International Journal on Artificial Intelligence Tools
Vol. 14, Nos. 1 & 2 (2005) 261–280

World Scientific
www.worldscientific.com

PROTOTYPE CLASSIFIER DESIGN WITH PRUNING

JIANG LI*, MICHAEL T. MANRY, CHANGHUA YU
Electrical Engineering Department, University of Texas at Arlington
Arlington, Texas, 76019 USA
**li@wcn.uta.edu*

D. RANDALL WILSON
Computer Science Department, Brigham Young University, Provo, Utah, 84602 USA
randy@axon.cs.byu.edu

Received 15 July 2004
Revised 11 December 2004

Algorithms reducing the storage requirement of the nearest neighbor classifier (NNC) can be divided into three main categories: Fast searching algorithms, Instance-based learning algorithms and Prototype based algorithms. We propose an algorithm, LVQPRU, for pruning NNC prototype vectors and a compact classifier with good performance is obtained. The basic condensing algorithm is applied to the initial prototypes to speed up the learning process. The learning vector quantization (LVQ) algorithm is utilized to fine tune the remaining prototypes during each pruning iteration. We evaluate LVQPRU on several data sets along with 12 other algorithms using ten-fold cross-validation. Simulation results show that the proposed algorithm has high generalization accuracy and good storage reduction ratios.

Keywords: Prototype selection; nearest neighbor classifier; editing; condensing; instance-based learning, pruning technology.

1. Introduction

The nearest neighbor classifier (NNC) is used for many pattern recognition applications where the underlying probability distribution of the data is unknown *a priori*. The behavior of the NNC is bounded by two times the optimal Bayes risk (Ref. 1). Since the traditional NNC stores all the known data points as labelled instances, the algorithm is prohibitive for very large databases. To overcome these challenges several techniques have been proposed by researchers. $k-d$ *trees* (Ref. 2) and *projection* (Ref. 3) can reduce the search time for the nearest neighbors but still do not decrease storage requirements, and they become less effective as the dimensionality of the data increases.

To reduce both the storage requirements and search time, a better way is to reduce the data size under the restriction that the classification accuracy is kept similar (or even higher if noisy instances are removed). To select a subset of instances from the original training set, there are usually three types of search directions to perform the selection procedure, including *Incremental*, *Decremental* and *Batch*. An *Incremental* search algorithm begins with an empty subset and adds an instance if it satisfies some criteria. For example, IB1 through IB5 (Ref. 4, 5) belong to this category. A *Decremental* search algorithm, however, begins with the full training set and removes instances until some predefined condition is fulfilled. Reduced nearest neighbor (RNN) (Ref. 6, 7), condensing nearest neighbor (CNN) (Ref. 1, 8), DROP1 through DROP5 (Ref. 9, 10), editing nearest neighbor (ENN) and repeated ENN (RENN) (Ref. 11) can be viewed as *decremental* search methods. The ENN and CNN algorithms can delete outliers or internal instances to improve the generalization accuracy and the processing speed. The performance can be further improved by employing adaptive NNC (Ref. 12). The All k-NN rule (Ref. 13) is an example of a *batch* search algorithm, in which the potential instances to be removed are flagged first. The actual deletion of the flagged instances is performed until finishing passing through all instances in the data.

The above classes of algorithms do not modify instances, but merely remove instances that appear to be noisy or redundant. One also can generate prototypes by modifying data instances such as in Chang's method (Ref. 14) or the Self-Organizing Map (SOM) (Ref. 15). Recently developed algorithms for reducing data size include boosting-based algorithms (Ref. 16, 17, 18, 19), Genetic algorithms (Ref. 20) and statistical pruning (Ref. 21).

Reducing data size for classification tasks is one application of case-based reasoning (CBR). CBR systems solve problems by retrieving and adapting the solutions to similar problems that have been previously stored as a case base. CBR deals with a wide range of problem-solving tasks such as prediction, planning, decision support, classification, etc. There are many approaches proposed in CBR community to obtain a concise case base with good quality and to manage the case base over time. This paper discusses reducing data size based on prototype selection algorithms for classification problem. Readers interested in CBR systems can refer to the papers by Yang & Zhu (Ref. 22) and by Portinale & Torasso (Ref. 23).

Generally, there are some problems a designer must face when using prototype-based algorithms. Consider the artificially constructed example in Fig. 1 where the probability density functions of both inputs are uniform. It is obvious that the classification error is minimized and a perfect decision boundary is defined with only one prototype located in the middle of C_1 and two prototypes in C_2. Two challenges are: 1) the number of prototypes necessary for a classifier is difficult to determine and 2) the optimal placement of these prototypes is not obvious. Note that in this example the ideal locations of the prototypes are not unique, we can move them and keep the decision boundary fixed and the classification error remains unchanged. In order to overcome these challenges, we propose an algorithm that prunes prototypes based on the error their elimination produces. LVQ2.1 (Ref. 24) is used to fine-tune the placements of the prototypes, and the number of prototypes needed by a NNC is determined by the *structural risk minimization* principle that is introduced later in the paper.

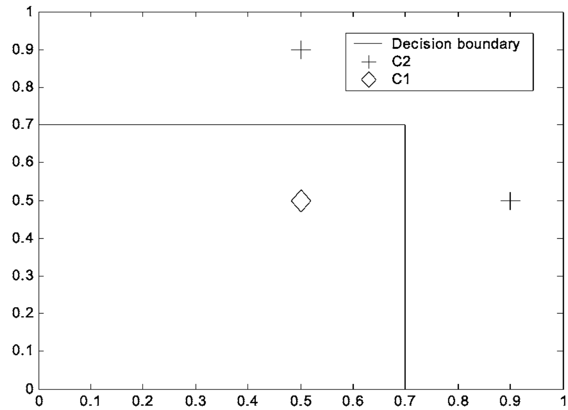

Fig. 1. Uniform Data Example

The rest of paper is organized as follows. Section 2 describes some existing instance-based algorithms. Section 3 introduces a distance measure which is used for measuring the similarity between two instances or prototypes. Section 4 presents details of the proposed algorithm. LVQPRU is demonstrated in Section 5. Section 6 presents empirical results comparing 12 existing algorithms with the LVQPRU method on 13 data sets. Finally, Section 7 gives conclusions and future work.

2. Related Work

Many researchers have investigated instance-based learning algorithms for training set size reduction, i.e., searching for a subset $\mathcal{S}$ of instances to keep from training set $\mathcal{D}$. In this section we give a briefly introduction of algorithms that will be compared with our proposed algorithm. Interested readers can find details in Wilson & Martinez (Ref. 10).

2.1. *Decremental Instance-Based Learning Algorithms*

Common decremental instance-based learning algorithms include CNN, ENN and the series of *Decremental Reduction Optimization Procedure* (DROP) algorithms proposed by Wilson & Martinez (Ref. 9, 10).

The basic idea of CNN (Ref. 1, 8) is to delete internal or central instances such that the subset $\mathcal{S}$ can be used to classify all the instances in $\mathcal{D}$ correctly. The algorithm begins by randomly selecting one instance belonging to each class from $\mathcal{D}$ and putting them into $\mathcal{S}$. Then using the instances in $\mathcal{S}$ to classify one instance in $\mathcal{D}$, if the instance is misclassified, it is added to $\mathcal{S}$. Otherwise, it is returned to $\mathcal{D}$. This process is repeated until either there is no instance left in $\mathcal{D}$ or there is no instance transferring from $\mathcal{D}$ to $\mathcal{S}$. The ENN (Ref. 11) algorithm begins with $\mathcal{S} = \mathcal{D}$, and then each instance in $\mathcal{S}$ is removed if it does not agree with the majority of its k nearest neighbors. Unlike the CNN algorithm, ENN keeps all the internal instances but deletes the border instances as well as the noisy instances. RENN applies ENN repeatedly until there are no instances in $\mathcal{S}$ that can be removed.

The aim of DROP algorithms is to find instance reduction techniques that provide noise tolerance, high generalization accuracy, insensitivity to the order of presentation of instances, and significant storage reduction. DROP1 is identical to RNN (Ref. 6, 7) except that the accuracy is evaluated on $\mathcal{S}$ instead of $\mathcal{D}$. DROP2 considers the effect of the removal of an instance on all the instances in $\mathcal{D}$ instead of considering only those instances remaining in $\mathcal{S}$. DROP3 uses a noise-filtering pass first. This is done using a rule similar to the ENN: Any instance misclassified by its k nearest neighbors is removed. This allows points internal to clusters to be removed early in the process, even if there were noisy points nearby. DROP4 is identical to DROP3 except that instead of blindly applying ENN, the noise-filtering pass removes each instance only if it is (1) misclassified by its k nearest neighbors, and (2) it does not hurt the classification of other instances.

2.2. *Incremental Instance-Based Learning Algorithms*

Aha et al. (Ref. 4, 5) proposed a series of Instance-Based Learning algorithms: IB1 through IB5. IB1 is just simple 1-NN algorithm and is used as a benchmark. IB2 is very similar to Hart's CNN rule, except that IB2 does not first pick one instance from each class to add to $\mathcal{S}$. It begins with an empty $\mathcal{S}$, and each instance in $\mathcal{D}$ is added to $\mathcal{S}$ if it is not classified correctly by the instances already in $\mathcal{S}$. This algorithms also retains the border instance and removes the central points. Like the CNN algorithm, IB2 is very sensitive to noisy instances. The IB3 algorithm (Ref. 4, 5) tries to solve the IB2's problem of keeping noisy instances by retaining only *acceptable* instances. An instance is *acceptable* if the lower bound on its accuracy is significantly higher at a 90 % confidence level than the upper bound on the frequency of its class. IB3 reduces the sensitivity to noisy instances seen in IB2, and achieves higher accuracy as well as greater reduction of instances than IB2. The IB4 and IB5 (Ref. 5) algorithms are to address irrelevant attributes and to handle the addition of new attributes to the problem after training has already begun respectively, and they are both extensions of IB3.

2.3. *Batch Instance-Based Learning Algorithms*

The *All* k-NN algorithm is an example of the *batch* learning algorithm. Here we also include the Encoding Length and Explore algorithms (Ref. 25) as batch algorithms even though they do not perform a strictly batch search.

Tomek (Ref. 13) extended the ENN to his *All* k*-NN* method of editing. Starting from $\mathcal{S} = \mathcal{D}$, this algorithm works as follows: for $i = 1$ to k, flag any instance as bad if it is not classified correctly by its i nearest neighbors. After completing the loop for all the instances, remove any instances from $\mathcal{S}$ which are flagged as bad. Cameron-Jones proposed an *Encoding Length Heuristic* (Ref. 25) to evaluate how good the subset $\mathcal{S}$ represents $\mathcal{D}$. The basic algorithm starts with a growing phase followed by a pruning phase. In the growing phase $\mathcal{S}$ begins with an empty subset, and each instance in $\mathcal{D}$ is added to it if that results in a lower cost than not adding it. After all instances in $\mathcal{D}$ are presented once, the pruning phase starts. Each instance in $\mathcal{S}$ is removed if doing so will lower the cost function. By adopting the terminology of Wilson & Martinez (Ref. 10) we call this algorithm the *Encoding Length Growing* (ElGrow) method. The *Explore* method (Ref. 25) begins by growing and pruning $\mathcal{S}$ using the ELGrow method, and then performs 1000 mutations to improve the classifier. Each mutation tries to add an instance to $\mathcal{S}$, removing one from $\mathcal{S}$ or swapping one in $\mathcal{S}$ with one in $\mathcal{D}$-$\mathcal{S}$, and retains these changes if they do not increase the cost function.

3. A Weighted Distance Measure

In this section, a distance measure is introduced which can suppress random or useless features in the input vector. Training data sometimes contains inputs, which are either useless or random. When the standard Euclidean distance is used in clustering such data during NNC training, this can lead to many more prototypes than is necessary. A variety of distance functions have been developed for continuously-valued attributes, including the Minkowsky (Ref. 26), the Context-Similarity measure (Ref. 27), the optimal distance measure (Ref. 28) and others. Euclidean is common for applications with linear attributes, *overlap* metric (Ref. 4) and Value Difference Metric (VDM) (Ref. 29) are useful for nominal, and the Heterogeneous Value Difference Metric (HVDM) (Ref. 10) is useful for applications with both.

Consider a training set $\mathcal{D} = \{\mathbf{x}_p, i_p\}_{p=1}^{N_v}$ for NNC design, where for the pth instance, $\mathbf{x}_p \in \mathbf{R}^N$ and i_p is the integer class label associated with $\mathbf{x}_p$. N_v is the total number of instances. Here we develop a weighted distance measure directly based on the available data with continuous attributes in the form

$$d(\mathbf{x}_p, \mathbf{m}_k) = \sum_{j=1}^{N} w(j)[x_p(j) - m_k(j)]^2 \tag{1}$$

where $\mathbf{m}_k$ is the kth prototype generated by SOM. We first design a simple classifier such as the functional link network (FLN) by minimizing

$$E = \frac{1}{N_v} \sum_{i=1}^{N_c} \sum_{p=1}^{N_v} (t_p(i) - t'_p(i))^2 \tag{2}$$

where N_c is the number of classes, $t_p(i)$ denotes the ith desired output for the pth input vector $\mathbf{x}_p$. Specifically, $t_p(i_c) = 1$ and $t_p(i_d) = 0$, where i_c denotes the correct class number for $\mathbf{x}_p$, and i_d denotes any incorrect class number for $\mathbf{x}_p$.

The ith observed output of the classifier for $\mathbf{x}_p$ can be written as

$$t'_p(i) = \sum_{j=1}^{N_u} w_o(i,j) X_p(j) \tag{3}$$

where $w_o(i,j)$ denotes the weight connecting the jth unit to the ith output unit. $X_p(j)$ denotes the jth basis function for the pth pattern, and N_u is the number of basis functions. In an FLN, $X_p(j)$ often represents a multinomial combination of N elements of $\mathbf{x}_p$, The following theorem provides the basis for determining useful distance measure weights from training data.

Theorem 3.1. For a given training set $\mathcal{D}$, convert i_p to $\mathbf{t}_p$, and use vector $\mathbf{t}(\mathbf{x})$ to denote the desired output. Let $\hat{\mathbf{t}}(\mathbf{x})$ denote the minimum mean-square error (MMSE) estimate of the desired output vector $\mathbf{t}(\mathbf{x})$. Assume that $x(j)$, the jth element of input vector $\mathbf{x}$, is statistically independent of $\mathbf{t}(\mathbf{x})$ and the other elements of the input vector. Then the derivative of $\hat{\mathbf{t}}(\mathbf{x})$ with respect to $x(j)$ is zero for all $\mathbf{x}$.

Proof. The MMSE estimate of $\mathbf{t}(\mathbf{x})$ is

$$\hat{\mathbf{t}}(\mathbf{x}) = \mathbf{E}[\mathbf{t}|\mathbf{x}] = \int_{-\infty}^{\infty} \mathbf{t}\mathbf{f_t}(\mathbf{t}|\mathbf{x})dt \tag{4}$$

where $\mathbf{f_t}(\mathbf{t}|\mathbf{x})$ denotes the joint probability density of the desired output vector $\mathbf{t}$ conditioned on $\mathbf{x}$. Using Bayes law,

$$\mathbf{f_t}(\mathbf{t}|\mathbf{x}) = \frac{\mathbf{f_{t,x}}(\mathbf{t},\mathbf{x})}{\mathbf{f_x}(\mathbf{x})} \tag{5}$$

Letting $\mathbf{x}'$ denote $\mathbf{x}$ without the element $x(j)$,

$$\begin{aligned}\mathbf{f_t}(\mathbf{t}|\mathbf{x}) &= \frac{\mathbf{f_{t,x'}}(\mathbf{t},\mathbf{x}')f_{x(j)}(x(j))}{\mathbf{f_{x'}}(\mathbf{x}')f_{x(j)}(x(j))}\\ &= \frac{\mathbf{f_{t,x'}}(\mathbf{t},\mathbf{x}')}{\mathbf{f_{x'}}(\mathbf{x}')}\\ &= \frac{\mathbf{f_t}(\mathbf{t}|\mathbf{x}')\mathbf{f_{x'}}(\mathbf{x}')}{\mathbf{f_{x'}}(\mathbf{x}')} = \mathbf{f_t}(\mathbf{t}|\mathbf{x}')\end{aligned}$$

Now the derivative of $\hat{\mathbf{t}}(\mathbf{x})$ with respect to $x(j)$ is

$$\begin{aligned}\frac{\partial\hat{\mathbf{t}}(\mathbf{x})}{\partial x(j)} &= \int_{-\infty}^{\infty}\frac{\partial}{\partial x(j)}[\mathbf{t}\mathbf{f_t}(\mathbf{t}|\mathbf{x}')]dt\\ &= \mathbf{0}\end{aligned}$$

We complete the proof. □

Corollary. Given the assumptions of **Theorem** 3.1,

$$\mathbf{E}\left[\left|\left|\frac{\partial\hat{\mathbf{t}}(\mathbf{x})}{\partial x(j)}\right|\right|\right] = \mathbf{0} \tag{6}$$

where $||\cdot||$ denotes the $\mathbf{L}_1$ norm.

Now we train a functional link network network, whose output for the pth pattern is denoted by $\hat{\mathbf{t}}_p$. The corollary above then implies that $u(j) \simeq 0$ where

$$u(j) = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{N_c} \left| \frac{\partial \hat{t}_p(i)}{\partial x_p(j)} \right| . \tag{7}$$

As a heuristic, the distance measure's weights are determined as

$$w(j) = \frac{u(j)}{\sum_{n=1}^{N} u(n)} . \tag{8}$$

$u(j)$, which represents the importance of $x(j)$ to the network outputs, is normalized to yield $w(j)$, which is used in (1) when calculating a distance.

4. NNC Pruning Algorithm

In this section, we describe the LVQPRU algorithm. First a list of the algorithm steps is presented, and then explanations for those steps are given.

4.1. *Algorithm Outline*

Suppose a testing data set with the same format as that of the training data set is available. Otherwise, the training data set could be divided to two parts for training and testing. The pruning algorithm is described as follows,

(1) Let N_{pc} be the number of prototypes per class, with N_{pc} sufficiently large
(2) Randomly initialize these $N_{tc} = N_{pc} \cdot N_c$ prototypes, and train a separate SOM network with N_{pc} prototypes for each class in the training data set. Denote the number of instances closest to the kth prototype as $N_v(k)$, where $1 \le k \le N_{tc}$
(3) Delete the kth prototype if it does not contain any members (empty prototypes), i.e., $N_v(k) = 0$, and decrease the total number of prototypes N_{tc} by 1
(4) Change the class label of a prototype if it disagrees with the plurality of the instances closest to it
(5) Use LVQ2.1 (Ref. 24) to refine the locations of the prototypes
(6) Apply the basic condensing algorithm to the remaining prototypes to remove the internal ones
(7) Prune the prototype whose removal causes the least increase in classification error, and set $N_{tc} = N_{tc} - 1$
(8) Use LVQ2.1 to fine-tune the locations of the remaining prototypes
(9) Apply the trained network to the testing data set to obtain the testing error
(10) If $N_{tc} > N_c$ go back to (7). Otherwise if $N_{tc} = N_c$, identify the network with the minimum testing error and stop.

The network corresponding to the minimum testing error is the final NNC classifier. In steps (5) and (8), we stop LVQ if it has run a predefined number of iterations (e.g, 10 in this paper) or if the classification error for the training data set increases, whichever comes first. If the testing error increases at the very beginning and never decreases, it probably means that the initial number of prototypes N_{tc} is not large enough. We then increase this number and rerun the algorithm.

4.2. *Algorithm Details*

In the section we present the details for each step in the algorithm and give some discussion.

4.2.1. *Choosing the Number of Initial Prototypes*

In the first step of the algorithm, determining the exact number of prototypes required for each class to accurately represent the given data is a very difficult model selection procedure, and there is no widely accepted method for doing this. Even though many researchers, for example Yager and Stephen (Ref. 30, 31), have proposed such methods, the user still needs to choose some parameters to initialize the algorithms.

It is well known that for a given finite set of training data, the training error can eventually go to zero if we keep increasing the number of prototypes. However, this "overtraining" or "memorization" decreases the generalization capability (Ref. 32). In our algorithm, instead of giving control parameters before training, we choose the final number of prototypes based on the testing result. This idea is based on the structural risk minimization principle (Ref. 33).

Consider an ensemble of classifiers $\{F(\mathbf{x}, \mathbf{m})\}$, and define a nested structure of n_s ensembles of such classifiers

$$\mathcal{F}_k = \{F(\mathbf{x}, \mathbf{m}), \mathbf{m} \in \mathcal{M}_k\}, k = 1, 2, 3..., n_s \tag{9}$$

where $\mathcal{M}_k$ is a set of parameter vectors, such that we have

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset \mathcal{F}_3 \ldots \subset \mathcal{F}_n. \tag{10}$$

Correspondingly, the complexity of the classifier satisfies the condition

$$h_1 \leq h_2 \leq h_3 \ldots \leq h_n. \tag{11}$$

The structural risk minimization may proceed as follows,

- The training error for each classifier is minimized to obtain $\mathbf{m}$.
- The classifier $\mathcal{F}^*$ with the smallest testing error is identified, which provides the best compromise between the training error and the confidence interval.

Our goal here is to find a classifier such that the testing error reaches its minimum. The principle of structural risk minimization may be implemented in a variety of ways, and in this paper we use a pruning algorithm. The number of prototypes corresponding to the minimum of the test error is chosen as the final model (classifier) for the given data set such that it satisfies the structural risk minimization principle. The initial number of prototypes should be a number that is greater than the final one.

4.2.2. *Training Initial SOM Network for the Data*

Given N_{pc} prototypes for each class, we then train a separate SOM network for each class in step (2). Due to the complexity of the data, there might be some empty prototypes that are not surrounded by data instances. Alternately, a prototype assigned to the ith class may have nearby data instances which belongs to the jth class. For the first case, these empty prototypes ($N_v(k) = 0$) are deleted in step (3). For the second case the category of the prototype needs to be changed as discussed in the following subsection.

4.2.3. *Changing a Prototype's Category*

In order to change the categories of the prototypes in step (4), we count the members of each prototype. If a plurality of the members of a prototype belongs to the ith class, for example, but the class category of that prototype initially is assigned as j, we change the category of the prototype from j to i.

For a NNC, let T_{jk} denote the number of instances from the kth class closest to the jth prototype. Also let $i_c(j)$ denote the class category of the jth prototype. The two-dimensional array containing T_{jk} is generated by the following algorithm.

(1) Set $T_{jk} = 0$ for $1 \le j \le N_{tc}$, $1 \le k \le N_c$
(2) For $p = 1$ to N_v
   (a) Read $\mathbf{x}_p$ and i_p
   (b) Find j such that $d(\mathbf{x}_p, \mathbf{m}_j)$ is minimized. Let k denote the value of i_p
   (c) Accumulate patterns as $T_{jk} \leftarrow T_{jk} + 1$
(3) For $j = 1$ to N_{tc}
   (a) Find k' that maximizes $T_{jk'}$
   (b) If $i_c(j) = k'$, go to 2. Otherwise change $i_c(j)$ to k'
(4) Stop.

4.2.4. *Pruning Prototypes Based on Classification Error*

The goal here is to develop an algorithm to eliminate the least useful prototypes for step (7). Let k be the index of a candidate prototype to be eliminated. Then $Err(k)$ is the number of misclassified data instances after prototype k has been pruned.

(1) Set $Err(k) = 0, 1 \le k \le N_{tc}$
(2) For $p = 1$ to N_v
   (a) Identify two nearest prototypes (whose class category is l and m respectively) to input vector $\mathbf{x}_p$, and let n denote the class label of $\mathbf{x}_p$
   (b) Accumulate errors as
      i. $Err(l) \leftarrow Err(l) + 1$, if $n = l, n \ne m$
      ii. $Err(l) \leftarrow Err(l)$, if $n = l = m$, or $n \ne l \ne m$

iii. $Err(l) \leftarrow Err(l) - 1$, if $n \neq l$, $n = m$

(3) Now find the smallest $Err(k)$ denoted by $Err(k_{min})$ and eliminate the k_{min}th prototype. Note that $Err(k_{min})$ can be negative sometimes.

(4) Stop.

After one prototype has been deleted, we use another epoch of LVQ2.1 to adjust the location of the remaining prototypes. Since LVQ is not guaranteed to converge, we stop it whenever the training error increases or it has run for a predefined number of iterations. This pruning process continues till there are N_c prototypes remaining.

5. Examples

We study the performance of the proposed algorithm on three different data sets: uniformly distributed data, normally distributed data and data from the handwritten numeral recognition problem.

5.1. *Uniformly Distributed Data*

We first apply the proposed algorithm to the artificially constructed data set of Fig. 1, which contains 1000 instances. Here 400 belong to C_1 and 600 belong to C_2, and both inputs have a uniform distribution. A testing data set which contains 500 instances is also generated. We select the initial number of prototypes as $N_{pc} = 20$ for each class. The "small disks" in Fig. 2 represent prototypes initially generated by SOM, and the "diamonds" represent the pruned result. The 4 squares represent the situation when 4 prototypes remain. Those 4 prototypes form the decision boundary which is the dotted line in the figure. The solid line in the figure denotes the optimal decision boundary for this problem. For this data, the number of prototypes corresponding to the minimum of the testing error is not unique. Actually, a zero valued testing error can always be obtained if the number of prototypes is greater than or equal to 3. From the previous discussion of structural risk minimization, a simpler classifier is always preferred if it can give the same testing error as a bigger one, so the number of prototypes finally chosen is 3. It is concluded from Fig. 2 that the proposed algorithm can find a good solution for this specific example, since the final prototypes form the optimal decision boundary and the testing error is zero. We notice that the performance will degrade a lot if we try to prune even one additional prototype. It is also observed that more prototypes for the data do not guarantee a better solution. In Fig. 2, the remaining 3 prototypes form the optimal decision boundary. However, the decision boundary (the dotted line) formed by the remaining 4 prototypes is not optimal even though the testing error is still zero. The final solution with 3 prototypes is always reached as long as N_v is large enough. Therefore, in this example the storage percentage (number of remaining prototypes divided by N_v) can approach zero.

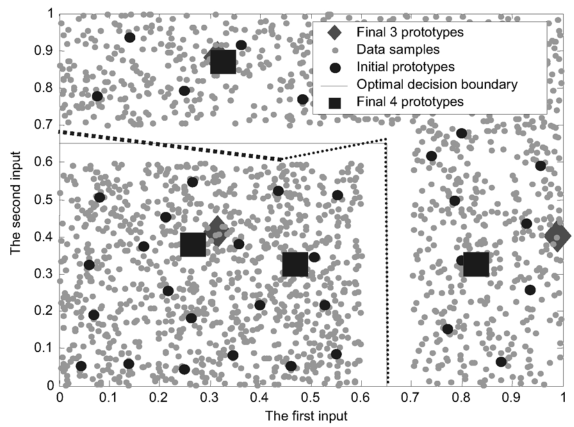

Fig. 2. The Design Result for Uniformly Distributed Data

5.2. *Normally Distributed Data*

The second experiment is performed on a normally distributed data set. As seen in Fig. 3 (a), the data instances from both classes have different mean and variance. We denote the instances outside the circle as class C_1 and those inside the circle as class C_2. The probability density function for the ith class is

$$f_{x,y}(x,y|C_i) = \frac{1}{2\pi\sigma_i^2} exp\{-(\frac{(x-m_{x_i})^2}{2\sigma_i^2} + \frac{(y-m_{y_i})^2}{2\sigma_i^2})\} \tag{12}$$

where $i = 1,2$, m_{x_i}, m_{y_i} are the means for the ith class, and σ_i is the standard deviation for the ith class. In this example, class C_1 has zero mean and $\sigma_1 = 0.5$, and the mean of class C_2 is $[m_{x_i}, m_{y_i}] = [2,2]$ and $\sigma_2 = 1$. Each class contains 20000 instances, but only 4000 of them for each class are plotted in Fig. 3 (a) for clarity. The optimal decision boundary calculated by Bayes decision rule for this example is

$$(x+\frac{2}{3})^2 + (y+\frac{2}{3})^2 = 4.48 \tag{13}$$

which is the circle plotted in Fig. 3. If a data instance is outside the circle, we decide it is from class C_1, otherwise it is from class C_2.

In this example we illustrate the pruning process in detail. The starting number of prototypes N_{pc} for each class is 30. In Fig.3 (b) we plot the initial prototypes N_{tc}, which is equal to 60, generated by the SOM algorithm. After applying the condensing algorithm to the 60 initial prototypes there are only 6 of them left as shown in Fig. 3 (c). In Fig. 3 (c) we illustrate the pruning process when the total number of remaining prototypes N_{tc} varies from 6 to 2. Fig. 3 (d) is the pruning process results (Ref. 34) in which the condensing algorithm was not used. The testing results corresponding to each network are listed in Table 1. The first row denotes the number of remaining prototypes, the second and the third rows represent the training and testing results (the values represent classification error in %) for the corresponding network respectively. The testing data contains the same number of instances as the training data but the two data sets have no patterns in common.

Remarks:

(1) The initial prototypes (Fig. 3 (b)) can approximately represent the probability distribution of the data. This agrees with the result of (Ref. 35).
(2) The basic condensing algorithm is applied directly to the initial prototypes without passing through the data. These prototypes work as a compressed version of the original data.
(3) Comparing Fig. 3 (b) with the first subgraph in Fig. 3 (c), the condensing algorithm does not delete all the internal prototypes but does remove some border prototypes, both of which are undesirable. These problems result from the fact that the basic condensing algorithm is sensitive to the data presentation order (Ref. 10).
(4) The final five prototypes do not represent the probability distribution of the data. Instead, they approximate the optimal Bayes decision boundary. The number of prototypes needed for the final network is determined by the data itself.
(5) Fig. 3 (d) shows the pruning process without the condensing algorithm. The final network determined by the algorithm also has 5 prototypes with a testing error 2.42% (Ref.34) which is slightly better. This performance difference is due to drawbacks of the basic condensing algorithm, and it should be overcome if a better condensing algorithm is used. Note that the computation has been sped up significantly since the pruning process starts with only 6 prototypes.

Under certain conditions, a multilayer perceptron (MLP) with sigmoid activation functions in the hidden layer can approximate a Bayes classifier (Ref. 36). We run the back propagation (BP) algorithm (Ref. 37) in a three layer MLP and compare it to the NNC. Using the early stopping method (Ref. 38), we first determine the iteration number for a given number of hidden units and then increase the number of hidden units until the testing error starts to increase. The best testing performance for the MLP, 2.42%, is observed when it has 3 hidden units with 180 training epochs. Note that the theoretical Bayes error percentage for this data is 2.41%. Thus, if the data is normally distributed, both the NNC and MLP classifiers can approach the optimal Bayes classifier. However, when data is not normally distributed, NNC is often employed since it is not based on any model. We thus test the NNC design on a real handwritten data set with features that are not normally distributed.

5.3. *Handwritten Numeral Data Set*

The raw data consists of images from handwritten numerals collected from 3,000 people by the Internal Revenue Service. We randomly chose 300 characters from

Table 1. Training and testing error percentage of normal distributed data for various networks N_{tc}.

N_{tc}	2	3	4	5	6
Training Error (%)	2.97	2.53	2.53	2.48	2.54
Testing Error (%)	2.76	2.49	2.48	2.44	2.47

each class to generate training data with 3,000 characters (Ref. 39). Images are 32 by 24 binary matrices. The feature set contains 16 elements and the 10 classes correspond to the 10 Arabic numerals. Using the same method we generated a testing data set that contains 3000 instances.

The initial number of prototypes is chosen to be 30 for each class so that the total initial number of prototypes is $N_{tc} = 300$. After deleting empty prototypes, 215 prototypes remain, and 107 of them are left after applying the basic condensing algorithm to the 215 prototypes. The training and testing results during the pruning process are plotted in Fig. 4. The best testing classification error of 8.83% is obtained when 62 prototypes remain. It is observed that if the number of prototypes increases the training error can decrease. However, the testing results become worse, which indicates that over training has occurred.

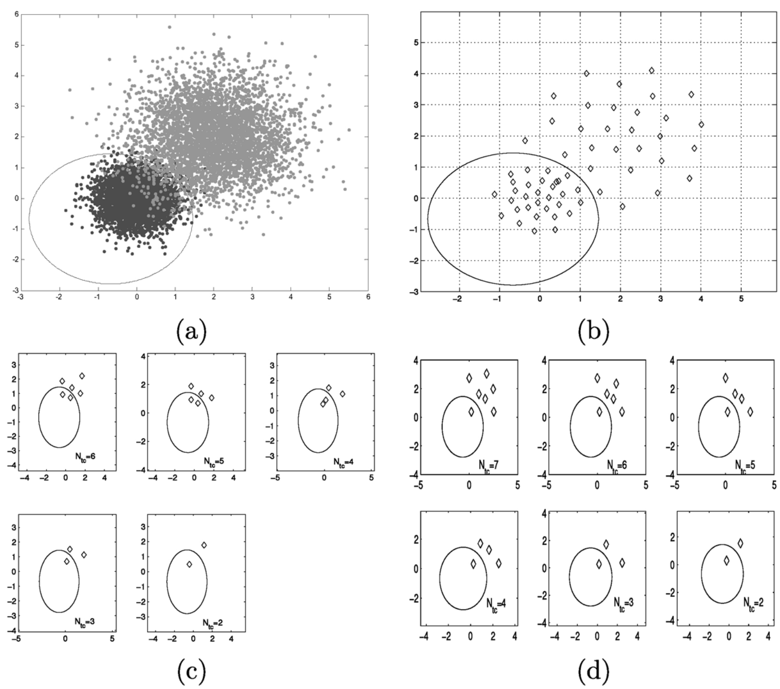

Fig. 3. Normal Distributed Data (a) The original data; (b) The initial 60 prototypes generated by the SOM; (c) The pruning process with condensing algorithm; (d) The pruning process without condensing algorithm.

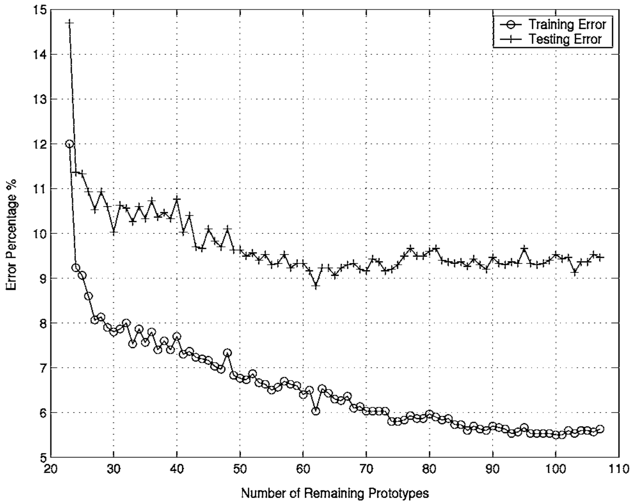

Fig. 4. Training and Testing Results for Handwritten Data

To illustrate the advantages of the proposed algorithm, we compare the designed network ($N_{tc} = 62$) to a directly designed network in which the pruning and condensing steps have been eliminated. The final network has 63 prototypes and the testing performance is 11.26% which is much worse than that of the pruned network even though they have almost the same number of prototypes.

An MLP classifier trained using BP is also compared to the LVQPRU algorithm. It is found that when the number of hidden units is 20 with 550 iterations of training, the MLP gets the best testing result. After training the MLP with 50 different sets of initial weights, the best testing result we get is 9.87%.

6. Evaluation of the Algorithm

In this section, the ten-fold cross-validation technique is used to evaluate the proposed algorithm. The computational complexity is also discussed.

6.1. *Cross-Validation*

We test the proposed algorithm with many other instance-based algorithms on 13 data sets from the machine learning database repository at the University of California, Irvine (Ref. 40). Data sets that have missing values are not considered in this paper. The algorithms we compared include four groups as described by Wilson (Ref. 10): The first group consisting of algorithms which do not have high accuracy, include CNN, IB2 and DROP1. The second group of algorithms has similar noise-filtering properties, high accuracy, and uses most of the instances. These include ENN, RENN and ALLKNN. The third group consists of two aggressive storage reduction algorithms: ELGrow and Explore. They can achieve reasonably good accuracy with only about 2% of the data. The 1-NN algorithm is also included in this group as a baseline. The final group consists of some advanced algorithms which have high accuracy and reasonably good storage reduction. These include IB3, DROP3 and DROP4.

All of the algorithms use k=1, and all except our LVQPRU algorithm use the HVDM distance measure (Ref. 10). LVQPRU uses the weighted distance measure discussed in section 3.

Ten-fold cross-validation followed by a paired-t test was used for each experiment. In ten-fold cross validation, we first randomly divide the available data into 10 equal-sized parts. Each of the ten parts is held out as a test set, and each pruning technique is given a training set $\mathcal{D}$ consisting of the remaining nine parts, from which it returns a subset $\mathcal{S}$ (or a set of prototypes from LVQPRU). The held out set is classified using only the instances or prototypes in $\mathcal{S}$. The training and testing procedure are repeated 10 times using a different one of the 10 parts as the test set. The average testing accuracy over the 10 runs is reported for each algorithm on each data set in tables 2 through 5. The average percentage of instances (or prototypes) in $\mathcal{D}$ that were included in the $\mathcal{S}$ is also reported in the tables. The average accuracy and storage percentage for each method over all of the 13 data sets is shown in **bold** at the bottom of the tables.

Table 2. Comparison between LVQPRU and some historical algorithms.

Database	CNN (Acc.)	SP (%)	IB2 (Acc.)	SP (%)	DROP1 (Acc.)	SP (%)	LVQPRU (Acc.)	SP (%)
Australian	73.04	25.23	72.17	25.46	*82.32*	8.81	**85.12**	**1.60**
Glass	*70.95*	37.96	**71.90**	38.16	*65.02*	21.39	*62.16*	**6.39**
Heart	74.07	27.74	74.07	27.74	75.19	12.76	**85.41**	**2.61**
Image segment	*91.43*	15.66	*91.91*	15.29	86.91	8.41	**92.17**	**4.93**
Ionosphere	81.20	23.04	81.20	23.04	84.04	6.36	**94.70**	**4.27**
Iris	91.33	12.44	91.33	12.44	78.00	8.67	**97.91**	**2.45**
Liver-bupa	60.34	40.26	60.34	40.26	63.66	20.26	**78.40**	**7.62**
Led-creator	63.50	36.11	64.30	36.11	71.90	10.04	**75.48**	**2.14**
Pima-indians	64.58	34.13	64.58	34.13	71.35	16.32	**78.68**	**1.82**
Vehicle	64.66	38.69	64.66	38.69	63.35	16.52	**77.55**	**4.58**
Vowel	**94.70**	22.73	**94.70**	22.73	90.34	30.37	*94.52*	**17.64**
Wine	92.71	14.98	92.71	14.98	88.17	5.43	**98.40**	**3.62**
Zoo	**98.89**	10.13	**98.89**	10.13	93.33	17.41	*96.91*	**9.03**
Average	**78.57**	**26.08**	**78.67**	**26.09**	**77.97**	**14.06**	**85.95**	**5.28**

Note: 'Acc.' denotes the classification accuracy in percent for the testing data, averaged over ten runs. 'SP' denotes the average storage percentage for the ten runs, which is the ratio in percentage of the number of instances (or prototypes) in $\mathcal{D}$ that were included in $\mathcal{S}$.

The **bold** values in each row are the highest accuracy or the lowest storage percentage for each data set obtained by the algorithm(s). A paired t-test is performed between the highest accuracy and each of the other accuracy values. If they are not significantly different at a 90% confidence level, the tested accuracy value is shown

Table 3. Comparison between LVQPRU and three similar noise-filtering algorithms.

Database	ENN (Acc.)	SP (%)	RENN (Acc.)	SP (%)	ALLKNN (Acc.)	SP (%)	LVQPRU (Acc.)	SP (%)
Australian	*85.51*	85.06	*85.08*	84.27	**85.65**	81.22	*85.12*	**1.60**
Glass	*70.61*	71.03	*70.61*	69.94	**71.02**	69.52	*62.16*	**6.39**
Heart	80.00	81.56	80.00	80.70	80.74	76.83	**85.41**	**2.61**
Image segment	**92.62**	92.64	*92.38*	92.56	*91.67*	92.51	*92.17*	**4.93**
Ionosphere	85.18	86.39	84.90	84.71	84.90	86.48	**94.70**	**4.27**
Iris	*96.00*	94.44	*96.00*	94.37	*96.00*	94.44	**97.91**	**2.45**
lliver-bupa	62.91	67.34	64.34	64.99	63.17	62.32	**78.40**	**7.62**
Led-creator	*73.20*	74.83	*73.20*	74.83	*73.60*	72.68	**75.48**	**2.14**
Pima-indians	73.31	74.51	73.44	73.61	72.79	70.30	**78.68**	**1.82**
Vehicle	68.67	74.56	68.44	72.97	68.80	69.98	**77.55**	**4.58**
Vowel	**98.29**	99.03	**98.29**	99.03	**98.29**	99.01	94.52	**17.64**
Wine	94.35	95.32	94.35	95.32	94.35	95.32	**98.40**	**3.62**
Zoo	*95.56*	98.52	*95.56*	98.52	*95.56*	98.52	**96.91**	**9.03**
Average	**82.78**	**84.25**	**82.81**	**83.53**	**82.81**	**82.24**	**85.95**	**5.28**

in italics. The paired t-test is also performed between the lowest storage percentage and the others for each data set. If they are not significantly different at a 90% confidence level, the tested storage percentage is shown in italics.

It is observed in Table 2 that LVQPRU has the highest accuracy for ten of the thirteen data sets. It also results in the greatest data reduction, keeping only 5.28% of the original data, to get an average accuracy of 85.95%. This represents a gain of 7% over the other three of the first group of algorithms. The DROP1 algorithm also has a good data reduction ratio and retains an average of 14.06% of the data instances.

Results for the second group of algorithms are shown in Table 3. For example, ALLKNN has the highest accuracy for three data sets but all the three noise filtering algorithms retain over 80% of the data instances. LVQPRU again has a higher accuracy for 9 data sets while only keeping 5.28% of the original data. The accuracies of this group of algorithms are much higher than those in the first group of algorithms.

Table 4 lists the comparison results between LVQPRN and the third group of algorithms. The basic 1-NN algorithm in table 4 keeps all the data instances for classifying a data instance. Both ELGrow and Explore achieved greater data reduction than LVQPRU, keeping a little more than 2% of the data instances. However, their accuracies are somewhat poor. In contrast, LVQPRU keeps about 5% of the data, but a much higher average accuracy has been achieved. The performance of LVQPRU is even better than that of 1-NN which retains all the instances. This occurs because LVQPRU has removed noisy instances and the remaining prototypes have been optimized so that smoother and more accurate decision boundaries are obtained.

Similar observations are obtained in Table 5, in which the comparison results between LVQPRU and the fourth group of algorithms are listed. LVQPRUN again

Table 4. Comparison between LVQPRU and two *Encoding Length* algorithms.

Database	1-NN (Acc.)	SP (%)	ELGrow (Acc.)	SP (%)	Explore (Acc.)	SP (%)	LVQPRU (Acc.)	SP (%)
Australian	81.45	100.00	*83.19*	*0.35*	**85.51**	**0.34**	*85.12*	1.60
Glass	**72.36**	100.00	63.12	**3.17**	*67.34*	4.57	62.16	6.39
Heart	78.52	100.00	77.78	**0.82**	*84.45*	0.82	**85.41**	2.61
Image segment	**92.86**	100.00	*90.00*	**2.36**	*90.95*	2.49	*92.17*	4.93
Ionosphere	86.32	100.00	77.49	**0.92**	80.34	1.01	**94.70**	4.27
Iris	*94.67*	100.00	90.00	3.18	*96.00*	*2.52*	**97.91**	**2.45**
Liver-bupa	62.91	100.00	62.79	**0.84**	59.06	1.00	**78.40**	7.62
Led-creator	*73.40*	100.00	*73.00*	**1.18**	72.30	*1.23*	**75.48**	2.14
Pima-indians	71.09	100.00	73.45	*0.42*	76.05	**0.39**	**78.68**	1.82
Vehicle	70.93	100.00	61.69	**2.09**	62.29	2.55	**77.55**	4.58
Vowel	**99.24**	100.00	56.83	**6.69**	70.09	9.21	94.52	17.64
Wine	95.46	100.00	91.57	**2.00**	*96.57*	2.25	**98.40**	3.62
Zoo	**98.89**	100.00	*94.44*	**7.90**	*95.56*	8.39	*96.91*	9.03
Average	**82.93**	**100.00**	**76.57**	**2.45**	**79.73**	**2.83**	**85.95**	**5.28**

Table 5. Comparison between LVQPRU and three advanced algorithms.

Database	IB3 (Acc.)	SP (%)	DROP3 (Acc.)	SP (%)	DROP4 (Acc.)	SP (%)	LVQPRU (Acc.)	SP (%)
Australian	84.35	6.25	**86.24**	6.88	*85.07*	8.34	*85.12*	**1.60**
Glass	*67.27*	31.93	**67.31**	19.99	*65.91*	22.12	*62.16*	**6.39**
Heart	75.93	11.44	76.67	11.89	78.52	14.32	**85.41**	**2.61**
Image segment	*90.00*	14.92	**93.10**	8.07	*92.86*	9.60	*92.17*	**4.93**
Ionosphere	88.89	14.24	86.03	7.03	89.45	8.77	**94.70**	**4.27**
Iris	92.66	12.07	94.00	10.30	94.00	10.59	**97.91**	**2.45**
Liver-bupa	54.13	15.27	59.96	24.09	59.96	27.47	**78.40**	**7.62**
Led-creator	*71.90*	24.51	*73.60*	8.94	*73.90*	9.01	**75.48**	**2.14**
Pima-indians	68.48	13.11	72.00	15.13	72.52	17.84	**78.68**	**1.82**
Vehicle	66.21	29.42	69.03	19.88	68.93	21.37	**77.55**	**4.58**
Vowel	94.89	23.65	**96.78**	32.49	96.78	32.49	94.52	**17.64**
Wine	93.79	13.49	94.97	11.11	94.97	11.11	**98.40**	**3.62**
Zoo	**98.89**	17.78	93.33	19.38	93.33	19.38	*96.91*	**9.03**
Average	**80.57**	**17.54**	**81.77**	**15.01**	**82.02**	**16.34**	**85.95**	**5.28**

has higher accuracy for 8 of the data sets and always reduces storage the most. Among all the algorithms, LVQPRU has the highest average accuracy of 85.95%, and the second highest average accuracy of 82.93% is obtained by the basic 1-NN algorithm. For storage reduction, LVQPRU ranks third. However, it still only keeps about 5% of the data which is much less than the other algorithms except for ELGrow and Explore.

6.2. *Complexity of the Algorithm*

The training speed of the algorithm is another issue that needs to be considered. A reasonable (e.g., $O(N_v^2)$ or faster, where N_v is the size of $\mathcal{D}$) time bound is desirable (Ref. 10).

The computational complexity of LVQPRU is not fixed for different data sets. Let the final size of $\mathcal{S}$ be m. Suppose we start the pruning with a size of $\mathcal{S} = 1.5m$, and the average number of runs of LVQ during each pruning iteration is 5 (observed for most cases during the experiments). The algorithm passes through the data $1.5m \cdot 5 = 7.5m$ times. In each pass it computes the distances between the prototypes and each data instance instead of computing the distances among instances, thus saving a lot of computation. This algorithm needs to compute about $7.5m \cdot 1.5m \cdot N_v = 11.25m^2 \cdot N_v$ distances and therefore it takes approximately $O(m^2 \cdot N_v)$ time. Based on tables 2 through 5, the average size for m is about 5% of N_v so that the complexity of the algorithm is reasonable. The above estimates do not consider the condensing algorithm in LVQPRU which also saves computation.

For a very large data base, LVQPRU can be sped up. For example, the original data could be replaced by SOM prototypes before LVQPRU is applied.

7. Conclusions

The basic nearest neighbor algorithm has had many successful applications but suffers from inadequate distance functions, large storage requirements, and a slow search speed. The LVQPRU pruning algorithm combines solutions to each of these problems into a comprehensive system.

LVQPRU uses a distance measure derived from data to suppress random or useless features. We observe that different algorithms are better suited for some problems than others. This suggests that the success of an algorithm is highly dependent on the structure of the data space. A general solution to the problem of optimal instance reduction has to be adapted to the specific structure of the data. We do not have a full understanding of this problem dependency, but our algorithm is a first step in achieving such adaptiveness. LVQPRU determines a distance measure directly from data, and it prunes prototypes, fine tunes the remaining prototypes and selects the final classifier based solely on the data itself. We argue that this procedure may be helpful to extract the information of the data structure and so that the solution is adapted to the data.

To summarize, we introduce an algorithm which performs well over 13 data sets. We have shown that LVQPRU archives better classification accuracies for most of the data sets and selects about 5% of the original data as prototypes. The distance measure used by LVQPRU only deals with continuous attributes, this remains an area of future research of its possible extensions for other kinds of attributes.

Acknowledgments

We thank the anonymous referees for constructive suggestions that helped us in significantly improving the paper from early versions. This work was funded by the Advanced Technology Program of the state of Texas under grant 003656-0129-2001.

References

1. T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Info. Theory*, vol. 13, no. 1, pp. 21–27, 1967.
2. Sproull and F. Robert, "Refinements to nearest neighbor searching in k-dimen. trees," *Algorithmica*, vol. 6, pp. 579–589, 1991.
3. Papadimitriou, H. Christos, and J. L. Bentley, "A worst-case analysis of nearest neighbor searching by projection," *Lecture Notes in Computer Science*, vol. 85, pp. 470–482, 1980, Automata Languages and Programming.
4. D. W. Aha, K. Dennis, and K. A. Marc, "Instance-based learning algorithms," *Machine Learning*, vol. 6, pp. 37–66, 1991.
5. D. W. Aha, "Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms.," *International Journal of Man-Machine Studies*, vol. 36, pp. 267–287, 1992.
6. G. W. Gates, "The reduced nearest neighbor rule," *IEEE Transactions on Information Theory*, vol. IT-18, no. 3, pp. 431–433, 1972.
7. C. Penrod and T. Wagner, "Another look at the edited nearest nearest neighbor rule," *IEEE Trans. Syst., Man, Cyber.*, vol. 7, pp. 92–94, 1977.
8. I. Tomek, "Two modifications of cnn," *IEEE Trans. on Syst., Man., and Cybern.*, , no. SMC-6:, pp. 769–772, 1976.
9. D. R. Wilson and T. T. Martinez, "Instance pruning techniques," *Proceeding of the Fourteenth International Conference on Machine Learning*, pp. 404–411, 1997.
10. D. R. Wilson and T. R. Martinez, "Reduction techniques for instance-based learning algorithms," *Machine Learning*, vol. 38, no. 3, pp. 257–286, 2000.
11. D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 2, no. 3, pp. 408–421, 1972.
12. S. Geva and J. Sitte, "Adaptive nearest neighbor pattern class.," *IEEE Trans. Neur. Net.*, vol. 2, no. 2, pp. 318–322, 1992.
13. I. Tomek, "An experiment with the edited nearest-neighbor rule," *IEEE Trans. on Syst., Man., and Cybern.*, vol. 6, no. 6, pp. 448–452, 1976.
14. Cin-Liang Chang, "Finding prototypes for nearest neighbor classifiers," *IEEE Transactions on Computers*, vol. 23, no. 11, pp. 1179–1184, Nov. 1974.
15. T. Kohonen, *Self-Organizing Maps*, Springer-verlag, Mass., 1995.
16. R. Nock and M. Sebban, "Advances in adaptive prototype weighting and selection," *International Journal on Artificial Intelligence Tools*, vol. 10, no. 1-2, pp. 137–156, 2001.
17. M. Sebban, R. Nock, and S. Lallich, "Stopping criterion for boosting-based data reduction techniques: from binary to multiclass problems," *Journal of Machine Learning Research*, vol. 3, pp. 863–885, Nov. 2002.
18. M. Kubat and M. Cooperson, "Voting nearest-neighbor subclassifiers," *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 503–510, Jun. 2000.
19. M. SEbban, R. Nock, and S. Lallich, "Boosting neighborhood-based classifiers," in *Proceedings of the Eighteenth International Conference on Machine Learning*. 2001, pp. 505–512, Morgan Kaufmann.
20. A. Rozsypal and M. Kubat, "Using the genetic algorithm to reduce the size of a nearest-neighbor classifier and to select relevant attributes," in *Proceedings of the 18th International Conference on Machine Learning*, Williamstown, MA, June 28-July 1 2001, pp. 449–457.
21. V. B. Zubek and T. G. Dietterich, "Pruning improves heuristic search for cost-sensitive

learning," in *Proceedings of the International Conference on Machine Learning*, San Francisco, 2002, pp. 27–34.
22. Q. Yang and J. Zhu, "A case-addition policy for case-base maintenance," *Computationa Intelligence*, vol. 17, no. 2, pp. 250–262, 2001.
23. L. Portinale and P. Torasso, "Case-base maintenance in a multimodal reasoning system," *Computationa Intelligence*, vol. 17, no. 2, pp. 263–279, 2001.
24. T. Kohonen, "Improved versions of learning vector quantization," *IJCNN International Joint Conference on Neural Networks*, pp. 545–550, 1990.
25. R. M. Cameron-Jones, "Instance selection by encoding length heuristic with random mutation hill climbing," *Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence*, pp. 99–106, 1995.
26. B. G. Batchelor, *Pattern Recognition: Ideas in Practice*, Plenum Press, New York, 1978.
27. Y. Biberman, "A context similarity measure," *Proceedings of the European Conference on Machine Learning*, pp. 49–63, 1994.
28. R. D. Short and K. Fukunaga, "The optimal distance measure for nearest neighbor classification," *IEEE Trans. on Information Theory*, vol. IT-27, no. 5, pp. 622–627, Sep. 1981.
29. C. Stanfill and D. Waltz, "Toward memory-based reasoning," *Communications of the ACM*, vol. 29, pp. 1213–1228, 1986.
30. R. Yager and S. P. Filev, "Approximate clustering via the mountain method," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, no. 8, pp. 1279–1284, 1994.
31. L. C. Stephen, "Fuzzy model identification based on cluster estimation," *Journal of Intelligent and Fuzzy Systems*, vol. 2, pp. 267–278, 1994.
32. G. F. Hughes, "On the mean accuracy of statistical pattern recognizers," *IEEE Transactions on Information Theory*, vol. 14, pp. 55–63, 1968.
33. V. N. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995.
34. Jiang Li, Michael T. Manry, and Changhua Yu, "Prototype based classifier design with pruning," *Proceedings of the Seventeenth International Conference of the Florida AI Research Society*, pp. 706–711, May 2004.
35. A.Gersho, "On the structure of vector quantizers," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 157–166, Mar. 1982.
36. E. A. Wan, "Neural network classification: A bayesian interpretation," *IEEE trans. on Neural Network*, vol. 1, no. 4, pp. 303–305, 1990.
37. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Parallel Distributed Processing*, vol. 1, chapter Learning internal representations by error propagation, MIT Press, Cambridge, MA, 1986.
38. M. H. Hassoun, *Fundamentals of Artificial Neural Networks*, Cambridge, MA: MIT Press, 1995.
39. W. Gong, H. C. Yau, and M. T. Manry, "Non-gaussian feature analyses using a neural network," *Progress in Neural Networks*, vol. 2, pp. 253–269, 1994.
40. C. J. Merz and P. M. Murphy, *UCI Repository of Machine Learning Databases*, Irvine, CA: University of California Irvine, Department of Information and Computer Science. Internet: http://www.ics.uci.edu/~mlearn/MLRepository.html, 1996.