.3(4)	**76.9±3.8(1)**	76.5±3.9(2)	75.2±4(3)	72.4±4(5)	**76.6±3.9(1)**	75.3±3.3(3)
statlog-heart	81±7.4(3)	**84.3±7(1)**	**84.3±7(1)**	76.4±9.1(4)	82.1±7.1(2)	**83.7±7.1(1)**	82.9±6.9(2)
statlog-vehicle	**83.4±3.8(1)**	75.6±4.2(3)	77.1±4.2(2)	**83.3±3.8(1)**	73±4.1(4)	68.2±4.5(5)	65.8±3.8(6)
teaching-assistant	51±11(3)	**59.5±12(1)**	**60.1±12(1)**	**61.3±13(1)**	42.4±12(4)	53.9±11(2)	39.7±11(5)
thyroid-disease	**96.2±0.51(1)**	93.3±0.29(6)	93.6±0.26(5)	95.5±0.56(2)	94.9±0.46(3)	93.7±0.35(4)	95.4±0.41(2)
vote	**96.8±3.2(1)**	**97±3.1(1)**	**97±3.1(1)**	94.3±4(2)	92.1±5.4(3)	**96.9±3.2(1)**	**96.9±3.1(1)**
wine	97.8±3.5(2)	**98.9±2.4(1)**	**98.9±2.4(1)**	93.7±8(4)	96.7±3.7(3)	96.4±3.8(3)	**98.3±2.8(1)**
zoo	61.8±12(4)	**94.8±5.7(1)**	92.8±7.3(2)	71.6±13(3)	40.3±9.4(5)	**93.8±5.9(1)**	35.2±12(6)
Mean Rank	1.98±0.15	2.82±0.27	2.89±0.24	2.07±0.16	3.02±0.19	3.16±0.27	3.04±0.25
Wins[unique]	19[6]	15[1]	10[1]	16[3]	6[1]	12[2]	9[4]

example, via inner CV learning) estimation of the number of kernels.

Additionally, the complexity of the new algorithm is $O(ml^2 + m^3n)$ and $\Omega(ml^2 + m^2n)$; however, we are working on a new version of DROP algorithms, whose complexity should be reduced from $O(m^3n)$ to an estimated complexity around $O(nm \log_2 m)$, whereupon this algorithm will be useful even for huge datasets.

References

Aha, D.W., Kibler, D. and Albert, M.K. (1991). Instance-based learning algorithm, *Machine Learning* **6**(1): 37–66.

Angiulli, F. (2007). Fast nearest neighbor condensation for large data sets classification, *IEEE Transactions on Knowledge and Data Engineering* **19**(11): 1450–1464.

Barandela, R., Ferri, F. and Sanchez, J. (2005). Decision boundary preserving prototype selection for nearest neighbor classification, *International Journal of Pattern Recognition and Artificial Intelligence* **19**(6): 787–806.

Boser, B.E., Guyon, I.M. and Vapnik, V. (1992). A training algorithm for optimal margin classifiers, *in* D. Haussler (Ed.), *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, ACM Press, New York, NY, pp. 144–152.

Brighton, H. and Mellish, C. (2002). Advances in instance selection for instance-based learning algorithms, *Data Mining and Knowledge Discovery* **6**(2): 153–172.

Brodley, C. (1995). Recursive automatic bias selection for classifier construction, *Machine Learning* **20**(1/2): 63–94.

Broomhead, D.S. and Lowe, D. (1988). Multivariable functional interpolation and adaptive networks, *Complex Systems* **2**(3): 321–355.

Cameron-Jones, R.M. (1995). Instance selection by encoding length heuristic with random mutation hill climbing, *Proceedings of the 8th Australian Joint Conference on Artificial Intelligence, Canberra, Australia*, pp. 99–106.

Cano, J.-R., Herrera, F. and Lozano, M. (2003). Using evolutionary algorithms as instance selection for data reduction in KDD: An experimental study, *IEEE Transactions on Evolutionary Computation* **7**(6): 561–575.

Chamara, L.L., Kasun, Zhou, H. and Huang, G.-B. (2013). Representational learning with ELMs for big data, *IEEE Intelligent Systems* **28**(6): 31–34.

Devi, V. and Murty, M. (2002). An incremental prototype set building technique, *Pattern Recognition* **35**(2): 505–513.

Garcia, S., Cano, J. and Herrera, F. (2008). A memetic algorithm for evolutionary prototype selection: A scaling up approach, *Pattern Recognition* **41**(8): 2693–2709.

Garcia, S., Derrac, J., Cano, J. and Herrera, F. (2012). Prototype selection for nearest neighbor classification: Taxonomy and empirical study, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **34**(3): 417–435.

Gates, G. (1972). The reduced nearest neighbor rule, *IEEE Transactions on Information Theory* **18**(3): 431–433.

Górecki, T. and Łuczak, M. (2013). Linear discriminant analysis with a generalization of the Moore–Penrose pseudoinverse, *International Journal of Applied Mathematics and Computer Science* **23**(2): 463–471, DOI: 10.2478/amcs-2013-0035.

Grochowski, M. and Jankowski, N. (2004). Comparison of instances selection algorithms. I: Results and comments, *in* L. Rutkowski *et al.* (Eds.), *Artificial Intelligence and Soft Computing*, Lecture Notes in Computer Science, Vol. 3070, Springer-Verlag, Berlin/Heidelberg, pp. 580–585.

Hart, P.E. (1968). The condensed nearest neighbor rule, *IEEE Transactions on Information Theory* **14**(3): 515–516.

Hattori, K. and Takahashi, M. (2000). A new edited k-nearest neighbor rule in the pattern classification problem, *Pattern Recognition* **33**(3): 521–528.

Huang, G.-B., Zhu, Q.-Y. and Siew, C.-K. (2004). Extreme learning machine: A new learning scheme of feedforward neural networks, *International Joint Conference on Neural Networks, Budapest, Hungary*, pp. 985–990.

Huang, G.-B., Zhu, Q.-Y. and Siew, C.-K. (2006). Extreme learning machine: Theory and applications, *Neurocomputing* **70**(1–3): 489–501.

Jankowski, N. and Grochowski, M. (2004). Comparison of instances selection algorithms. II: Algorithms survey, *in* L. Rutkowski *et al.* (Eds.), *Artificial Intelligence and Soft Computing*, Lecture Notes in Computer Science, Vol. 3070, Springer-Verlag, Berlin/Heidelberg, pp. 598–603.

Kuncheva, L. (1995). Editing for the k-nearest neighbors rule by a genetic algorithm, *Pattern Recognition Letters* **16**(8): 809–814.

Lozano, M., Sanchez, J. and Pla, F. (2003). Using the geometrical distribution of prototypes for training set condensing, *Conference of the Spanish Association for Artificial Intelligence, San Sebastian, Spain*, pp. 618–627.

Marchiori, E. (2008). Hit miss networks with applications to instance selection, *Journal of Transactions on Machine Learning Research* **9**: 997–1017.

Marchiori, E. (2010). Class conditional nearest neighbor for large margin instance selection, *IEEE Transactions Pattern Analysis and Machine Intelligence* **32**(2): 364–370.

Merz, C.J. and Murphy, P.M. (1998). *UCI Repository of Machine Learning Databases*, `https://archive.ics.uci.edu/ml/index.php`.

Riquelme, J., Aguilar-Ruiz, J. and Toro, M. (2003). Finding representative patterns with ordered projections, *Pattern Recognition* **36**(4): 1009–1018.

Sanchez, J., Pla, F. and Ferri, F. (1997). Prototype selection for the nearest neighbor rule through proximity graphs, *Pattern Recognition Letters* **18**(6): 507–513.

Schölkopf, B., Sung, K., Burges, C., Girosi, F., Niyogi, P., Poggio, T. and Vapnik, V. (1996). Comparing support vector machines with Gaussian kernels to radial basis function classifiers, *Technical Report AI, Memo No 1599, CBCL Paper No 142*, MIT, Cambridge, MA.

Schölkopf, B., Sung, K.-K. and Burges, C. (1997). Comparing support vector machines with Gaussian kernels to radial basis function classifiers, *IEEE Transactions on Signal Processing* **45**(11).

Schwenker, F., Kestler, H.A. and Palm, G. (2001). Three learning phases for radial-basis-function networks, *Neural Networks* **14**(4–5): 439–458.

Skalak, D.B. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms, *International Conference on Machine Learning, New Brunswick, NJ, USA*, pp. 293–301.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, NY.

Wilson, D. (1972). Asymptotic properties of nearest neighbor rules using edited data, *IEEE Transactions on Systems, Man, and Cybernetics* **2**(3): 408–421.

Wilson, D.R. and Martinez, T.R. (2000). Reduction techniques for instance-based learning algorithms, *Machine Learning* **38**(3): 257–286.

Woźniak, M. and Krawczyk, B. (2012). Combined classifier based on feature space partitioning, *International Journal of Applied Mathematics and Computer Science* **22**(4): 855–866, DOI: 10.2478/v10006-012-0063-0.

Yousef, R. and el Hindi, K. (2005). Training radial basis function networks using reduced sets as center points, *International Journal of Information Technology* **2**(1): 21–35.

Zhao, K., Zhou, S., Guan, J. and Zhou, A. (2003). C-pruner: An improved instance pruning algorithm, *2nd International Conference on Machine Learning and Cybernetics, Xi'an, China*, pp. 94–99.

Norbert Jankowski has been an associate professor at Nicolaus Copernicus University in Toruń, Poland, since 2012. He obtained his DSc in computer science at the Systems Research Institute, Polish Academy of Sciences, and his PhD in computer science at the Institute of Biocybernetic and Biomedical Engineering, Polish Academy of Sciences. He also holds an MSc in computer science from the Institute of Computer Science, University of Wrocław, Poland. His main research areas include computational intelligence, meta-learning, machine learning, neural networks, data mining, pattern recognition, complexity, algorithms and data structures. He is the author of two books: *Ontogenic Neural Networks* (2003) and *Meta-Learning in Computational Intelligence* (2011), as well as 77 papers.

Received: 27 December 2017
Revised: 22 May 2018
Re-revised: 27 July 2018
Accepted: 12 August 2018