

The Nearest Sub-class Classifier: a Compromise between the Nearest Mean and Nearest Neighbor Classifier

Cor J. Veenman* and Marcel J.T. Reinders†

Abstract We present the Nearest Sub-class Classifier (NSC), which is a classification algorithm that unifies the flexibility of the nearest neighbor classifier with the robustness of the nearest mean classifier. The algorithm is based on the Maximum Variance Cluster algorithm and as such it belongs to the class of prototype-based classifiers. The variance constraint parameter of the cluster algorithm serves to regularise the classifier, that is, to prevent overfitting. With a low variance constraint value the classifier turns into the nearest neighbor classifier and with a high variance parameter it becomes the nearest mean classifier with the respective properties. In other words, the number of prototypes ranges from the whole training set to only one per class. In the experiments, we compared the NSC with regard to its performance and data set compression ratio to several other prototype-based methods. On several data sets the NSC performed similarly to the k -nearest neighbor classifier, which is a well-established classifier in many domains. Also concerning storage requirements and classification speed, the NSC has favorable properties, so it gives a good compromise between classification performance and efficiency.

Keywords: Classification, regularisation, cross-validation, prototype selection.

*Corresponding author

†The authors are with the Department of Mediamatics, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, P.O. Box 5031, 2600 GA, Delft, the Netherlands. E-mail: {C.J.Veenman, M.J.T.Reinders}@ewi.tudelft.nl

1 Introduction

One of the most intriguing problems in automatic object classification is preventing overfitting to training data. The problem is that perfect training performance by no means predicts the same performance of the trained classifier on unseen objects. Given that the training data is sampled similarly from the true distribution as the unseen objects, the cause of this problem is twofold. First, the training data set contains an unknown amount of noise in the features and class labels, so that the exact position of the training objects in feature space is uncertain. Second, the training data may be an undersampling of the true data distribution. Unfortunately this is often the case, so that the model assumptions about the data distribution are not justified. Consider for instance the data set in Fig. 1(a), which is generated according to two Gaussian distributions. In the figure, a decision boundary is displayed that was estimated with the nearest neighbor rule, which can be seen to be overfit if one knows the origin of the data. Clearly, the optimal decision boundary between these two Gaussian distributions is a straight line. Without such prior knowledge it is much harder to know whether or not a learned classifier is overfit. In many cases, however, it is profitable to select less complex classifiers. Therefore, the basic assumption underlying overfitting prevention schemes is that simpler classification models are better than more complex models (especially in situations where the errors on the training data are equal). Unfortunately, there are situations in which this assumption does not hold [43], so a proper classifier validation protocol is essential. Consequently, if the bias for a simpler model was unjust, at least a proper error estimate can be given.

A common way to prevent overfitting, i.e., poor generalisation performance, is to incorporate a penalty function as a form of *regularisation* in the classification scheme. Regularisation is a way of trading off bias and variance in the classifier model, see for instance [24]. The purpose of the penalty function is to restrain the complexity of the classifier, so that the decision boundary becomes smoother or fewer features are effectively utilised. The classifier can be regularised by tuning an additional parameter that weights the penalty function with some model error. For instance, in ridge regression, a λ -parameter weights a penalty function that sums the squared weights in a linear classifier model with the model error [30]. When the total function that sums the model error expression and the λ -weighted penalty function is minimised, the squared weights are forced to be low proportionally to λ , leading to stress on or the removal of certain features.

In the case where there is neither undersampling nor noise in the training data, it is easy to model the data and the labels of unseen samples can be predicted from the model. The question is of course:

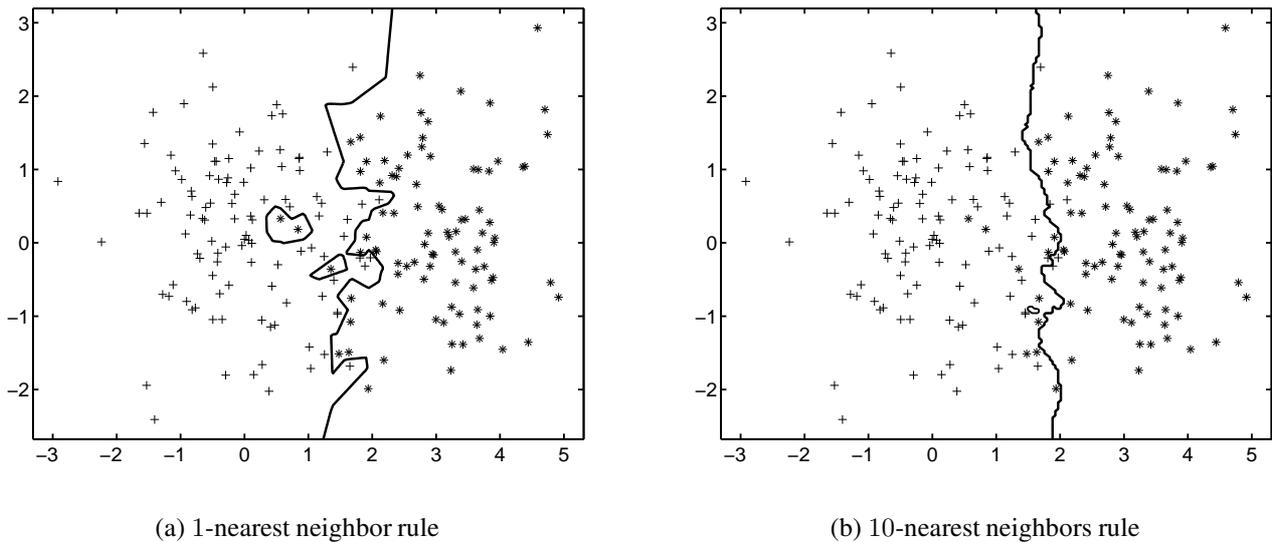


Figure 1: A data set consisting of two classes that are each generated according to a Gaussian distribution. In (a) the decision boundary is computed with the 1-nearest neighbor rule and is clearly overfit to the training data. In (b) the 10-nearest neighbors rule is applied and the corresponding decision boundary is closer to a straight line, which is optimal for the underlying distributions.

how does one know when there is enough data? Or, in other words: to what extent is the data representative of the underlying distribution? Other questions are: how does one know how much noise there is in the features, and whether all features have the same amount of noise? Since these questions are impossible to answer in general, the best solution is to restrain the training data fit with some form of regularisation such that the flexibility of the classifier can be controlled by an additional complexity parameter. Fortunately, the amount of regularisation can be learned from the data. We will return to this later on.

In this paper, we introduce a prototype-based classifier that employs the Maximum Variance Cluster algorithm (MVC) [48] to find proper prototypes. The number of clusters or prototypes follows from the imposed variance constraint value of the MVC, so that the number of prototypes can differ per class. Accordingly, through the variance constraint parameter the proposed classifier offers a compromise between the nearest mean classifier and the nearest neighbor classifier. Throughout the paper we will use the term regularisation in a more general way than only referring to the usual penalty function scheme. We consider any tuning that aims at avoiding overfitting as regularisation. A good example (without a penalty function) is the tuning of the number of neighbors involved in the k -nearest neighbors classifier, which clearly constrains overfitting. Compare for instance Fig. 1(a) and 1(b), where the decision boundary was estimated using the 1-nearest neighbor rule and 10-nearest

neighbor rule, respectively.

In the next section, we first introduce prototype-based classifiers. In Section 3, we present the classifier that is based on the Maximum Variance Cluster algorithm. In the experiments, Section 4, we compare the newly introduced classifier to other prototype-based classifiers and focus on the differences between them such as their performance with unevenly distributed classes.

2 Prototype-based Classification

Prototype-based classifiers are among the oldest types of classifiers. On the extremes of this type of classifiers are the Nearest Neighbor Classifier (NNC) [13], [14], [21] and the Nearest Mean Classifier (NMC) [28] (Ch. 13). The first does not abstract the data, but rather uses all training data to label unseen data objects with the same label as the nearest object in the training set. Consequently, it is a typical lazy learning algorithm with as many prototypes M as data objects N . The nearest mean classifier, on the other hand, only stores the mean of each class, i.e., one prototype per class. It classifies unseen objects with the label of the nearest class prototype.

The nearest neighbor classifier is very flexible and easily overfits to the training data. Accordingly, instead of 1-nearest neighbor, generally k nearest neighboring data objects are considered. Then, the class label of unseen objects is established by majority vote. We abbreviate this classifier as $\text{NNC}(k)$, where the parameter k represents the number of neighbors involved. Tuning k as a way to regularise the NNC gives a trade-off between the distribution of the training data with the a priori probability of the classes involved. When $k = 1$, the training data distribution and a priori probability is considered, while when $k = N$, only the a priori probability of the classes determines the class label.

The nearest mean classifier is very robust. It generally has a high error on the training data and on the test data, but the error on the training data is a good prediction of the error on the test data. When considered as a regularised version of the $\text{NNC}(1)$, the NMC has only one prototype per class instead of as many prototypes as the number of training objects. Clearly, reducing the number of labeled prototypes is another way of regularising the $\text{NNC}(1)$, where a high number of prototypes makes the classifier more (training data) specific and a low number makes it more general.

In this section, we focus on reducing the set of prototypes in order to regularise the NNC. Additionally, if not stated differently we employ the 1-nearest neighbor (prototype) rule to classify objects based on the reduced set of labeled prototypes.

Besides regularisation, there are other reasons for reducing the number of prototypes for nearest

neighbor classifiers. The most important ones are reducing the amount of required storage and improving the classification speed. Despite the continuous increase in memory capacity and CPU speed, especially in data mining, storage and efficiency issues become even more and more prevalent, see for instance [22].

2.1 Prototype Set Design Problem

Before we describe several ways of finding a set of prototypes for a given data set, we first formalise the problem. Let $X = \{x_1, x_2, \dots, x_N\}$ be a data set where x_i is a feature vector in a Euclidian space, and $N = |X|$ is the number of objects in X . In addition to a feature vector x_i , each object has a class label λ_i , where $1 \leq \lambda_i \leq z$, and z is the number of classes. Alternatively, we use $\lambda(x_i)$ and $\lambda(X)$ to refer to the label of object x_i or a set of labels of data set X , respectively. The prototype set design problem is to find a set P of $M = |P|$ prototypes that represent X such that P can be used for classification using the nearest neighbor rule. Accordingly, the elements of P are feature vectors in the same space as the objects in X . Further, P is the union of the class prototypes P_λ , where $P_\lambda = \{q_1, q_2, \dots, q_{M_\lambda}\}$ is the set of M_λ prototypes q_i with label λ .

The resulting set of prototypes P is either a subset of the original training set or it may be the result of a way of abstraction, usually averaging. The first design type is called instance filtering, where $P \subseteq X$. The reduced set is then also called a set of *S*-prototypes (*Selection*) [37]. When P is obtained by abstraction from the original training set, the reduction process is called instance averaging or instance abstraction. In [37] the resulting set is called a set of *R*-prototypes (*Replacement*). Instance filtering can always be used to reduce the set of prototypes. However, in order to apply an instance abstraction method certain conditions must be met; for example, it must be possible to compute the mean of two objects. That is, usually a Euclidian feature space is assumed, as we do in this paper. Combinations of instance filtering and abstraction have recently been reported in [38] and [39].

In the following sections, we shortly review instance filtering and abstraction methods. For the methods that we use in the experiments, we introduce an abbreviation. Further, in case an algorithm has a parameter to control the number of resulting prototypes we add that *tunable* or *regularisation* parameter in parentheses.

2.2 Instance Filtering

One of the earliest approaches in prototype selection was to reduce the prototype set size, while the error on the training set was to remain *zero*. The result of these so-called *condensing* methods is a *consistent subset* of the training set [27]. The *minimal* consistent subset is the smallest possible consistent subset. Finding the minimal consistent subset is an intractable problem [50] for which many suboptimal algorithms have been proposed like MCS [15] and [1], [23], [27], [42], [46].

There are also methods that relax the prototype consistency property. First, optimisation-based approaches combine classification accuracy with the minimisation of the prototype set size. Typically a combinatorial optimisation scheme is formulated to find that prototype set that minimises the sum of the error on the training set and the α weighted size of the prototype set. Several optimisation methods have been used ranging from random search [37], hill climbing [45], and genetic algorithms [37], to tabu search TAB(α) [5], [10].

Another instance filtering technique that does not aim at prototype consistency is called the Reduction Technique RT2 [52]. RT2(k) starts with $P = X$ and removes objects from P provided that the classification performance for *other* objects is not negatively affected by leaving the object out. The correct classification of the objects that are among the k nearest neighbors of the examined object x is considered.

A different example of how to obtain prototypes by instance filtering can be found in [12]. In this work, first objects in the most dense areas of the data set are identified, while object labels are ignored. The density around an object is defined as the number of other objects within a certain given distance h_n . Then, a heuristic is used to find a lower limit for the density of prototypes. This leads to a set of candidate prototypes from which a subset is chosen such that the distance between all object pairs is at least $2h_n$. Alternatively, in [40] the Multiscale Data Condensation algorithm MDC(k) is proposed that defines the density around an object as the distance r_i of the k -th nearest object.

Besides, outlier removal methods exist, which are by definition instance filtering approaches. The objective is to prevent the nearest neighbor rule from fitting to the training data without restrictions. An early example of such a procedure can be found in [51]. This method first determines whether or not an object is an outlier by a majority vote amongst the nearest neighbors of the object. If the label of the object differs from the label of the majority of its k neighbors, it is considered an outlier and the object is removed from the set of prototypes. We call this scheme the k -Edited Neighbors Classifier ENC(k) where the parameter k indicates the number of neighbors involved in the majority vote of the outlier detection scheme. An alternative outlier removal method is the repeatedly edited neighbors

method, where the procedure is repeated until the prototype set does not change anymore. In [47], a method is described with a parameter k that starts with $P = X$. Then, an object is removed from P if it would be removed by the just described method [51] with either a 1, 2, ... k nearest neighbor majority vote.

Lately, several combined filtering approaches have been reported like [9], [16], [52], [53]. These methods select a set of prototypes after the training set has been screened for outliers. For example, the RT3(k) algorithm [52] (also called DROP3 [53]) is a combination of ENC(k) and RT2(k). ENC is used to remove outliers or border objects before RT2 is applied.

2.3 Instance abstraction

Also using instance abstraction several condensing methods have been proposed to yield consistent prototypes. For instance in [7] and [11] the two closest objects are repeatedly merged as long as the error on the training set remains zero. The main difference between these methods is the way in which the mean of two objects is computed, i.e., the weighted mean in [11] and the normal arithmetic mean in [7]. In contrast, in [41] a more advanced class-conditional agglomerative hierarchical clustering scheme is employed to obtain consistent prototypes.

A simple and reportedly adequate method is the BootStrap technique BTS(M) [5], [26]. This method first randomly draws M objects from the data set. A candidate prototype set is constructed by replacing these objects by the mean of their k nearest neighbors. Then the error on the training set using the nearest neighbor rule with these prototypes is computed. This procedure is repeated for T trials and the prototype set with the lowest error on the training set is returned.

Another class of abstraction methods uses a kind of density estimation based on clustering techniques. With clustering techniques, there are several ways to obtain the prototypes and to involve the class labels in the clustering process. First the class labels can be ignored during the clustering and the prototypes can be labeled afterwards. These prototype labels can then be obtained in a number of ways such as using another classifier for this purpose or counting the majority of labels in each cluster. This scheme is called *post-supervised* learning in [36]. Second, the objects from the distinct classes can be clustered separately so that the prototypes directly receive the label of their class. In [36] this type of class supervision is called *pre-supervised* learning. The authors conclude that pre-supervised design methods generally have better performance than post-supervised methods [5], [36]. Moreover, the authors state that abstraction methods generally outperform filtering methods.

Another possibility is the optimisation of the positions of the prototypes with a certain class label

by considering objects from all classes, where objects with the same label contribute positively and objects with a different one negatively. This leads to prototype positioning that can incorporate the training error in the optimisation. This is neither a pre-supervised nor a post-supervised scheme, but rather a usual *supervised* scheme. An example of this type of prototype positioning is the supervised Learning Vector Quantisation LVQ(M) [5], [35].

Supported by the conclusions from [5] and [36], we propose a method that uses pre-supervised clustering (abstraction) to reduce the number of prototypes. We attempt to model the classes in order to reduce overfitting, while assuming certain amount of feature noise. Perhaps, the most widely used cluster algorithm is the k -means algorithm [3]. One way to find a reduced set of prototypes in a pre-supervised way is by running this cluster algorithm with a fixed number of clusters separately for each class. For instance, in Fig. 2 a data set is shown consisting of two curve-shaped classes. In Fig. 2(a) the data set is shown with a NNC(1) decision boundary and in Fig. 2(b) it is shown with the corresponding NNC(1) decision boundary based on the four prototypes with which each class is represented. The prototypes have been determined with the k -means algorithm. We call a classifier based on k -means prototypes positioning with a 1-nearest neighbor classification rule the K -Means Classifier KMC(M) where the parameter M indicates the number of prototypes contained in each class [28] (Ch. 13). When $M = 1$, the KMC equals the nearest mean classifier (NMC). Other clustering methods can also be used to estimate a fixed number of prototypes per class, e.g. Gaussian mixture modeling [32] and fuzzy C-means clustering [4].

The approach of selecting a fixed number of prototypes per class as an overfitting avoidance strategy seems straightforward, though it can be inadequate. When the class distributions differ from each other, either in the number of objects, the density of the objects or the shapes of the classes, the optimal number of prototypes may be different for each class. Consider for instance the data set in Fig. 3. It consists of a set of 300 objects from a curved class and a set of 100 objects from a radial class. To estimate a set of prototypes for this data set one needs fewer prototypes for the radial class than for the curved class. In Fig. 3(a) we show the prototypes and the decision boundary with four prototypes per class (KMC(4)) and in Fig. 3(b) one class is modeled with only one prototype and the other class with 4 prototypes. The latter classifier is simpler and from a test on a larger data set generated according to the same distributions it follows that it has a lower error on test data. With high dimensional data sets it is certainly not possible to inspect the decision boundary as we do here. In the experiments section, we study this data set in more detail and we show how to obtain a suitable number of prototypes in the general case, i.e. without visual inspection.

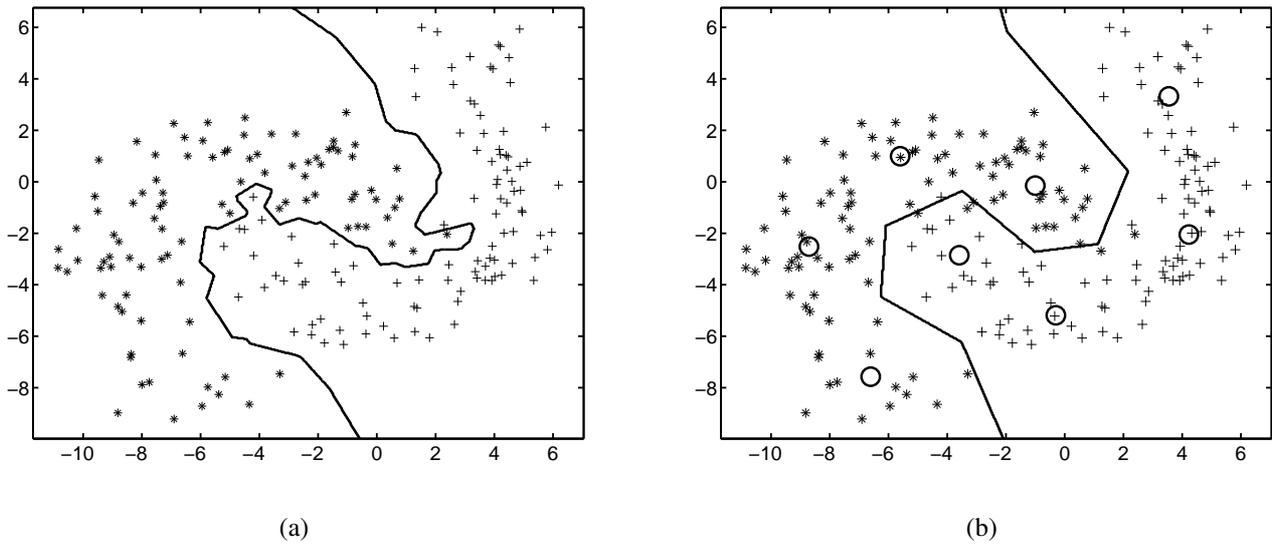


Figure 2: Dataset with two curve-shaped classes. In (a) the decision boundary is constructed using the NNC(1) rule. In (b) the data set is modeled with four prototypes per class and the corresponding NNC(1) decision boundary is included, i.e. the KMC(4) classifier.

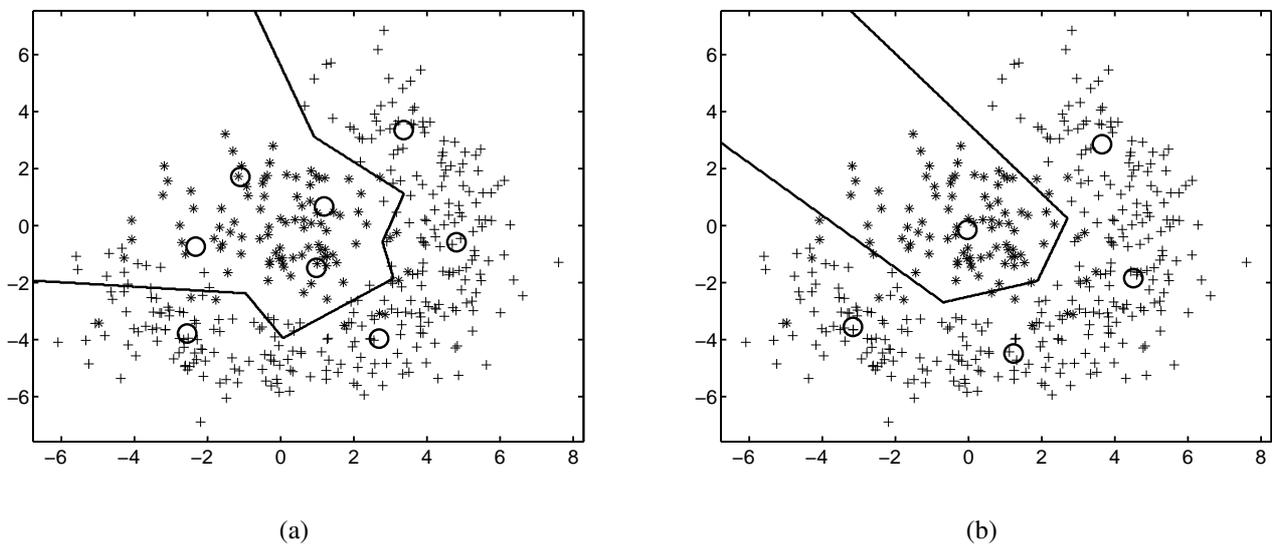


Figure 3: Dataset with two classes consisting of 300 and 100 objects. In (a) the data set is modeled with four prototypes per class and in (b) one class is modeled with four prototypes and the other with one prototype. The prototypes are positioned using the k -means algorithm and the decision boundaries are computed using the 1-nearest neighbor (prototype) rule.

The last example shows that when we devise the number of prototypes as a regularisation parameter of a nearest neighbor classifier, there should be one parameter per class instead of one global regularisation parameter M for all classes. This is, however, not common practice for prototype-based classifiers, where all classes usually have the same number of prototypes [28] (Ch. 13). As a result, some classes may be overfit while other classes are underfit.

In the next section, we propose a scheme that is able to assign a different number of prototypes to each class based on a single parameter.

3 The Nearest Sub-class Classifier

In this section we introduce the Nearest Sub-class Classifier (NSC). There are two assumptions underlying the NSC. First, we assume that the features of every object contain the same amount of noise. We do not model label noise, so the influence of wrongly labeled objects will be similar to that of outliers. Further, the undersampling of the classes is the same everywhere in feature space. This leads to the rationale behind the NSC: find the number of prototypes for each class in the data set such that the variance 'covered' by each prototype is the same. We introduce a variance constraint parameter that imposes the number of prototypes per class instead of the other way around. The classifier implements a pre-supervised scheme and classifies unseen objects with the label of their nearest prototype.

Before we describe the nearest sub-class classifier, we first outline the cluster algorithm that is used for prototype positioning. The cluster algorithm is the Maximum Variance Cluster algorithm (MVC) which is based on [49] and described in [48]. The MVC is a partitional cluster algorithm that aims at minimising the squared error for all objects with respect to their cluster mean. Besides minimising the squared error, it imposes a joint variance constraint on any two clusters. The joint variance constraint parameter σ_{max}^2 prevents the trivial solution where every cluster contains exactly one object. Moreover, the joint variance constraint generally leads to clusterings where the variance of every cluster is lower than the variance constraint value σ_{max}^2 [49].

More precisely, according to the MVC model a valid clustering of X into a set of clusters $C = \{C_1, C_2, \dots, C_M\}$, where $C_i \subseteq X$ and M is the number of clusters, is the result of minimising the squared error criterion, which is defined as:

$$\frac{\sum_{i=1}^M H(C_i)}{N}, \quad (1)$$

subject to the joint variance constraint:

$$\forall C_i, C_j, i \neq j : Var(C_i \cup C_j) \geq \sigma_{max}^2. \quad (2)$$

where

$$H(Y) = \sum_{x \in Y} \|x - \mu(Y)\|^2 \quad (3)$$

expresses the cluster homogeneity with $\mu(Y)$ is the mean vector of the objects in Y .

The MVC algorithm is a stochastic optimisation algorithm for the MVC model. We describe the algorithm while we refer the pseudo code in Alg. 1. The algorithm starts with as many clusters as samples [line 1]. Then in a sequence of epochs [line 2] every cluster has the possibility to update its content. Conceptually, in each epoch the clusters act in parallel, or, alternatively, sequentially in random order [line 5,7]. During the update process, each cluster undergoes a series of tests, each of which causes a different update action for that cluster. Since all actions aim at rearranging the objects in order to lower the squared error criterion and to satisfy the joint variance constraint, the algorithm exploits nearness in feature space. To this end the MVC algorithm maintains two index structures for each cluster C_a . The first index structure is the outer border B_a of the cluster. The outer border of order k contains the union of the k nearest foreigners of all cluster objects together with the cluster in which they are contained. A foreigner of an object is an object contained in a different cluster. Accordingly, the outer border helps in finding near clusters and objects therein. The second index structure is the inner border I_a , which simplifies the collection of remote cluster objects. The inner border of order q is the union of the q furthest neighbors of all objects in a cluster.

We explain the algorithmic steps of the MVC algorithm in detail. The algorithm has six parameters: the dataset X of size N , the maximum variance constraint value σ_{max}^2 , the maximum number of epochs without change $noChange_{max}$ for termination detection, and the order of the inner border q and outer border k . The output of the algorithm is a set of N clusters C_i , where M clusters are non-empty.

We continue the explanation assuming that the current cluster is C_a . The three key steps for C_a are:

S1 Isolation: When the variance of cluster C_a is above σ_{max}^2 , it is possible to lower the squared error criterion by making a singleton cluster from an object in the inner border. Since the constrained optimisation problem (1)-(3) may result in clusters with a variance higher than σ_{max}^2 , we allow step *S1* only for a certain period (E_{max} epochs). In this way we prevent oscillations between step *S1* and the next step *S2*.

The isolation step works as follows. Check C_a to see whether its variance exceeds the predefined maximum σ_{max}^2 and the epoch counter does not exceed E_{max} [line 12]. If so, randomly select $i_a = \lfloor \sqrt{|I_a|} \rfloor$ candidates from the inner border I_a [line 13]. Isolate the candidate that is furthest from the cluster mean $\mu(C_a)$ [line 14]. The isolated sample is removed from C_a [line 15] and forms a new cluster [line 16].

S2 Union: When the variance of the union of two clusters remains below σ_{max}^2 , the joint variance constraint is violated. These clusters should be merged to enable the satisfaction of the joint constraint in a future epoch.

This is achieved as follows. Check if the variance of C_a is below σ_{max}^2 [line 17]. Then search for a neighboring cluster with which C_a can be united, where a neighboring cluster is a cluster that contains an object from the outer border B_a of C_a . To this end, compute the joint variance of C_a with each of its neighbors [line 18-21]. If the lowest joint variance remains under σ_{max}^2 , then the corresponding neighboring cluster is merged with C_a [line 22]. For termination detection we remember that a cluster changed in this epoch [line 23].

S3 Perturbation: Finally, if neither *S1* nor *S2* applies, the squared error criterion (1) can possibly be lowered by swapping an object between two clusters.

To this end, randomly collect $b_a = \lfloor \sqrt{|B_a|} \rfloor$ candidates from the outer border B_a of C_a [line 25]. Compute for these candidates from neighboring clusters the gain in the squared error criterion that can be obtained when moving them from the current cluster C_b to C_a [line 27-30].

We define the criterion gain between C_a and C_b with respect to $x \in C_b$ as:

$$G_{ab}(x) = H(C_a) + H(C_b) - H(C_a \cup \{x\}) - H(C_b - \{x\}). \quad (4)$$

If the best candidate x_{max} has a positive gain then this candidate moves from the neighbor C_m to C_a [line 31-34].

The algorithm terminates if nothing happened to any of the clusters for $noChange_{max}$ epochs [line 38]. As explained for step *S1*, after E_{max} epochs object isolation is no longer allowed to prevent oscillations between *S1* and *S2*. With that precaution, the algorithm will certainly terminate, since the overall homogeneity criterion only decreases and it is always greater than or equal to zero. For a performance analysis and a comparison with other techniques like the k -means algorithm and the

```

input : Data set  $X$  of  $N$  objects, constraint value  $\sigma_{max}^2$ 
         parameters:  $E_{max}$ ,  $noChange_{max}$ ,  $k$ ,  $q$ 
output: Set of  $N$  clusters  $C_i$ , where  $M$  clusters  $C_i \neq \emptyset$ 
1 for  $i = 1$  to  $N$  do  $C_i = x_i$ ;
2  $epoch = lastChange = 0$ ;
3 repeat
4    $epoch = epoch + 1$ ;
5    $index = randomPermutation(\{1..N\})$ ;
6   for  $j = 1$  to  $N$  do
7      $a = index(j)$ ;
8     if  $C_a = \emptyset$  then continue;
9      $B_a = outerBorder(C_a, k)$ ;  $b_a = \sqrt{|B_a|}$ ;
10     $I_a = innerBorder(C_a, q)$ ;  $i_a = \sqrt{|I_a|}$ ;
11    switch do
12      case  $Var(C_a) > \sigma_{max}^2$  and  $epoch < E_{max}$ 
13        /* Isolation: lower squared error criterion */
14         $Y = randomSubset(I_a, i_a)$ ;
15         $x = furthest(Y, \mu(C_a))$ ;
16         $C_a = C_a - \{x\}$ ;  $C_m = \{x\}$ ; /* where  $C_m$  was previously empty */
17      case  $Var(C_a) \leq \sigma_{max}^2$  and  $\exists b : Var(C_a \cup C_b) \leq \sigma_{max}^2$ 
18        /* Union: satisfy joint variance constraint */
19         $s_{min} = \infty$ ;
20        foreach  $\{b \mid C_b \cap B_a \neq \emptyset\}$  do
21          if  $Var(C_b \cup C_a) < s_{min}$  then  $s_{min} = Var(C_b \cup C_a)$ ;  $m = b$ ;
22        end
23         $C_a = C_a \cup C_m$ ;  $C_m = \{ \}$ ;
24         $lastChange = epoch$ ;
25      otherwise
26        /* Perturbation: lower squared error criterion */
27         $Y = randomSubset(B_a, b_a)$ ;
28         $g_{max} = -\infty$ ; /*  $g_{max}$  stores the maximum gain */
29        foreach  $x \in Y$  do
30           $b = (n \mid C_n \cap x \neq \emptyset)$ ;
31          if  $G_{ab}(x) > g_{max}$  then  $g_{max} = G_{ab}(x)$ ;  $m = b$ ;  $x_{max} = x$ ;
32        end
33        if  $g_{max} > 0$  then
34           $C_a = C_a \cup \{x_{max}\}$ ;  $C_m = C_m - \{x_{max}\}$ ;
35           $lastChange = epoch$ ;
36        end
37      end
38    end
39  end
40 until  $epoch - lastChange > noChange_{max}$ ;

```

Algorithm 1: The MVC algorithm

mixture of Gaussians technique, refer to [48]. In short, the MVC handles outliers adequately and clearly better than the k -means algorithm. Further, it finds the (close to) global optimum of its cluster model more often, and when the number of clusters is high, it is also faster than the k -means algorithm and mixture of Gaussians technique with EM optimisation.

For the classification problem, class λ has a set of prototypes $P_\lambda = \{q_1, q_2, \dots, q_{M_\lambda}\}$, where $q_k = \mu(C_k)$ has been established after clustering the objects X_λ with label λ from X into M_λ distinct clusters. Further, the estimated label λ' of an unlabeled object x is established according to the nearest neighbor rule.

MVC has a few optimisation parameters: $E_{max} = 100$, $noChange_{max}$, $k = 3$, and $q = 1$. Further, it has only one (tunable) model parameter, namely σ_{max}^2 . Consequently, the Nearest Sub-class Classifier (NSC) also has only one model parameter. We therefore refer to this classifier as NSC(σ_{max}^2). When σ_{max}^2 is set to $\sigma_{max}^2 = 0$, the variance constraint causes every object to be put in a separate cluster, so that the NSC turns into the NNC(1) with as many prototypes as objects. Consequently, when σ_{max}^2 is low, the NSC is flexible but it easily overfits to the training data. At the other extreme, when $\sigma_{max}^2 \rightarrow \infty$, the NSC turns into the NMC with its associated properties. In other words, the σ_{max}^2 parameter offers a convenient way to traverse the scale between the nearest neighbor and the nearest mean classifier. In contrast with the KMC(M) classifier, the number of prototypes is adjusted per class through the variation of one parameter, i.e. the maximum variance parameter σ_{max}^2 .

4 Experiments

In this section, we describe our experiments for the evaluation of the NSC classifier. For the performance evaluation, we used one artificial and several real-life data sets as shown in Table 1. We first elaborate on the tuning of the parameters of the NSC and other classifiers included in the experiments. Further, we describe the way we rate the classifier performance. Then, we list the algorithms that we included in the performance comparison and we describe the results.

4.1 Parameter Tuning

When we train a classifier on a labeled data set, we estimate a set of parameters such that the classifier can predict the labels afterwards. For several classifiers, including prototype-based classifiers that we consider here, there are additional model parameters that cannot be learned directly from the training

data. The number of neighbors for the NNC classifier and the maximum variance σ_{max} for NSC are examples of such parameters. Often these parameters are regularisation parameters that constrain the fitting to the training data.

For prototype-based classifiers, regularisation is implemented through the number of prototypes. Two classifiers that we concern in the experiments are based on cluster algorithms; in these cases the number of prototypes corresponds to the number of clusters. In the classification domain, the usual way of establishing a suitable number of clusters is not appropriate, that is, by computing cluster validation criteria for several numbers of clusters and choosing an optimum number, as in [6], [17], [18], [31]. That is, it is debatable whether well-separated groups of objects *per class* exist. For the same reason, the plateau heuristic [48] in the squared error criterion will not help in finding a proper setting of the σ_{max}^2 parameter for the NSC.

Alternatively, the number of prototypes can be optimised by means of a cross-validation protocol, either n -fold or leave-one-out. For example, the variance constraint σ_{max}^2 in the NSC(σ_{max}^2) that results in the lowest leave-one-out cross-validation error can be said to optimally regularise this classifier. The resulting classifier is expected to have the best generalisation performance for the respective parameter setting. However, because the parameter is optimised with a cross-validation feedback loop this can no longer be called *validation*. In the experiments we used 10-fold cross-validation. In order to make the parameter estimation more reliable, we repeated the cross-validation three times, i.e. for three independent draws of 10 non-overlapping subsets from the training set.

4.2 Performance Estimation

Since the tuning by cross-validation procedure is a specific training procedure, an additional validation protocol is needed to estimate the performance of the classifier, see e.g. [2], [29], and [34].

In the validation procedure we handle artificial data different from real-life data. Since we know the distribution of the artificial data sets, we generate a large data set for testing purposes. We draw 20 smaller samples from the same distribution on which we train the classifiers and tune their parameters by cross-validation. Per sample the error is computed by testing the performance on the large reference data set. We report the performance as the average of the 20 tests. For real-life data sets on the other hand, we use a 10-fold cross-validation protocol to estimate the validation performance. In that case, we repeat the tuning by cross-validation procedure in every fold of the 10-fold cross-validation procedure. We also repeat the 10-fold cross-validation 10 times and we report the average error. Further, we consider the classifier performance for a certain dataset equal to the best classifier, when their

performance is equal according to the paired t test with significance level $\alpha = 0.05$.

Among the motivations for prototype set design methods is the reduction of storage requirements and the increase of classification speed. For instance, one of the algorithms included in the experiments, the MDC algorithm, was especially designed to reduce data storage. For this reason we also measured the compression ratio r_c of the trained classifiers, where $r_c = \frac{|P|}{|X|} = \frac{\#prototypes}{dataset\ size}$.

4.3 Algorithms

In the evaluation of the NSC classifier, we compare it to other prototype-based classifiers. In this comparison we included the K -Means Classifier $KMC(M)$ [28], the k -Nearest Neighbors Classifier $NNC(k)$ [13], the k -Edited Neighbors Classifier $ENC(k)$ [51], the Multiscale Data Condensation algorithm $MDC(k)$ [40], the Bootstrap technique $BTS(M)$ with $T = 100$ and $k = 3$ [5], [26], and Learning Vector Quantisation $LVQ(M)$ [5], [35] with $\alpha = 0.3$, $\eta = 0.8$, and $T = 100$ as in [5]. The LVQ algorithm we used is $LVQ1$, which was a top performer (averaged over several data sets) in [5]. We did not consider more advanced LVQ versions because these have even more parameters.

In addition to the classifiers for which we optimised their parameter with tuning by cross-validation, we included four classifiers with a fixed parameter setting and one parameterless classifier. First, we included the $NNC(3)$ classifier, which is often considered a reasonably regularised version of the 1-nearest neighbor classifier, see e.g. [19], [33], [53]. Second, we included Reduction Technique 3 $RT3(k)$ [52], [53], for which we set $k = 3$ as used in the reported experiments, and Tabu Search $TAB(\alpha)$ [5], [10] with $\alpha = 0.05$, $T_t = 0.03N$, $T_i = 20$, and $T = 100$ as in [5]. We did not tune the parameter of these two classifiers, solely because this process appeared to be too time consuming. Finally, we compared the performance to that of the MCS condensing algorithm [15].

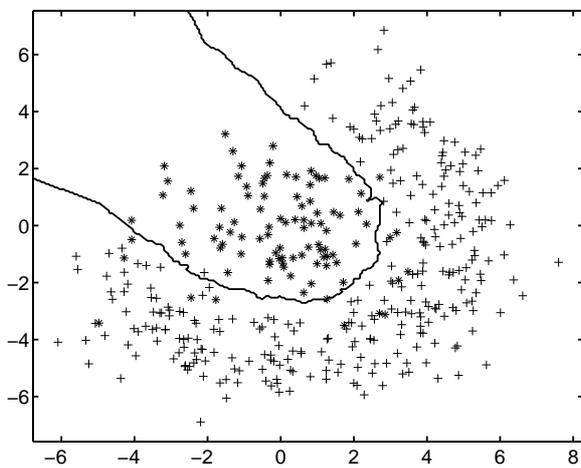
4.4 Results

In order to stress the difference between the $NSC(\sigma_{max}^2)$ and the $KMC(M)$, which is the most similar to the NSC, we used an artificial data set containing classes with different shapes, numbers of objects, and densities. The artificial data set was generated as a curved distribution around a Gaussian distribution. The curved distribution was generated as a Gaussian distribution superimposed on a uniformly distributed half circle. In Fig. 4 we show one sample from this data set with 300 and 100 objects per class that is used to train and tune by cross-validation. The reference data set that is used for the testing contains 3000 objects in the curved distribution and 1000 objects in the other distribution.

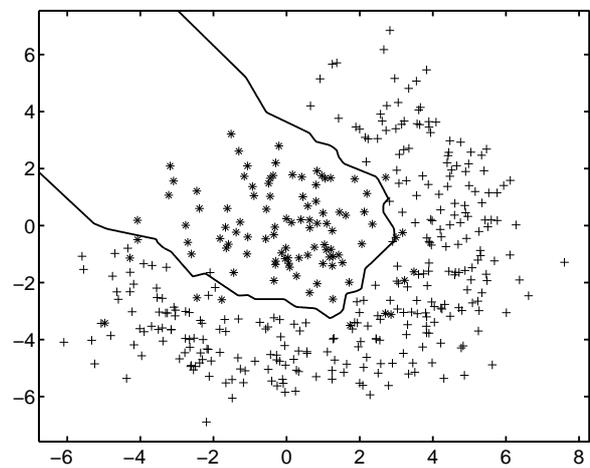
For the NSC(σ_{max}^2), KMC(M), NNC(k), ENC(k), MDC(k), BTS(M), and LVQ(M), we first tuned their regularisation parameter on the sample of 400 objects (that is also shown in Fig. 3). In Fig. 3(a) and 3(b), we already showed the decision boundary corresponding to the optimal parameter setting of the NSC and KMC for this sampling of the data set. In Fig. 4 and 5, we additionally show the decision boundary corresponding to the optimum parameter setting for the NNC, ENC, MDC, BTS, and LVQ. The validation results (averaged over 20 draws) in Table 2(a) show that the NSC outperforms the KMC. This is not surprising, since with the optimum σ_{max}^2 setting the NSC models the classes with respectively $M_1 = 1$ and $M_2 = 4$ prototypes, while the KMC utilises $M = 4$ prototypes for both classes. As a consequence, with the KMC one class is overfit leading to an unsmooth decision boundary as can be seen in Fig. 3(a) and 3(b). As can be seen in Table 2(a), ENC, BTS, LVQ and TAB perform similarly with regard to NSC and the NNC(k) performs slightly better than the NSC.

With the first real-life data set, we wanted to show how the NSC works on a well-known data set and how its performance relates to the other algorithms. For this purpose we used the Iris data set [20]. This data set consists of 150 objects that are subdivided in three Iris classes and each object contains four features. With the optimum regularisation parameter value for the NSC ($\sigma_{max}^2 = 0.29$), the number of prototypes per class is $M_1 = 2$, $M_2 = 3$, and $M_3 = 4$. The optimum result of the KMC gave 3 prototypes per class. The cross-validation results displayed in Table 2(a) show that except the MCS and RT3 all other algorithms have similar performance.

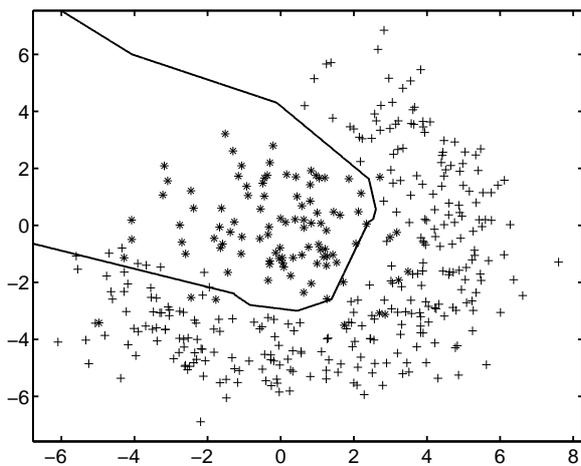
Further, we used two real-life data sets that clearly show the difference between the KMC and the NSC. The cause of this difference is that for both data sets one class can be modeled with substantially fewer prototypes than the other. The first data set is the Wisconsin Breast Cancer Dataset [54]. After removing incomplete data records, this data set contained 683 objects with 9 numerical features each. Of the 683 patients, 444 are in the benign class and 239 in the malignant class. Interestingly, for the optimum σ_{max}^2 setting obtained via tuning by cross-validation on the whole data set ($\sigma_{max}^2 = 35$) the NSC needed only one prototype for the benign class and 9 prototypes for the malignant class. However, since the cluster algorithms in the KMC and NSC cannot generally find well-separated clusters, it cannot be concluded that the clusters in the malignant class represent distinct groups of patients. On the other hand, the large difference between the numbers of clusters suggests that at least the variance is larger in the malignant class and that the malignant class possibly contains several groups of similar tissues. The KMC has a peak in the tuning by cross-validation curve at $M = 1$ and a second one around $M = 9$ (see Fig. 6), which illustrates the conflict between choosing



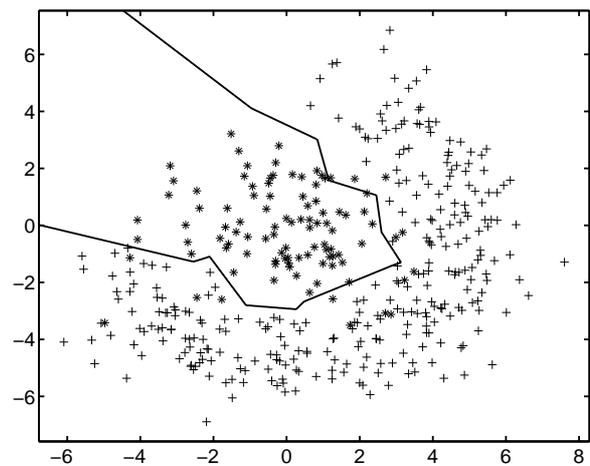
(a) NNC(15)



(b) ENC(6)

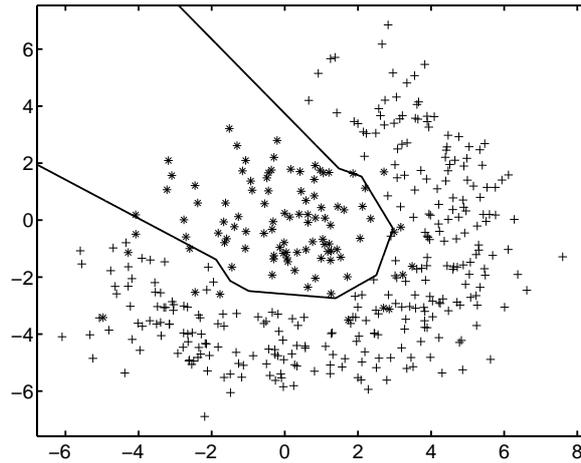


(c) MDC(16)



(d) BTS(23)

Figure 4: Dataset with two classes consisting of 300 and 100 objects. In (a), (b), and (c) the decision boundary is drawn according to the tuned NNC($k = 15$), ENC($k = 6$), MDC($k = 16$), and BTS($M = 23$).



(a) LVQ(9)

Figure 5: Dataset with two classes consisting of 300 and 100 objects. The decision boundary is drawn according to the tuned LVQ($M = 9$).

Dataset	objects (n)	features (p)	classes (z)
Artificial	400	2	2
Iris	150	4	3
Breast cancer	683	9	2
Ionosphere	351	34	2
Glass	214	9	6
Liver disorders	345	6	2
Pima Indians	768	8	2
Sonar	208	30	2
Wine	178	13	3

Table 1: Overview of the characteristics of the data sets used in the experiments.

a suitable number of prototypes for the benign class and the malignant class. For the parameter values obtained with tuning by cross-validation of the other algorithms see Table 2(b), where we also show the achieved compression ratio of the algorithms. The cross-validation results are shown in Table 2(a). The results show that indeed the NSC performs better than the KMC by modeling the classes with a separate, adjustable number of clusters. Among the other tuned classifiers the differences in performance are generally small, while MCS and RT3 again performed worst.

The second data set that apparently has pronounced differences in class distributions is the Ionosphere data set [44]. After removing objects with missing values, the Ionosphere data set contained 351 objects with 34 numerical features. The objects are subdivided in 225 "good" objects and 126

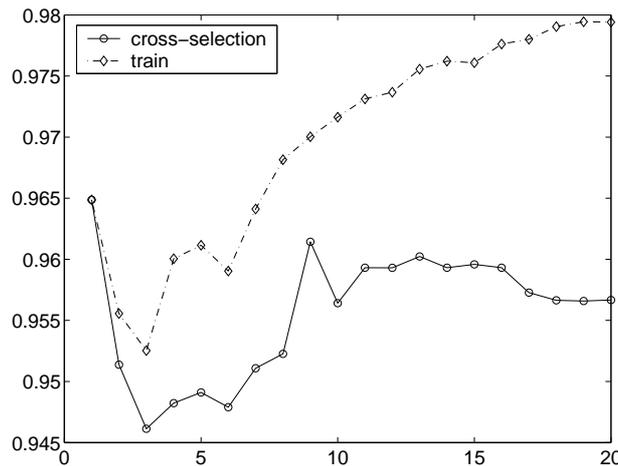


Figure 6: Tuning by cross-validation curve of the KMC classifier for the Wisconsin Breast Cancer Dataset.

”bad” objects. The parameter values obtained with tuning by cross-validation on the whole data set are displayed in Table 2(b). The NSC had 9 prototypes in the ”good” class and 100 prototypes in the ”bad” class at its optimal σ_{max}^2 value. The cross-validation results are shown in Table 2(a). The results show again that the NSC performed better than the KMC by modeling the classes with a separate, adjustable number of clusters. Moreover, on this problem the NCS performed better than all the other algorithms in our test.

Finally, we tested all the algorithms on five more data sets from the Machine Learning Database Repository [8]. The data sets were selected for their numerical features and because they have no missing data. As can be seen in Table 2(a), in all these experiments the NSC performed similar or better than the KMC classifier. The last rows in Table 2(a) summarise the performance of all algorithms. The table shows that, with respect to the average performance and the number of times the algorithms were top-ranked (not significantly worse than the best algorithm), with the given datasets only $NNC(k)$ achieves better results than $NCS(\sigma_{max}^2)$. Further, the tuned $NNC(k)$ consistently performed better than the $NNC(3)$ classifier.

Remarkably, the tuned MDC often turns into the $NNC(1)$, that is, the optimal parameter setting was $k = 1$. Sometimes the resulting MDC classifier was in agreement with $NNC(k)$, but in these cases NSC performed similarly while resulting in a smaller set of prototypes. As a result, when optimised for classification performance the storage reduction of the MDC is limited.

When we consider the prototype set size of the trained classifiers, we can conclude that the $KMC(M)$ and $RT3(3)$ achieved the highest average compression, see Table 2(b). $RT3(3)$ had, how-

Dataset	NSC(σ_{max}^2)	KMC(M)	NNC(k)	ENC(k)	MDC(k)	BTS(M)	LVQ(M)	MCS	TAB(0.05)	NNC(3)	RT3(3)
Artificial	93.9 ± 0.4	92.2 ± 0.9	94.5 ± 0.3	94.2 ± 0.3	92.6 ± 1.3	93.7 ± 0.5	93.9 ± 0.8	90.1 ± 1.2	93.9 ± 0.5	93.8 ± 0.5	87.4 ± 4.5
Iris	96.3 ± 0.4	96.2 ± 0.8	96.7 ± 0.6	96.3 ± 0.7	95.3 ± 0.4	95.6 ± 1.0	96.1 ± 0.6	93.2 ± 0.9	94.0 ± 3.7	96.0 ± 0.3	81.1 ± 8.6
Breast cancer	97.2 ± 0.2	95.9 ± 0.3	97.0 ± 0.2	96.7 ± 0.3	95.6 ± 0.7	97.1 ± 0.2	96.3 ± 0.4	93.8 ± 0.7	97.0 ± 0.3	97.0 ± 0.3	94.7 ± 1.0
Ionosphere	91.9 ± 0.8	87.4 ± 0.6	86.1 ± 0.7	83.3 ± 0.7	86.0 ± 0.7	88.9 ± 1.3	86.4 ± 0.8	86.9 ± 0.6	88.5 ± 1.4	84.8 ± 0.5	70.2 ± 3.4
Glass	70.2 ± 1.5	68.8 ± 1.1	72.3 ± 1.2	67.8 ± 0.7	73.1 ± 0.7	70.0 ± 3.2	68.3 ± 2.0	67.9 ± 1.5	69.3 ± 2.5	69.6 ± 1.1	56.5 ± 1.5
Liver disorders	62.9 ± 2.3	59.3 ± 2.3	67.3 ± 1.6	67.9 ± 1.0	61.0 ± 1.5	63.9 ± 2.9	66.3 ± 1.9	57.1 ± 1.2	64.3 ± 3.0	64.1 ± 1.1	60.5 ± 2.9
Pima indians	68.6 ± 1.6	68.7 ± 0.9	74.7 ± 0.7	74.0 ± 0.8	67.9 ± 1.7	74.0 ± 0.8	73.5 ± 0.9	63.8 ± 0.7	74.4 ± 1.4	69.1 ± 0.6	68.9 ± 0.9
Sonar	81.3 ± 1.1	81.9 ± 2.5	81.8 ± 1.4	79.8 ± 1.4	82.7 ± 1.0	75.4 ± 3.1	78.3 ± 2.4	80.5 ± 1.9	73.5 ± 2.1	81.2 ± 0.8	65.5 ± 4.1
Wine	75.3 ± 1.7	71.9 ± 1.9	73.9 ± 1.9	68.9 ± 1.1	75.2 ± 1.7	70.0 ± 1.8	72.3 ± 1.5	72.6 ± 1.5	72.4 ± 2.9	73.5 ± 1.6	67.9 ± 3.1
Average	82.0 ± 1.1	80.3 ± 1.2	82.7 ± 1.0	81.0 ± 0.8	81.0 ± 1.1	81.0 ± 1.6	81.3 ± 1.3	78.4 ± 1.1	80.8 ± 2.0	81.0 ± 0.7	72.5 ± 3.3
Top	4	2	8	3	3	2	1	0	2	1	0

(a) Validation performance

Dataset	NSC		KMC		NNC		ENC		MDC		BTS		LVQ		MCS	TAB	NNC(3)	RT3(3)
	r_c	σ_{max}^2 *	r_c	M^*	r_c	k^*	r_c	k^*	r_c	k^*	r_c	M^*	r_c	M^*	r_c	r_c	r_c	r_c
Artificial	1.5	4.25	2.0	4	100	14	96	9	3.5	16	3.8	15	5.5	22	11	12	100	4.8
Iris	7.3	0.25	8.0	4	100	14	96	3	9.3	5	10	15	15	22	9.3	3	100	7.3
Breast cancer	1.8	35.0	0.29	1	100	5	97	5	1.8	19	1.6	11	5.9	40	8.4	7	100	1.3
Ionosphere	31	1.25	4.0	7	100	2	90	2	100	1	11	39	6.8	24	15	24	100	3.7
Glass	97	0.005	17	6	100	1	73	1	100	1	56	119	45	97	38	13	100	15
Liver disorders	4.9	600	11	19	100	14	69	8	100	1	4.1	14	8.4	29	51	18	100	18
Pima Indians	1.7	2600	1.0	4	100	17	74	15	8.1	4	0.3	2	3.4	26	43	15	100	11
Sonar	70	0.050	17	18	100	1	81	3	100	1	18	37	19	40	26	12	100	19
Wine	96	4.0	29	17	100	1	70	16	100	1	4.5	8	32	57	35	7	100	12
Average	35		9.9		100		83		58		12		16		26	12	100	10

(b) Properties of trained classifiers

Table 2: Overview of the experimental results. In (a) the performance (in percentages) of the classifiers is displayed with standard deviation. The performance is printed in bold face, when the respective classifier performs equally well as the best classifier for that data set. The last rows show for each algorithm the average performance over all datasets, and how many times each algorithm scored as best. Table (b) shows the compression ratio r_c of the trained/tuned classifiers. The classifiers left of the double bar were optimised with tuning by cross-validation. In (b), for these classifiers also the optimal parameter value is shown.

ever, the worst overall classification performance. The proposed NSC also achieved high compression ratios; better than the ENC, MDC, and of course the NNC. That is, the NNC classifiers use always all data for classification. The ENC does not aim at redundancy reduction in the first place. Accordingly, in all experiments its compression ratio is limited.

5 Discussion and Conclusions

In this paper, we introduced the Nearest Sub-class Classifier (NSC), a prototype-based classifier that uses the Maximum Variance Cluster algorithm (MVC) [48] to position its prototypes in feature space.

In contrast to the K-Means Classifier (KMC) [28] (Ch. 13), which is a typical cluster-based classifier, the variance constraint parameter of the MVC algorithm results in a different number of prototypes per class. Consequently, the NSC algorithm requires a single parameter while the KMC needs one parameter per class to achieve a similar performance.

Our experiments showed that it is indeed beneficial to have a distinct number of prototypes per class as an overfitting avoidance strategy. With the given datasets, the NSC always performed similarly or better than the KMC. Moreover, the optimisation properties of the NSC are favorable compared to those of the KMC. When the number of prototypes needed is high, the differences between the underlying clustering algorithms, MVC and k -means, are clear [48]. That is, especially when the number of clusters is high the MVC finds the (close to) global optimum more often and faster.

In the experiments, we further compared the NSC to the k -Nearest Neighbors Classifier (NNC(k)) [13] the k -Edited Neighbors Classifier (ENC(k)) [51], the Multiscale Data Condensation algorithm (MDC(k)) [40], the Bootstrap technique BTS(M) [5], [26], and Learning Vector Quantisation LVQ(M) [5], [35]. These algorithms all have a tunable (regularisation) parameter: the number of neighbors for classification for NNC, the number of neighbors for editing for ENC, the number of neighbors for the density computation for MDC, and the number of prototypes for BTS and LVQ. We optimised these tunable parameters for all classifiers by means of the cross-validation protocol. In the experiments, we also included the Minimal Consistent Subset algorithm MCS [15], Reduction Technique 3 (RT3) [52] with $k = 3$, Tabu Search TAB(α) [5], [10], [25] with $\alpha = 0.05$, and NNC(3) as reference classifier.

The experiments showed that on several data sets, the proposed NSC performed comparably to the tuned NNC, which is a well-established classifier in many domains. Apart from the NNC(k), the NSC has the highest average performance and it scored 'best' the most times. The NNC(k), however, needs all training data to classify new objects, which is computationally expensive both in time and storage.

Based on this set of datasets, it is hard to predict to which type of data sets the NSC should be applied. The NSC algorithm assumes numerical data without missing values. Further, the NSC algorithm is a density estimation based classifier. Model bias introduced by the NSC can be beneficial like in general with density based classifiers. As future work, we consider studying more diverse and larger data sets as an important step to extend our knowledge on the general applicability of the NSC.

As could be expected and can be derived from the improved performance of the NNC(k) compared to the NNC(3), we can conclude that tuning by cross-validation is indeed profitable. This tuning procedure is, however, computationally demanding. Especially when the tuned classifier has to be

cross-validated to estimate its performance as in this paper. Consequently, we did not use this tuning procedure for the RT3 algorithm, which could be the reason for its disappointing performance compared to the other algorithms. Also, the Tabu search algorithm was too time-consuming to be tuned. Remarkably, its average performance was similar to the NNC(3) and most of the tuned algorithms.

Finally, when storage and classification speed issues are considered, the NSC has favorable properties. It did not result in the smallest average prototype set size, but it yielded the best compromise between classification performance and efficiency.

Acknowledgements

This research is supported by PAMGene within the BIT program ISAS-1. We thank Dr. L.F.A. Wessels for proofreading and valuable comments.

References

- [1] D.W. Aha, D. Kibler, and M.K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.
- [2] C. Ambroise and G.J. McLachlan. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the National Academy of Sciences (PNAS)*, 99(10):6562–6566, May 2002.
- [3] G.H. Ball and D.J. Hall. A clustering technique for summarizing multivariate data. *Behavioral Science*, 12:153–155, mar 1967.
- [4] J.C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, 1981.
- [5] J.C. Bezdek and L.I. Kuncheva. Nearest prototype classifier designs: an experimental study. *International Journal of Intelligent Systems*, 16:1445–1473, 2001.
- [6] J.C. Bezdek and N.R. Pal. Some new indexes of cluster validity. *IEEE Transactions on Systems, Man, and Cybernetics—Part B*, 28(3):301–315, 1998.

- [7] J.C. Bezdek, T.R. Reichherzer, G.S. Lim, and Y. Attikiouzel. Multiple-prototype classifier design. *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, 28(1):67–79, 1998.
- [8] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
- [9] H. Brighton and C. Mellish. Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, 6:153–172, 2002.
- [10] V. Cerverón and F.J. Ferri. Another move toward the minimum consistent subset: A tabu search approach to the condensed nearest neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 31(3):408–413, 2001.
- [11] C-L. Chang. Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers*, 23(11):1179–1184, November 1974.
- [12] D. Chaudhuri, C.A. Murthy, and B.B. Chaudhuri. Finding a subset of representative points in a data set. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(9):1416–1424, 1994.
- [13] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.
- [14] B.V. Dasarathy, editor. *Nearest neighbor (NN) norms: NN pattern classification techniques*. IEEE Computer Society Press, Los Alamitos, CA, 1991.
- [15] B.V. Dasarathy. Minimal consistent set (MCS) identification for optimal nearest neighbor decision systems design. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(3):511–517, 1994.
- [16] B.V. Dasarathy, J.S. Sánchez, and S. Townsend. Nearest neighbor editing and condensing tools – synergy exploitation. *Pattern Analysis and Applications*, 3(1):19–30, 2000.
- [17] D.L. Davies and D.W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):224–227, April 1979.
- [18] J. C. Dunn. Well separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4:95–104, 1974.

- [19] B. Efron and R. Tibshirani. Improvements on cross-validation: The .632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548–560, 1997.
- [20] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.
- [21] E. Fix and J.L. Hodges. Discriminatory analysis: Nonparametric discrimination: Consistency properties. *USAF School of Aviation Medicine*, Project 21-49-004(Report Number 4):261–279, 1951.
- [22] V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining very large databases. *IEEE Computer*, 32(8):38–45, 1999.
- [23] G.W. Gates. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, 18(3):431–433, 1972.
- [24] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.
- [25] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, 1997.
- [26] Y. Hamamoto and S. Uchimura and S. Tomita. A bootstrap technique for nearest neighbor classifier design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(1):73–79, January 1997.
- [27] P.E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(3):515–516, 1968.
- [28] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.
- [29] R.J. Henery. *Machine Learning, Neural and Statistical Classification*, chapter 7, pages 107–124. Ellis Horwood, 1994.
- [30] A.E. Hoerl and R.W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.
- [31] L.J. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2:193–218, 1985.

- [32] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall Inc., New Jersey, 1988.
- [33] A.K. Jain and D.E. Zongker. Feature selection: Evaluation, application, and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):153–158, February 1997.
- [34] R. Kohavi and G.H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97:273–324, December 1997.
- [35] T. Kohonen. Improved versions of learning vector quantization. In *Proceedings of the International Joint Conference on Neural Networks*, volume I, pages 545–550, San Diego, CA, 1990.
- [36] L.I. Kuncheva and J.C. Bezdek. Presupervised and postsupervised prototype classifier design. *IEEE Transactions on Neural Networks*, 10(5):1142–1152, sep 1999.
- [37] L.I. Kuncheva and J.C. Bezdek. Nearest prototype classification: Clustering, genetic algorithms, or random search. *IEEE Transactions on Systems, Man, and Cybernetics*, 28(1):160–164, 1998.
- [38] W. Lam, C.K. Keung, and C.X. Ling. Learning good prototypes for classification using filtering and abstraction of instances. *Pattern Recognition*, 35(7):1491–1506, July 2002.
- [39] W. Lam, C.K. Keung, and D. Liu. Discovering useful concept prototypes for classification based on filtering and abstraction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(8):1075–1090, August 2002.
- [40] P. Mitra, C.A. Murthy, and S.K. Pal. Density-based multiscale data condensation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(6):734–747, 2002.
- [41] R.A. Mollineda, F.J. Ferri, and E. Vidal. An efficient prototype merging strategy for the condensed 1-NN rule through class-conditional hierarchical clustering. *Pattern Recognition*, 35:2771–2782, 2002.
- [42] G.L. Ritter, H.B. Woodruff, S.R. Lowry, and T.L. Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, 21(6):665–669, 1975.
- [43] C. Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993.

- [44] V.G. Sigillito, S.P. Wing, L.V. Hutton, and K.B. Baker. Classification of radar returns from the ionosphere using neural networks. *John Hopkins APL Technical Digest*, 10:262–266, 1989.
- [45] D.B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In W. Cohen and H. Hirsch, editors, *Proceedings of the 11th International Conference on Machine Learning*, pages 293–301, New Brunswick, New Jersey, 1994.
- [46] C.W. Swonger. Sample set condensation for a condensed nearest neighbor decision rule for pattern recognition. *Frontiers of Pattern Recognition*, pages 511–519, 1972.
- [47] I. Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 6(6):448–452, 1976.
- [48] C.J. Veenman, M.J.T. Reinders, and E. Backer. A maximum variance cluster algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1273–1280, September 2002.
- [49] C.J. Veenman, M.J.T. Reinders, and E. Backer. A cellular coevolutionary algorithm for image segmentation. *IEEE Transactions on Image Processing*, 12(3):304–316, March 2003.
- [50] G. Wilfong. Nearest neighbor problems. In *Proceedings of the seventh annual symposium on Computational geometry*, pages 224–233. ACM Press, 1991.
- [51] D.L. Wilson. Assymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, 2(3):408–421, July 1972.
- [52] D.R. Wilson and T.R. Martinez. Instance pruning techniques. In D. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 404–411. Morgan Kaufmann Publishers, 1997.
- [53] D.R. Wilson and T.R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38:257–286, 2000.
- [54] W.H. Wolberg and O.L. Mangarasian. Multisurface method of pattern separation for medical diagnoses applied to breast cytology. *Proceedings of the National Academy of Sciences*, 87:9193–9196, December 1990.