Neural Networks 24 (2011) 824–830

Contents lists available at SciVerse ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2011 Special Issue

LVQ algorithm with instance weighting for generation of prototype-based rules

Marcin Blachnik [a,*], Włodzisław Duch [b,c]

[a] Department of Management and Informatics, Silesian University of Technology, Katowice, Krasinskiego 8, Poland

[b] Department of Informatics, Nicolaus Copernicus University, Poland

[c] School of Computer Science, Nanyang Technological University, Singapore

ARTICLE INFO

Keywords:
Learning Vector Quantization
Knowledge-based clustering
Prototype-based rules
Similarity-based methods

ABSTRACT

Crisp and fuzzy-logic rules are used for comprehensible representation of data, but rules based on similarity to prototypes are equally useful and much less known. Similarity-based methods belong to the most accurate data mining approaches. A large group of such methods is based on instance selection and optimization, with the Learning Vector Quantization (LVQ) algorithm being a prominent example. Accuracy of LVQ depends highly on proper initialization of prototypes and the optimization mechanism. This paper introduces prototype initialization based on context dependent clustering and modification of the LVQ cost function that utilizes additional information about class-dependent distribution of training vectors. This approach is illustrated on several benchmark datasets, finding simple and accurate models of data in the form of prototype-based rules.

1. Introduction

One of the problems with applications of neural networks in safety-critical situations is the "black-box nature" of its solutions. The best explanation of the data structures obviously depends on the particular problem, on standards commonly accepted in a given field. Most research on understanding data structures using neural networks has focused on extracting propositional logical rules (Duch, Adamczak, & Grąbczewski, 2001; Duch, Setiono, & Zurada, 2004), but such rules, in crisp or fuzzy form, are not always the best way to understand the data. Expressive power of crisp or fuzzy rules (C-rules and F-rules) has serious limitations. The "majority voting" concept expressed by C-rules or F-rules requires an exponential number of conditions, but it is easily implemented in a general, weighted form by the M-of-N threshold rules, provided by perceptrons. Prototype-based rules use sophisticated measures of similarity that may capture some intuitive forms of evaluation, as used for example in medical diagnostics. Human categorization of natural objects is largely based on memorization of numerous examples, replaced by prototypes that enable generalization of these examples (Roth & Bruce, 1995). In real life "intuitive understanding" reflecting the experience of an expert is used more often than propositional logics and works well in rather simple situations and in abstract sciences.

While the literature on fuzzy rules is huge, P-rules are much less popular and there seems to be some prejudice against their comprehensibility. P-rules may represent typical or exceptional cases, covering many examples, or specific cases close to the decision borders, with similarity functions using only those features that are important for making discriminations locally around each prototype. Numerous arguments in favor of P-rules as compared to C-rules and F-rules have been given in Blachnik and Duch (2008) and Duch and Grudziński (2001). They include more flexible decision borders, ability to handle exceptions in an easy way, stability of sets of rules, connection with basis set expansion regularization networks (Lowe, 1995; Poggio & Girosi, 1990), vector quantization and self-organized networks (Kohonen, 1984). P-rules provide the most general form of knowledge representation: F-rules are equivalent to P-rules with separable similarity metrics (Duch & Blachnik, 2004) and C-rules are a special case of F-rules with Chebychev distance function. P-rules may also represent M-of-N rules in a natural way using prototype threshold rules (Blachnik & Duch, 2006). Two general types of P-rules are possible, with decision borders based either on the nearest prototypes, or on similarity thresholds for a given prototype. For example, a single prototype threshold rule gives over 97.5% accuracy on a well known Wisconsin Breast Cancer (WBC) benchmark dataset (Grąbczewski & Duch, 2002), selecting important features to distinguish a "worst-case" malignant prototype from the benign cases. This is the simplest and most accurate description of the WBC data structure found so far.

Despite these arguments prototype-based rules are still much less popular than other forms of rules. The use of prototype-based rule systems with linear functions for control (Tang & Lawry, 2010) may use linguistic labels but it is simply a basis set expansion technique that does not lead to comprehensible description of

* Corresponding author.
E-mail address: marcin.blachnik@polsl.pl (M. Blachnik).

0893-6080/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2011.05.013

data. Similarity-Based Methods offer a rich framework (Duch, 2000; Duch, Adamczak, & Diercksen, 2000) for construction of such methods. Many methods used for data analysis may be easily accommodated in this framework, including neural networks (Duch, Adamczak, & Diercksen, 1999), probabilistic and fuzzy methods (Duch & Blachnik, 2004), kernel approaches and many other methods (Pękalska & Duin, 2005). Prototype-based rules also belong to this class of methods, specifically aiming to represent the knowledge hidden in the data in a comprehensible way. This is achieved by reduction of the number of prototype vectors (selection of prototypes), minimization of the number of features used to create the final model, and the use of simple similarity metrics.

The Learning Vector Quantization (LVQ) algorithm is usually presented in the neural network form, but it also belongs to the Similarity-Based Methods framework. Several LVQ algorithms became popular after the publication of Kohonen's book "Self-Organization and Associative Memory" (Kohonen, 1984). They are simple and usually quite accurate, using the nearest neighbor rule for classification. The LVQ decision model is therefore based on the similarity (or dissimilarity, i.e. distance) measure between the test object and the reference vectors, called the codebook vectors ($\mathbf{p}_i$), or prototypes. The output label of the test object is the same as the label of the nearest codebook:

$$\begin{aligned} k &= \arg\min_i \left(D\left(\mathbf{x}, \mathbf{p}_i\right)\right) \\ C(\mathbf{x}) &\leftarrow C(\mathbf{p}_k) \end{aligned} \tag{1}$$

where $D(\cdot, \cdot)$ is the distance function. The square of the distance $D(\cdot, \cdot)^2$ may also be used as it does not change the relative distance ordering and is computationally more efficient, $\mathbf{x}$ is the test object, and $C(\cdot)$ is the label or class indicator of a given object. Distance or similarity measures may be used in the decision rule, replacing arg min by the arg max.

The learning process of the LVQ network determines the optimal position of the prototypes. To reach this goal one has first to:

- define objective function;
- determine appropriate distance measure;
- construct optimization algorithm;
- select optimal number of codebook vectors;
- determine initial position of codebook vectors to avoid sub-optimal solutions.

Many variants of the LVQ algorithms have been created, depending on the specific solutions to the points listed above.

LVQ also has many different applications. It may determine positions of centers of radial basis functions in the radial basis function (RBF) neural network training (Schwenker, Kestler, & Palm, 2001). The centers of basis functions optimized by LVQ reflect real classification boundaries, not just grouping of the input vectors determined by clustering methods. LVQ is connected in many ways to kernel algorithms. Kernel methods have great generalization capabilities, but have also some drawbacks. For large datasets computational complexity is high (proportional to $O(n^2)$ for n samples) ending with a large number of potential support vectors. Several solutions of this problem exist (Lee & Huang, 2007), reducing the number of support vectors while trying to preserve original decision borders. Although reformulation of the objective function for SVM training is possible, a simpler solution is based on clustering or LVQ algorithm for the selection of good candidates for support vectors (Blachnik & Duch, 2008).

The LVQ approach can be used as a model for training neuro-fuzzy models. It is not hard to show (Kuncheva, 1996) that the LVQ neural network is equivalent to the Takagi–Sugeno fuzzy rule-based system with aggregation operator defined as max(·). LVQ networks have also been applied to preprocessing of images. A common representation of image properties is based on histogram features, derived from sampling patches of images and binning of intensity, color, texture, edge-orientation and other features. The most commonly used method to determine relevant samples is based on clustering, but LVQ optimization of the codebook vectors results in much higher discrimination of images based on their histograms (Blachnik & Laaksonen, 2008). The diverse range of applications mentioned above demonstrate the universality of the LVQ approach.

The concept of prototype-based rules has been proposed in Duch and Grudziński (2001). After selection of a small set of reference vectors $\mathbf{p}_i$ the nearest neighbor rule is used to associate them with class labels or any other output information:

$$\text{if } \mathbf{p} = \operatorname*{argmin}_{\mathbf{p}' \in \mathbf{P}} D(\mathbf{x}, \mathbf{p}') \text{ then } y = C(\mathbf{p}). \tag{2}$$

In more general form a fuzzy rule is associated with each prototype, and an aggregation operator is used to determine final decision. Distance functions are replaced by similarity estimation (for example activation of Gaussian nodes that compute distances $S(\mathbf{x}, \mathbf{p}) = \exp\left(-D(\mathbf{x}, \mathbf{p})^2\right)$) estimating similarity between the vector $\mathbf{x}$ and the prototype $\mathbf{p}$ in a fuzzy sense. Members of each pair – prototype and its corresponding class label – are related to each other by a single fuzzy rule:

(1) if $\mathbf{x} \circledast \mathbf{p}_1$ then $y_1 \circledast C(\mathbf{p}_1)$
(2) if $\mathbf{x} \circledast \mathbf{p}_1$ then $y_2 \circledast C(\mathbf{p}_2)$
...
(v) if $\mathbf{x} \circledast \mathbf{p}_v$ then $y_v \circledast C(\mathbf{p}_v)$

where $\circledast$ denotes the "*is similar to*" operation simply computed as $S(\mathbf{x}, \mathbf{p})$. The aggregation operator may implement a max function leading to the nearest prototype model or softmax function, summing the activity of all nodes associated with a given class and dividing it over total activity, but there are many other choices (Kuncheva, 2004).

The main objective of such an approach goes beyond mere classification or approximation ($y(\mathbf{x}; \mathbf{p}) = C(\mathbf{p})$ may be used for local approximation). Finding the nearest prototype is useful in many ways (Duch et al., 2000), providing information for case-based reasoning (Rissland, 2006). For example, in law, medical or machine diagnostics retrieving the most similar case may provide much more information than just simple diagnosis or prediction of expected outcomes. Comprehensibility of the P-rule systems depends on a significant reduction of the number of reference cases by the LVQ algorithm. Sophisticated similarity measures may capture intuitive evaluation of experts that could be hard to explain in the form of propositional rules.

The accuracy of weak classifiers is improved strongly using boosting techniques (Schapire & Singer, 1999) that introduce weights for training instances. This is a relatively unexplored aspect of the LVQ algorithms. Appropriate weights may guide the optimization of codebook positions, reducing the influence of outliers by increasing the importance of instances close to the border. The next section is dedicated to the problem of codebook vector initialization, including the use of context dependent clustering. In Section 3 some modifications of the LVQ cost function are introduced, Section 4 is devoted to the problem of instance weighting, and in Section 5 a weighted LVQ network is compared with the standard LVQ approach on a few datasets. The final Section 6 contains a brief discussion.

2. Initialization of LVQ codebook vectors

All neural network models require initialization of network parameters. The LVQ approach requires specification of initial

codebook positions in the feature space. To solve this problem many different solutions have been proposed. Random selection of the codebooks from the set of training instances is inefficient, leading to a large variance and low accuracy of models after training. Several methods used to reduce the number of reference vectors in the kNN algorithm may be employed for codebook initialization. Selection algorithms that work in top–down fashion include:

- *Edited Nearest Neighbor (ENN)* (Wilson, 1972) algorithm that removes from the training set instances that are incorrectly classified by kNN for $k \geq 3$;
- *DROP3* (Wilson & Martinez, 2000) removes instance $\mathbf{x}$ from the training set if it does not change the classification of instances for which $\mathbf{x}$ is one of the k nearest neighbors;
- *Iterative Case Filtering (ICF)* (Brighton & Mellish, 2002) starts from DROP3 and creates hyperspheres with instances from a single-class.

Other instance selection algorithms include:

- Bottom–up *Condensed Nearest Neighbor rule (CNN)* (Hart, 1968) starts with one vector per class, and adds training instances that are wrongly classified by the current reference set.
- Graph-based *Gabriel Editing (GE)* algorithm (Bhattacharya, Mukherjee, & Toussaint, 2005) uses a Gabriel graph to remove from the training dataset all instances from the same class as all their neighbors, leaving only those instances that define decision borders between different classes.
- Probability-density estimating algorithms, such as the *Edited Normalized RBF (NRBF)* (Jankowski & Grochowski, 2004) removes instances $\mathbf{x}$ inconsistent with probability density estimation around point $\mathbf{x}$.

For an in-depth review of these algorithms see (Grochowski & Jankowski, 2004; Jankowski & Grochowski, 2004; Wilson & Martinez, 2000). Codebook initialization methods based on clustering have lower computational complexity in comparison to the instance selection methods, however they are usually less accurate. To improve accuracy context dependent clustering is used instead of classical clustering algorithms. Such algorithms are guided by instance weights that exert local influence on the clustering process in the desired area. In classification problems the focus should be on precise description of the class border areas, combined with general prototype-based rules that cover large clusters of pure class data.

Context dependent clustering uses context information to guide the clustering process. In the simplest case context is described by an external variable that estimates the importance of each vector described by an instance weight. Conditional Fuzzy C-Means (CFCM) (Pedrycz, 2005) is an extension of the FCM clustering algorithm, with additional variable $\mathbf{y}$, defined for every vector $\mathbf{x}$, that helps to cluster related data. For every vector $\mathbf{x}_i$ the strength of its relation with the external variable y_i is defined by a function $f_i = \mu_A(\mathbf{y}) \in [0, 1]$, where $\mu_A(\mathbf{y})$ is interpreted as a membership function for some fuzzy set A, or simply as a weight that creates a clustering condition in context A.

FCM and CFCM are based on minimization of a cost function defined as:

$$J_m(\mathbf{U}, \mathbf{V}) = \sum_{k=1}^{v} \sum_{i=1}^{n} (u_{ki})^m \|\mathbf{x}_i - \mathbf{p}_k\|_A^2 \tag{3}$$

where v is the number of clusters centered at $\mathbf{p}_k$ (such centers will be used as reference vectors or the LVQ classifier), n is the number of vectors, $m > 1$ determines clustering fuzziness, and $\mathbf{U} = (u_{ki})$ is a $v \times n$ dimensional membership matrix with elements $u_{ki} \in [0, 1]$ defining the membership degree of vector $\mathbf{x}_i$ in cluster $\mathbf{p}_k$. The matrix $\mathbf{U}$ has to fulfill three conditions:

1° each vector x_i is a member of the k-th cluster to a degree between 0 and 1:

$$u_{ki} \in [0, 1]; \quad k = 1, \ldots, v; \; i = 1, \ldots, n \tag{4}$$

2° sum of the membership values of i-th vector x_i in all the clusters is equal to f_i

$$\sum_{k=1}^{v} u_{ki} = f_i; \quad i = 1, \ldots, n \tag{5}$$

3° clusters are not empty, and a single cluster does not cover the whole space:

$$\sum_{i=1}^{n} u_{ki} \in (0, n); \quad k = 1, \ldots, v. \tag{6}$$

Centers of the clusters are calculated as:

$$\mathbf{p}_k = \sum_{i=1}^{n} (u_{ki})^m x_i \Big/ \sum_{i=1}^{n} (u_{ki})^m; \quad k = 1, \ldots, v. \tag{7}$$

The partition matrix is then calculated using cluster centers, and the cost function (3) is minimized iteratively:

$$u_{ki} = f_i \Big/ \sum_{k=1}^{C} \left(\frac{\|x_i - \mathbf{p}_k\|}{\|x_i - \mathbf{p}_k\|} \right)^{2/(m-1)}; \quad i = 1, \ldots, n; \; k = 1, \ldots, v. \tag{8}$$

3. Introducing context in the LVQ algorithm

The cost function of the basic LVQ1 algorithm is defined as:

$$E(\mathbf{P}) = \frac{1}{2} \sum_{k=1}^{v} \sum_{i=1}^{n} \mathbf{1}(\mathbf{x}_i \in R_k) \, \mathbf{1}(C(\mathbf{x}_i) = C(\mathbf{p}_k)) \, \|\mathbf{x}_i - \mathbf{p}_k\|^2 - \frac{1}{2} \sum_{k=1}^{v} \sum_{i=1}^{n} \mathbf{1}(\mathbf{x}_i \in R_k) \, \mathbf{1}(C(\mathbf{x}_i) \neq C(\mathbf{p}_k)) \, \|\mathbf{x}_i - \mathbf{p}_k\|^2 \tag{9}$$

where $\mathbf{1}(L)$ is the switching function, it returns 1 when condition L is *true*, and 0 otherwise. $C(\mathbf{x})$ returns class label of vector $\mathbf{x}$, and R_k is the Voronoi area defined for prototype $\mathbf{p}_k$. This relation can also be interpreted as $\mathbf{x}_i \in R_k$ if $\mathbf{p}_k = \underset{l=1,\ldots,v}{\operatorname{argmin}} (D(\mathbf{x}_i, \mathbf{p}_l))$. This cost function is minimized in respect to the position of all prototypes $\mathbf{P}$. Determining the derivatives of that cost function leads to two formulas representing the iterative updates of prototypes positions:

$$\begin{aligned} \mathbf{p}_k &= \mathbf{p}_k + \alpha(j) \mathbf{1}(C(\mathbf{x}_i) = C(\mathbf{p}_k)) \, (\mathbf{x}_i - \mathbf{p}_k) \\ \mathbf{p}_k &= \mathbf{p}_k - \alpha(j) \mathbf{1}(C(\mathbf{x}_i) \neq C(\mathbf{p}_k)) \, (\mathbf{x}_i - \mathbf{p}_k) \end{aligned} \tag{10}$$

where $\alpha(j) \in [0, 1]$ is a monotonically decreasing step size function and j is the iteration number.

In the original LVQ algorithm only local information is taken into account, therefore it is very sensitive to the presence of outliers. The influence of outliers can be reduced by defining instance weights in a similar way as in the context dependent clustering. The $f(\mathbf{x}_i)$ factors should incorporate information about overall distribution of vectors from different classes, giving outliers and vectors that are far from decision border less influence on the value of the cost function. These factors are taken into account through reformulation of the objective function:

$$E(\mathbf{P}) = \frac{1}{2}\sum_{k=1}^{v}\sum_{i=1}^{n}\mathbf{1}(\mathbf{x}_i \in R_k)\,\mathbf{1}(C(\mathbf{x}_i) = C(\mathbf{p}_k))\,f(\mathbf{x}_i)\|\mathbf{x}_i - \mathbf{p}_k\|^2 - \frac{1}{2}\sum_{k=1}^{v}\sum_{i=1}^{n}\mathbf{1}(\mathbf{x}_i \in R_k) \times \mathbf{1}(C(\mathbf{x}_i) \neq C(\mathbf{p}_k))\,f(\mathbf{x}_i)\|\mathbf{x}_i - \mathbf{p}_k\|^2. \quad (11)$$

The cost function can be minimized by modifying the formula (10) for the adjustment of prototypes:

$$\begin{aligned}\mathbf{p}_k &= \mathbf{p}_k + \alpha(j)f(\mathbf{x}_i)\mathbf{1}(C(\mathbf{x}_i) = C(\mathbf{p}_k))\,(\mathbf{x}_i - \mathbf{p}_k)\\ \mathbf{p}_k &= \mathbf{p}_k - \alpha(j)f(\mathbf{x}_i)\mathbf{1}(C(\mathbf{x}_i) \neq C(\mathbf{p}_k))\,(\mathbf{x}_i - \mathbf{p}_k)\end{aligned} \quad (12)$$

where the context factor $f(\mathbf{x}_i)$ describes the class-dependent distribution of vectors around $\mathbf{x}_i$. The $f(\mathbf{x}_i)$ value can also be understood as an instance weight describing the importance of that instance during the training process.

The sketch of the weighted LVQ algorithm (WLVQ) is presented in Algorithm 1. Because the weights are calculated only at the beginning, computational complexity of the new algorithm is the sum of the complexity of the weight calculation part, and the basic LVQ algorithm $O(n \cdot v \cdot z)$, where z is the number of iterations, usually not exceeding the number of samples.

Algorithm 1 WLVQ algorithm

```
Require: T, P, α, z
  w ← determine_instance_weights(T)
  n ← instances_of(T);
  v ← instances_of(P);
  for k = 1 ... z do
    for i = 1 ... n do
      for j = 1 ... v do
        d_j ← D(T_i, P_j)
      end for
      i ← arg min_{j=1,...,v} (d_j)
      if C(T_i) == C(P_i) then
        P_i = P_i + α · w_i · (T_i − P_i)
      else
        P_i = P_i − α · w_i · (T_i − P_i)
      end if
    end for
    α = α/(1 + α)
  end for
  return P
```

4. Determining the instance weights

Defining appropriate instance weights should help to find the minimum number of LVQ prototypes providing a simple but accurate model of the data. Support vector machines work well because they focus on the border area, providing hyperplanes with wide margins and defining sharp decision borders. Support vectors in the kernel space act as prototypes $\mathbf{p}_i$, with kernel values $K(\mathbf{x}, \mathbf{p}_i)$ serving as a weighted distance from these prototypes. Therefore weights of the training instances should stress the importance of the border areas. There are many different solutions that can be used for that purpose, and in practice any classifier can be used to identify the instance weights. Here we shall investigate the use of two weighting algorithms, one based on the inter–intra class similarity function, and another on using the Edited Nearest Neighbor Rule (ENN) instance selection algorithm.

4.1. Inter–intra class similarity based context

To encourage a selection of prototypes that lie close to the decision border, instance weights are defined with the help of a coefficient w_k describing the distribution of vector positions around $\mathbf{x}_k$ within the same, and relatively to other classes. This is defined by the ratio of a within-class scatter to out-of-class scatter:

$$w_i = w(\mathbf{x}_i) = \frac{(n - n_c)}{n_c}\frac{\sum_{j,C(\mathbf{x}_i)=C(\mathbf{x}_j)}\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{\sum_{j,C(\mathbf{x}_i)\neq C(\mathbf{x}_j)}\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}. \quad (13)$$

Here $C(\mathbf{x}_i)$ denotes the class label of the vector $\mathbf{x}_i$, with n_c vectors in this class. The normalized w_i coefficients (are in the range $[0 \quad \cdots \quad 1]$)reach values that are close to 0 for vectors inside large homogenous clusters, and close to 1 if the vector $\mathbf{x}_i$ is near the vectors of the opposite classes and far from other vectors from the same class (for example, if it is an outlier). To avoid such situations weights should be centered around $\mu \in [0.6, 1]$, with Gaussian distribution defining the context factor:

$$f_i = f(\mathbf{x}_i) = \exp\left(-\gamma\,(w(\mathbf{x}_i) - \mu)^2\right). \quad (14)$$

The value of $\gamma \in [0.4, 0.8]$ is recommended, determined empirically for a wide range of datasets. The μ parameter controls where the prototypes will be placed; for small μ they are closer to the center of the cluster and for larger μ closer to the decision borders. The range in which they are sought is determined by the γ parameter. Because this algorithm requires calculation of the distances between all instances the computational complexity is quadratic in the number of samples $O\left(n^2\right)$. This function is used to determine the clustering context for the CFCM clustering in the weighted LVQ algorithm (WLVQ).

4.2. ENN based context

The Edited Nearest Neighbour (ENN) rule is based on the leave one out test. In the original algorithm all vectors that are incorrectly classified by the kNN classifier ($k \geq 3$) are marked for deletion. We have used here a modified version of this approach adopted to determine instance weights. This modified version does not remove any instances, but instead the prediction confidence is used as an instance weight. The diagram of the program is shown in Algorithm 2, where $\mathbf{T}$ denotes the training set, and con$(\cdot)$ is the confidence of the predicted class, equal to the fraction of k neighbors that belong to the majority class $C(\mathbf{x}_i)$. As in the previous methods weights w_i should be centered around μ with a Gaussian distribution.

Algorithm 2 Modified ENN algorithm

```
Require: T, k
  n ← instances_of(T);
  w_i ← 0;
  for i = 1 ... n do
    C(x_i) = kNN({T \ x_i}, x_i);
    w_i = con(C(x_i));
  end for
  for i = 1 ... n do
    f_i = exp(−γ (w_i − μ)^2)
  end for
  return f
```

The ENN algorithm can be used for simultaneous codebook initialization and instance weighting for LVQ training. Unfortunately weights determined by this algorithm are usually sensitive to outliers. Similar to the weighting procedure described in the previous

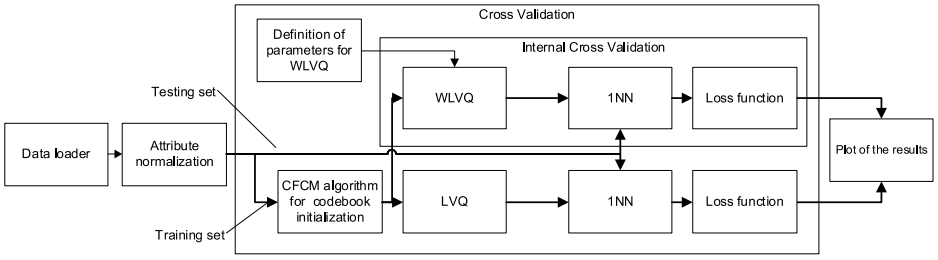

Fig. 1. Data flow diagram.

section this algorithm also requires calculation of the distances between all instances, so the computational complexity is of the order $O\left(n^2\right)$.

5. Illustrative calculation

In order to investigate properties of the algorithms described in previous sections, several tests have been done on seven benchmark datasets taken from the UCI repository (Asuncion & Newman, 2007) (Sonar, Wisconsin Breast Cancer (WBC), Spambase, Diabetes (Pima Indian), Heart disease (Cleveland), Ionosphere and Appendicitis). Three datasets are discussed in detail: Sonar, WBC, Spambase, and the best results for the remaining four are shown in Table 2. Table 3 also shows a comparison with other popular machine learning algorithms like the C4.5 decision tree, and SVM with linear and Gaussian kernel. Some basic characteristics of these datasets are provided in Table 1.

All experiments have been carried out in an identical manner using the 10-fold cross-validation procedure, with all algorithms embedded in the crossvalidation folds to achieve an unbiased estimation of generalization error. The diagram presenting data flow in computational experiments is presented in Fig. 1. In these experiments accuracy of the nearest-prototype classifier with prototypes obtained from the classical LVQ algorithm have been compared with those based on prototypes optimized using the WLVQ algorithm. In both cases prototypes/codebooks were initialized using the CFCM clustering algorithm. The cross validation process has been repeated two times, consequently the accuracy and standard deviation has been calculated from 20 different results. In the case of the WLVQ algorithm, optimal values of γ and μ were selected using a greedy search algorithm, for the Inte-Intra class similarity method from values of $\gamma = [0.3, 0.6, 0.9]$ and values of $\mu = [0.4, 0.6, 0.8, 1]$, and for the ENN method from values of $\gamma = [0.4, 0.6, 0.8]$ and 4 values of $\mu = [0, 0.1, 0.2, 0.3]$. The number of nearest neighbors in the ENN algorithm was set to $k = 10$.

Results are presented in Figs. 2–4. Each graph shows the average accuracy and variance as a function of the number of selected prototypes per class. In all plots a solid thin line represents accuracy of the reference LVQ algorithm and the bold dotted lines represent mean accuracy obtained with the WLVQ algorithm for weights determined by both types of algorithms.

For the *Sonar* dataset the significant difference between weighted and non-weighted algorithms vanishes for the higher number of prototypes per class (#codebooks $\geq$ 9). For both weighting functions the increase of the accuracy is over 10% for the small number of prototypes. This solution gives the most valuable P-rule, and thus it is important to maximize its accuracy. Using only 1 prototype per class adds about 13% accuracy, but is still worse on about 7% compared with the results of the one nearest neighbor classifier 2.

For the *WBC* dataset both weighting functions behave almost identically. One prototype per class is optimal, with ENN weights

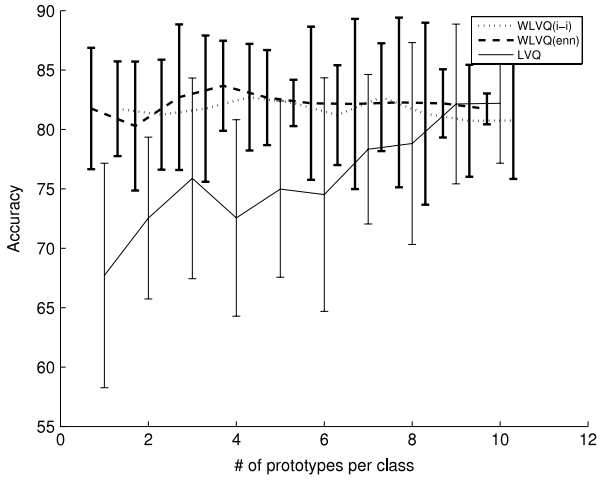

Fig. 2. Results for the *Sonar* data.

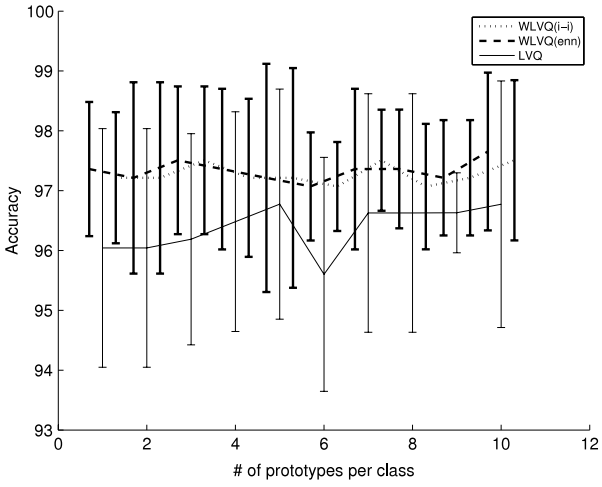

Fig. 3. Results for the *WBC Breast Cancer* data.

allowing for a small (statistically insignificant) improvement in accuracy. Adding more prototypes does not justify increased complexity of the model. Average accuracy of weighted versions has always been higher then that of the pure LVQ algorithm, but these differences are rather small.

The most significant difference has been obtained for the largest of these datasets, the *Spam* data. For this dataset both weighted versions of the LVQ algorithm work better than the standard LVQ, and for three prototypes per class the gain is statistically significant, over 4%. Although full 1-NN is still better by about 3%, which is significant, a great reduction in complexity has been achieved on a rather difficult data. A comparison of all results reported here is presented in Table 2.

Table 1
Summary of datasets used in experiments.

Title	#Features	#Samples	#Samples per class	Source
Sonar	60	208	111/97	Asuncion and Newman (2007)
WBC	9	683	444/239	Asuncion and Newman (2007)
Spambase	57	4601	1813/2788	Asuncion and Newman (2007)
Pima Indian diabetes	8	758	500/268	Asuncion and Newman (2007)
Cleveland	13	297	160/137	Asuncion and Newman (2007)
Ionosphere	33	351	225/126	Asuncion and Newman (2007)
Appendicitis	7	106	85/21	Shalom Weiss

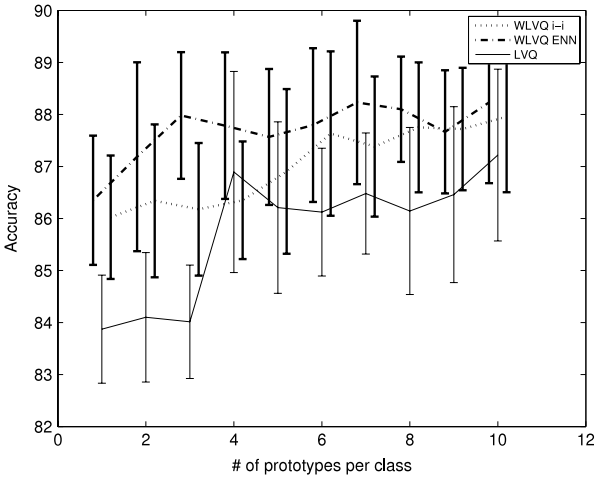

Fig. 4. Results for the *Spam* data.

6. Conclusions

Simple description of data structures is usually attempted using classical or fuzzy logic rules (Duch et al., 2004). Rules are comprehensible if their number is small and conditions are simple. Prototype-based rules provide an alternative description of data that may be simpler and more intuitive than propositional rules. One of the methods to generate good P-rules is based on the LVQ algorithm. A single prototype with a threshold decision rule may work well for data that has an approximately Gaussian distribution of samples (for example, on the Wisconsin Breast Cancer benchmark dataset Grąbczewski & Duch, 2002), requiring a prototype that represents either a typical or an extreme case in each class. In such a situation linear discrimination methods will also work quite well. However, in more complex cases, when nonlinear kernel methods show their advantages, more prototypes concentrated around border areas are needed. Kernel methods tend to use many support vectors along the decision border as a reference, implicitly providing new kernel-space features $z_i(\mathbf{x}) = K(\mathbf{x}, \mathbf{p}_i)$ that "flatten" decision borders (Maszczyk & Duch, 2010). Hyperplanes in this space provide good discrimination of the data, but interpretation of results becomes difficult because the data model becomes rather complex.

In this paper improvements of the LVQ algorithm based on instance weighting have been proposed. The goal is to decrease model complexity without loss of accuracy. Context dependent clustering has been applied for initialization of LVQ prototypes, with two weighting algorithms used for training: one based on the inter–intra class similarity, and another based on the Edited Nearest Neighbor rule instance selection algorithm. Results show that this combination may create simple solutions with a relatively small number of prototypes located near the border. In the case of Sonar data with inter–intra weighting only two prototypes per class are sufficient, while for WBC and Spam data ENN gives a more comprehensive description.

Table 2
Comparison of mean accuracy and the number of prototype vectors for 6 nearest neighbor based algorithms.

Dataset	Model	Accuracy	#Refs.
Sonar	1NN	87 ± 6.8	208
	ENN + 1NN	81.2 ± 6.5	174
	CNN + 1NN	85.8 ± 8.5	63
	WLVQ (inter–intra)	81.7 ± 3.9	2
	WLVQ (ENN)	81.7 ± 5.1	2
	LVQ	82.2 ± 5.0	20
WBC	1NN	95.9 ± 1.8	683
	ENN + 1NN	96.9 ± 1.9	661
	CNN + 1NN	94.7 ± 2.1	67
	WLVQ (inter–intra)	97.2 ± 1.2	2
	WLVQ (ENN)	97.4 ± 1.3	4
	LVQ	96.0 ± 2	10
Spam	1NN	89 ± 1.3	4600
	ENN + 1NN	90.3 ± 1.9	4165
	CNN + 1NN	88.4 ± 1.5	1055
	WLVQ (inter–intra)	87.6 ± 1.6	12
	WLVQ (ENN)	88.0 ± 1.2	6
	LVQ	86.9 ± 2.0	8
Pima Indian	1NN	66.3 ± 4.9	268
	ENN + 1NN	$76.2.2 \pm 5.2$	4165
	CNN + 1NN	66.3 ± 5.0	334
	WLVQ (inter–intra)	77.7 ± 4.2	2
	WLVQ (ENN)	77.4 ± 4.3	4
	LVQ	75.0 ± 2.7	2
Cleveland	1NN	75.1 ± 10.6	297
	ENN + 1NN	81.2 ± 7.7	4165
	CNN + 1NN	75.1 ± 10.6	104
	WLVQ (inter–intra)	$83.2.6 \pm 4.3$	2
	WLVQ (ENN)	83.2 ± 4.3	2
	LVQ	82.8 ± 3.7	4
Ionosphere	1NN	86.6 ± 4.9	351
	ENN + 1NN	84.9 ± 3.8	4165
	CNN + 1NN	86.6 ± 4.9	67
	WLVQ (inter–intra)	89.2 ± 2.0	4
	WLVQ (ENN)	89.2 ± 2.3	4
	LVQ	87.2 ± 3.1	4
Appendicitis	1NN	75.5 ± 10.2	106
	ENN + 1NN	88.7 ± 7.35	4165
	CNN + 1NN	75.5 ± 10.2	31
	WLVQ (inter–intra)	88.7 ± 8.4	2
	WLVQ (ENN)	88.7 ± 8.4	2
	LVQ	87.8 ± 7.8	2

Table 3
Comparison of mean accuracy for 3 algorithms not based on nearest neighbor rule.

Dataset	C.45	SVM-rbf	SVM-linear
Sonar	71.2 ± 7.1	90.39 ± 5.1	77.8 ± 8.4
WBC	94.6 ± 3.6	97.21 ± 1.1	96.9 ± 2.1
Spam	92.9 ± 1.2	93.6 ± 0.9	92.9 ± 0.6
Diabetes	73.8 ± 5.7	77.9 ± 2.4	76.8 ± 4.0
Cleveland	77.8 ± 8.8	84.15 ± 3.4	83.8 ± 6.3
Ionosphere	91.5 ± 3.3	95.45 ± 2.0	89.5 ± 4.0
Appendicitis	86.1 ± 11.9	87.7 ± 4.2	88.6 ± 5.9

For high dimensional data, selection of instances should be complemented with selection of features used by the LVQ

similarity metrics, and this is done in an elegant way using adaptive relevance estimation (Schneider, Hammer, & Biehl, 2009). Weighted LVQ may also be used with SVM kernel approaches to reduce the number of support vectors. PCA is commonly used for reduction of dimensionality, but a linear combination of futures may not provide as much information as distance-based features. The key to the success of the kernel methods lies in the creation of feature space based on distances to prototypes. Additional features greatly enhance the information available in the feature space, allowing for the construction of simpler models based on linear discrimination and other techniques (Maszczyk & Duch, 2010). It has been suggested that kernel methods may be useful in modeling human category learning in cognitive science (Jäkel, Schölkopf, & Wichmann, 2009), but prototype-based rules seem to provide a more plausible explanation preserving essential advantages of kernel methods.

References

Asuncion, A., & Newman, D. (2007). UCI machine learning repository. http://www.ics.uci.edu/~mlearn/MLRepository.html. URL: http://www.ics.uci.edu/~mlearn/MLRepository.html.

Bhattacharya, B., Mukherjee, K., & Toussaint, G. (2005). Geometric decision rules for instance-based learning problems. *Lecture Notes in Computer Science*, *3776*, 60–69.

Blachnik, M., & Duch, W. (2006). Prototype-based threshold rules. *Lecture Notes in Computer Science*, *4234*, 1028–1037.

Blachnik, M., & Duch, W. (2008). Prototype rules from SVM. In *Springer studies in computational intelligence*: *Vol. 80. Rule extraction from support vector machines* (pp. 163–184) Springer, (Chapter).

Blachnik, M., & Laaksonen, J. (2008). Image classification by histogram features created with learning vector quantization. *Lecture Notes in Computer Science*, *5163*, 827–836.

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

Duch, W. (2000). Similarity based methods: A general framework for classification, approximation and association. *Control and Cybernetics*, *29*, 937–968.

Duch, W., Adamczak, R., & Diercksen, G. H. F. (1999). Distance-based multilayer perceptrons. In M. Mohammadian (Ed.), *International conference on computational intelligence for modelling control and automation* (pp. 75–80). Amsterdam, The Netherlands: IOS Press.

Duch, W., Adamczak, R., & Diercksen, G. (2000). Classification, association and pattern completion using neural similarity based methods. *Applied Mathematics and Computer Science*, *10*, 101–120.

Duch, W., Adamczak, R., & Grąbczewski, K. (2001). A new methodology of extraction, optimization and application of crisp and fuzzy logical rules. *IEEE Transactions on Neural Networks*, *12*, 277–306.

Duch, W., & Blachnik, M. (2004). Fuzzy rule-based systems derived from similarity to prototypes. *Lecture Notes in Computer Science*, *3316*, 912–917.

Duch, W., & Grudziński, K. (2001). Prototype-based rules—a new way to understand the data. In *IEEE international joint conference on neural networks* (pp. 1858–1863). Washington, DC: IEEE Press.

Duch, W., Setiono, R., & Zurada, J. (2004). Computational intelligence methods for understanding of data. *Proceedings of the IEEE*, *92*(5), 771–805.

Grąbczewski, K., & Duch, W. (2002). Heterogenous forests of decision trees. *Springer Lecture Notes in Computer Science*, *2415*, 504–509.

Grochowski, M., & Jankowski, N. (2004). *Lecture Notes in Computer Science*, *3070*, 580–585.

Hart, P. E. (1968). The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, *114*, 515–516.

Jäkel, F., Schölkopf, B., & Wichmann, F. A. (2009). Does cognitive science need kernels? *Trends in Cognitive Sciences*, *13*(9), 381–388.

Jankowski, N., & Grochowski, M. (2004). Comparison of instance selection algorithms. I. Algorithms survey. *Lecture Notes in Computer Science*, *3070*, 598–603.

Kohonen, T. (1984). *Self-organization and associative memory*. Heidelberg, Berlin: Springer-Verlag.

Kuncheva, L. (1996). On the equivalence between fuzzy and statistical classifiers. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *15*, 245–253.

Kuncheva, L. (2004). *Combining pattern classifiers. methods and algorithms*. New York: J. Wiley & Sons.

Lee, Y.-J., & Huang, S.-Y. (2007). Reduced support vector machines: a statistical theory. *IEEE Transactions on Neural Networks*, *18*(1), 1–13.

Lowe, D. (1995). Radial basis function networks. In M. A. Arbib (Ed.), *The handbook of brain theory and neural networks* (pp. 937–940). Cambridge, MA: MIT Press.

Maszczyk, T., & Duch, W. (2010). Support feature machines: support vectors are not enough. In *World congress on computational intelligence* (pp. 3852–3859). IEEE Press.

Pedrycz, W. (2005). *Knowledge-based clustering: from data to information granules*. Wiley-Interscience.

Pękalska, E., & Duin, R. (2005). *The dissimilarity representation for pattern recognition: foundations and applications*. World Scientific.

Poggio, T., & Girosi, F. (1990). Network for approximation and learning. *Proceedings of the IEEE*, *78*, 1481–1497.

Rissland, E. L. (2006). Ai and similarity. *IEEE Intelligent Systems*, *21*(3), 39–49.

Roth, I., & Bruce, V. (1995). *Perception and representation* (2nd ed.) Open University Press.

Schapire, R., & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, *37*, 297–336.

Schneider, P., Hammer, B., & Biehl, M. (2009). Adaptive relevance matrices in learning vector quantization. *Neural Computation*, *21*, 3532–3561.

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

Tang, Y., & Lawry, J. (2010). A prototype-based rule inference system incorporating linear functions. *Fuzzy Sets and Systems*, *161*(21), 2831–2853.

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

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