Journal of Machine Learning Research 7 (2006) 2125-2148 Submitted 12/05; Revised 5/06; Published10/06

Adaptive Prototype Learning Algorithms: Theoretical and Experimental Studies

Fu Chang FCHANG @IIS.SINICE.EDU.TW
Institute of Information Science
Academia Sinica
Taipei, Taiwan

Chin-Chin Lin ERIKSON@IIS.SINICA.EDU.TW
Department of Electrical Engineering
National Taipei University of Technology
Taipei, Taiwan

Chi-Jen Lu CJLU@IIS.SINICA.EDU.TW
Institute of Information Science
Academia Sinica
Taipei, Taiwan

Editor: Rocco Servedio

Abstract

In this paper, we propose a number of adaptive prototype learning (APL) algorithms. They employ the same algorithmic scheme to determine the number and location of prototypes, but differ in the use of samples or the weighted averages of samples as prototypes, and also in the assumption of distance measures. To understand these algorithms from a theoretical viewpoint, we address their convergence properties, as well as their consistency under certain conditions. We also present a soft version of APL, in which a non-zero training error is allowed in order to enhance the generalization power of the resultant classifier. Applying the proposed algorithms to twelve UCI benchmark data sets, we demonstrate that they outperform many instance-based learning algorithms, the *k*-nearest neighbor rule, and support vector machines in terms of average test accuracy.

Keywords: adaptive prototype learning, cluster-based prototypes, consistency, instance-based prototype, pattern classification

1 Introduction

We divide this section into two parts, with the first part addressing the background of all related methods and the second part discussing our contributions.

1.1 Background

In pattern cognition, one method for classifying objects, expressed as feature vectors, is to compute the distance between the vectors and certain labeled vectors, called prototypes. This approach selects the *k* nearest prototypes for each test object and classifies the object in terms of the labels of the prototypes and a voting mechanism. Prototypes are vectors that reside in the same vector space as feature vectors and can be derived from training samples in various ways. The simplest way is to use all training samples as prototypes (Fix and Hodges, 1951, 1952, 1991a, 1991b). Besides not incurring any training costs, this approach has two major advantages.

First, for a finite set of training samples S, the error rate using all samples as prototypes does not exceed twice the Bayes risk (Cover and Hart, 1967). Second, it ensures consistency, or asymptotic Bayes-risk efficiency (Stone, 1977; Devroye and Györfi, 1985; Zhao, 1987; Devroye et al., 1994).

However, recruiting all training samples as prototypes can incur a high computational cost during the test procedure, which is prohibitive in applications with large corpora. Consequently, certain editing rules have been proposed to reduce the number of prototypes. The condensed nearest neighbor (CNN) rule (Hart, 1968) was the first, and perhaps simplest, proposal among many subsequent ones, all of which try to extract a subset from a collection of samples. These algorithms execute a process iteratively to check the satisfaction of certain criteria for the current set of prototypes, and add or drop prototypes until a stop condition is met. Wilson and Martinez (2000) collected and compared many algorithms of this type (in particular, DROP1 to DROP5), and categorized them as instance-based learning (IBL) algorithms. More recently, an alternative IBL algorithm called the Iterative Case Filtering (ICF) algorithm (Brighton and Mellish, 2002) was proposed. ICF runs faster than most IBL algorithms, which drop rather than add samples (this point is discussed further in Section 7.2), yet it achieves comparable accuracy to the latter algorithms.

Another method for finding prototypes can be categorized as cluster-based learning (CBL) algorithms, in which prototypes are not samples per se, but can be derived as the weighted averages of samples. The *k*-means clustering algorithm (Lloyd, 1982; Max, 1960; Linde et al. 1980), the fuzzy *c*-means algorithm (Bezdek, 1981; Höppner et al., 1999), and the learning vector quantization algorithm (Kohonen, 1988, 1990) are examples of this method. Instead of representing prototypes as the weighted averages of samples, they can be represented as centroids of clusters (Devi and Murty, 2002), or as hyperrectangles (high-dimensional rectangles) (Salzberg, 1991). In the latter case, the distance between a sample and a hyperrectangle not containing the sample is defined as the Euclidean distance between the sample and the nearest face of the hyperrectangle.

In their guidelines for the design of prototype learning algorithms, Devroye et al. (1996, Chapter 19) propose some sufficient conditions for the consistency of this kind of algorithm. The conditions stipulate that: (a) the algorithm should minimize the empirical error, which is the error in classifying training samples; and (b) the number of prototypes should grow as a lower order of the number of training samples.

Support vector machines (SVM) can also be used for pattern classification. In this approach, objects are classified by maximizing the margins between samples with different labels, where the margin is defined as the gap between two parallel hyperplanes (Figure 1a). The consistency of SVM is assured if the samples are bounded and the margin between samples with different labels holds (Vapnik, 1995; Schölkopf et al. 1999; Cristianini and Shawe-Taylor, 2000).

1.2 Our Contributions

The requirement that data should be bounded is reasonable, since it is a common practice in applications to normalize feature values to a certain bounded interval (between 0 and 1, for example). The margin assumption, on the other hand, is unique to SVM. However, we can prove the consistency of CNN under a more relaxed assumption (Figure 1b). For convenience, we say that two labeled entities (that is, samples or prototypes) are homogeneous if they have the same label; otherwise, they are heterogeneous. We require a non-zero distance between heterogeneous samples.

Despite its consistency, CNN could be improved in two ways. First, its criterion for prototype satisfaction is rather weak and could be strengthened. Second, it is not difficult to develop an alternative process by using a cluster-based rule to construct prototypes. Experiments show that the latter process often achieves better test accuracy than CNN. Another issue with this algorithm is its theoretical standing. The consistency of CNN derives from the fact that its prototypes are samples and thus always keep a certain distance from each other. The cluster centers, on the other hand, are not samples but the weighted averages of samples, so it is difficult to control the distances between them. To resolve this problem, we adopt a hybrid solution that combines cluster centers and certain selected samples to maintain a desirable separation between all the resultant prototypes.

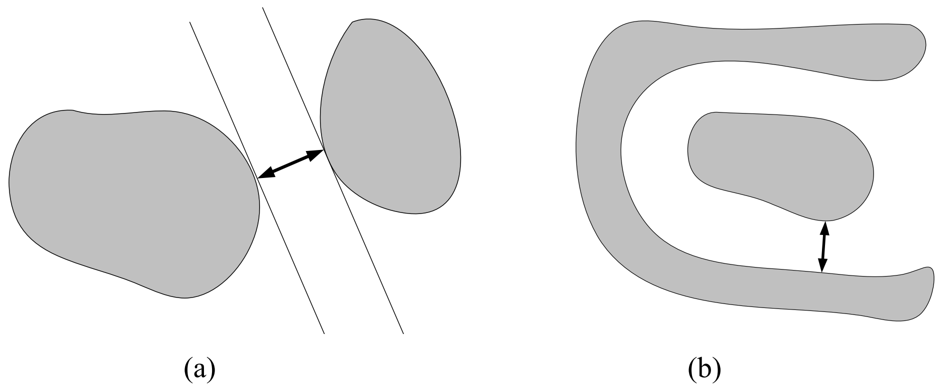

Figure 1. (a) A margin exists between two data sets. (b) A positive distance exists between two data sets.

Note that it is not always appropriate to minimize training errors for SVM. Sometimes, a higher number of training errors should be tolerated so that prediction errors can be reduced. Such flexibility, which is built into the "soft-margin" version of SVM (Cortes and Vapnik, 1995; Bartlett and Shaw-Taylor, 1999), yields better test accuracy than the "hard-margin" version. Fortunately, this flexibility also exists in adaptive prototype learning (APL) algorithms, and can be derived by a tradeoff between the number of prototypes and their predictive power. However, although APL reduces training errors by adding prototypes, it increases the risk of overfitting. A balance between these two factors is made possible by a cross-validation study, similar to that used for SVM. We discuss this point further in Section 6.

In summary, we propose two types of prototype learning algorithm. The first is an instance-based algorithm, which adds samples as prototypes according to an enhanced absorption criterion. The advantage of this approach (discussed in Section 7.2) is that it achieves substantially higher test accuracy at a relatively low training cost, compared to other instance-based algorithms, whose major merit is a lower ratio of prototypes to training samples. Although our algorithm achieves higher test accuracy at the expense of a somewhat higher ratio of prototypes to training samples, we believe this is acceptable, since it enables the proposed classifier to even outperform the *k*-nearest neighbor (*k*-NN) rule in terms of accuracy. The second approach is a hybrid method that constructs prototypes as either samples or the weighted averages of samples. Compared to SVM, the hybrid prototype learning method yields higher test accuracy, at the expense of a higher training cost (discussed in Section 7.3).

The remainder of the paper is organized as follows. In the next section, we present the Vapnik-Chervonenkis (VC) theory of multiclass classification. In Section 3, we provide proof of the consistency of CNN under certain conditions. In Section 4, the extension of CNN to APL is discussed, along with the convergence of APL and its consistency under the same conditions. In Sections 5 and 6, respectively, we describe a kernelized version and a soft version of APL. Section 7 contains experimental studies of APL and comparisons with some instance-based learning algorithms, namely, *k*-NN, CNN and SVM. Finally, in Section 8, we present our conclusions.

2 Vapnik-Chervonenkis Theory of Multiclass Classification

In this section, we develop a basic theory of prototype-learning algorithms. In particular, we derive an asymptotic result for generalization errors of prototype learning algorithms. For the case of binary classification, in which an object is classified as one of two class types, the standard Vapnik-Chervonenkis (VC) theory provides such a bound. This theory, however, is not sufficient for our purpose, since we deal with multiclass classifications in which we want to classify an object into one of *m* classes, with $m \geq 2$. Here, we focus on extending the standard VC theory to such a case.

The standard VC theory is a probabilistic theory that has great breadth and depth. To present a complete version of the theory in a journal paper is impossible. In fact, it is also unnecessary, since a comprehensive treatment can be found in the book *A Probabilistic Theory of Pattern Recognition* (Devroye et al., 1996). For this reason, we follow its notations closely (with some minor changes to suit our purpose) and quote those theorems that are relevant to our task.

We assume there are *n* training samples ($\mathbf{x}_1$, y_1), …, ($\mathbf{x}_n$, y_n), and a test sample ($\mathbf{x}$, y) drawn independently from the set $R^d \times \Lambda$ according to the same distribution, where $\Lambda = \{1, 2, \ldots, m\}$ is a set of labels or class types. Then, for a classifier $g: R^d \rightarrow \Lambda$, we define its training error $\hat{L}_n(g)$ and testing error $L(g)$ as follows.

Definition 1 *The training error of a classifier g is defined as the fraction of training samples misclassified by g, that is,* $\hat{L}_n(g) = (1/n)\sum_{i=1}^{n} I_{\{g(\mathbf{x}_i) \neq y_i\}}$, *where I is the indicator function such that* $I_{\{g(\mathbf{x}_i) \neq y_i\}} = 1$ *if and only if* $g(\mathbf{x}_i) \neq y_i$. *The testing error of a classifier g is defined as the probability that a test sample has been misclassified by g, that is,* $L(g) = \Pr\{g(\mathbf{x}) \neq y\}$.

Typically, from the training samples, a learning algorithm tries to build a classifier *g* of a generic class *C*, with the objective that *g* can generalize well in the sense that it has a small testing error. The standard VC theory provides a bound for the testing error of *binary* classifiers. This bound can be expressed in terms of the following complexity measure of *C*.

Definition 2 *Let C be a collection of binary classifiers of the form* $g : R^d \rightarrow \{0, 1\}$. *For any n, the* n^{th} *shatter coefficient of C is defined as*

$$S(C,n) = \max_{T \subseteq R^d, |T|=n} |\{g_T : g \in C\}|,$$

where g_T *is the function obtained by restricting g to the domain T, and |X| for any set X is the number of elements of X.*

Intuitively, the n^{th} shatter coefficient of *C* is the maximum number of ways that an *n*-element subset can be partitioned by the classifiers in *C*. The following well-known result of Vapnik and Chervonenkis (1971, 1974a, 1974b) provides a bound for the testing error of classifiers in *C*, which we denote as the VC-bound. We adopt this result from Theorem 12.6 in Devroye et al. (1996).

Theorem 3 *Let C be a collection of binary classifiers. Then, for any n and any* $\varepsilon > 0$,

$$\Pr\left\{\sup_{g \in C} |\hat{L}_n(g) - L(g)| > \varepsilon\right\} \leq 8S(C,n)e^{-n\varepsilon^2/32}.$$

Next, we explain how to obtain an analogous result for multiclass classifiers. For a collection C of multiclass classifiers g: $R^d \rightarrow \Lambda$, let C^B be the class of binary classifiers g^i: $R^d \rightarrow \{0, 1\}$ such that $g^i(x) = 1$ if and only if $g(x) = i$, for $g \in C$ and $i = 1, 2, \ldots, m$.

Theorem 4 *Let C be a collection of multiclass classifiers of the form g: $R^d \rightarrow \Lambda$. Then, for any n and any $\varepsilon > 0$,*

$$\Pr\left\{\sup_{g\in C} |\hat{L}_n(g) - L(g)| > \varepsilon\right\} \le 8S(C^B, n)e^{-n\varepsilon^2/(8m^2)}.$$

Proof: First, consider any classifier g: $R^d \rightarrow \Lambda$. A sample $(\mathbf{x}, y) \in R^d \times \Lambda$ is misclassified by g if and only if it is misclassified by both g^y and $g^{g(\mathbf{x})}$. Thus, $2\hat{L}_n(g) = \sum_{i=1}^m \hat{L}_n(g^i)$ and $2L(g) = \sum_{i=1}^m L(g^i)$. Then, by triangle inequality, the condition $|L_n(g) - L(g)| > \varepsilon$ implies that

$$\sum_{i=1}^m |\hat{L}_n(g^i) - L(g^i)| \ge |\sum_{i=1}^m \hat{L}_n(g^i) - \sum_{i=1}^m L(g^i)| = |2\hat{L}_n(g) - 2L(g)| > 2\varepsilon,$$

and $|\hat{L}_n(g^i) - L(g^i)| > 2\varepsilon/m$ for some i. Therefore,

$$\Pr\left\{\sup_{g\in C} |\hat{L}_n(g) - L(g)| > \varepsilon\right\} \le \Pr\left\{\sup_{g^i\in C^B} |\hat{L}_n(g^i) - L(g^i)| > 2\varepsilon/m\right\}.$$

This theorem follows from the previous theorem with ε replaced by $2\varepsilon/m$. □

As stated in the above theorem, the VC-bound is the product of two terms. The first is just the shatter coefficient, whose magnitude depends on the collection of classifiers C. The second term decays exponentially to zero as $n \rightarrow \infty$. To obtain an asymptotic result from this product, we need to know how fast the shatter coefficient grows as $n \rightarrow \infty$. If its growth is slower than the decay of the second term, then the VC-bound approaches zero as $n \rightarrow \infty$.

Let us now define some terms. A prototype data pair $(\mathbf{p}, y)$ consists of a prototype $\mathbf{p} \in R^d$ and its label y. We say that a classifier uses the 1-NN rule based on prototype data pairs if g assigns to each $\mathbf{x} \in R^d$ the label of the nearest prototype to $\mathbf{x}$. The collection of all *multiclass* classifiers using the1-NN rule based on k prototype data pairs is denoted by $C_{(k)}$, while the collection of *binary* classifiers using the 1-NN rule based on k prototype data pairs is denoted by $B_{(k)}$. We want to derive a result for $C_{(k)}$ in terms of a known result of $B_{(k)}$. To do this, we adopt the following lemma from Devroye et al. (1996, p. 305), which provides a bound for $S(B_{(k)}, n)$.

Lemma 5 *$S(B_{(k)}, n) \le (ne/(d+1))^{(d+1)(k-1)}$.*

From Theorem 4 and Lemma 5, we derive the following result for $C_{(k)}$.

Theorem 6 *For any n and any $\varepsilon > 0$,*

$$\Pr\left\{\sup_{g\in C_{(k)}} |\hat{L}_n(g) - L(g)| > \varepsilon\right\} \le 8(ne/(d+1))^{(d+1)(k-1)} e^{-n\varepsilon^2/(8m^2)}.$$

Proof: In order to apply Theorem 4, we need to find a bound for $S(C_{(k)}^B, n)$, where $C_{(k)}^B$ derives from $C_{(k)}$ in the same way as C^B derives from C. Since $C_{(k)}^B \subseteq B_{(k)}$, we have $S(C_{(k)}^B, n) \le S(B_{(k)}, n)$, which follows easily from the definition of the shatter coefficient.

Therefore, by Lemma 5, we have $S(C^B_{(k)}, n) \le (ne/(d+1))^{(d+1)(k-1)}$. Combining this inequality and Theorem 4, we obtain the desired result. □

For a given sequence of training data $D_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, a classification rule is a sequence of classifiers $\{g_n\}$ such that g_n is built on D_n. Such a rule is said to be *consistent*, or asymptotically Bayes-risk efficient, if

$$\lim_{n\to\infty} \Pr\{L(g_n) - \inf_g L(g) > \varepsilon\} = 0,$$

for any $\varepsilon > 0$, where $\inf_g L(g)$ is the infimum (i.e., the greatest lower bound) of the testing errors of all classifiers of the form g: $R^d \to \Lambda$. A prototype classification rule, on the other hand, is a sequence $\{g_n\}$ such that g_n uses the 1-NN rule based on k_n prototypes for some k_n. The following corollary provides a sufficient condition for the consistency of a prototype classification rule. Note that, in stating the corollary, we use o($f(n)$) to denote a quantity whose ratio to $f(n)$ approaches zero as $n \to \infty$.

Corollary 7 *Suppose that $\{g_n\}$ is a prototype classification rule such that $\hat{L}_n(g_n) = 0$ and k_n = o($n\varepsilon^2/(m^2 d\log n)$) for all n. Then, for any $\varepsilon > 0$,*

$$\lim_{n\to\infty} \Pr\{L(g_n) - \inf_g L(g) > \varepsilon\} = \lim_{n\to\infty} \Pr\{L(g_n) > \varepsilon\} = 0.$$

Proof: Since $\hat{L}_n(g_n) = 0$, the condition that $L(g_n) > \varepsilon$ implies that $|\hat{L}_n(g_n) - L(g_n)| > \varepsilon$ and thus $\sup_{g\in C_{(k_n)}} |\hat{L}_n(g) - L(g)| > \varepsilon$ as well. Hence, $\Pr\{L(g_n) > \varepsilon\} \le \Pr\left\{\sup_{g\in C_{(k_n)}} |\hat{L}_n(g) - L(g)| > \varepsilon\right\}$. Also, since k_n = o($n\varepsilon^2/(m^2 d\log n)$), by Theorem 6, we have $\Pr\{L(g_n) > \varepsilon\} \to 0$ as $n \to \infty$. Finally, $L(g_n) - \inf_g L(g) > \varepsilon$ implies that $L(g_n) > \varepsilon$, so the probability of the former inequality also approaches zero as $n \to \infty$. □

3 The Condensed Nearest Neighbor Rule

Following the notations defined in Section 2, we assume that a set of observed data, or samples, $(\mathbf{x}_1, y_1)$, $(\mathbf{x}_2, y_2)$, ..., $(\mathbf{x}_n, y_n)$ is given. Our goal here is to extract a subset U_n from $X_n = \{\mathbf{x}_i\}_{i=1}^n$ in such a way that if $\mathbf{u}$ is the nearest member of U_n to $\mathbf{x}_i$, then $l(\mathbf{u}) = y_i$, where $l(\mathbf{u})$ is the label of $\mathbf{u}$. Members of U_n are called prototypes, and samples whose labels match those of their nearest prototypes are said to be *absorbed*.

The CNN rule (Hart, 1968) is a simple way of solving the above problem. Starting with $U_n = \{\mathbf{x}_0\}$, where $\mathbf{x}_0$ is randomly chosen from X_n, CNN scans all members of X_n. It then adds to U_n a member $\mathbf{x}$ of X_n whose nearest prototype's label does not match that of $\mathbf{x}$. The algorithm scans X_n as many times as necessary, until all members of X_n have been absorbed or, equivalently, no more prototypes can be added to U_n.

Let $\delta_n = \min\{\|\mathbf{x}_i - \mathbf{x}_j\| : \mathbf{x}_i, \mathbf{x}_j \in X_n, \text{and}\, l(\mathbf{x}_j) \ne l(\mathbf{x}_j)\}$, that is, δ_n is the minimal distance between heterogeneous samples. Since $\{\delta_n\}$ is a decreasing sequence, there exists a δ such that $\delta_n \to \delta$ as $n \to \infty$. The consistency of the CNN rule can be proved under the following two conditions. 1) *Boundedness*: all samples are included in a bounded set; that is, there exists a region H of radius R such that $\Pr\{\mathbf{x} \in H\} = 1$. 2) *Non-zero separation*: the limit δ of $\{\delta_n\}$ is non-zero.

Lemma 8 *Under the conditions of boundedness and non-zero separation, the number of CNN prototypes cannot exceed* $(2R/\delta+1)^d$*, where R is the radius of H and* δ *is the limit of* $\{\delta_n\}$*.*

Proof: We want to prove that all prototypes are δ-separated, that is, their distance is at least δ. This is true for any two prototypes with different labels, since all prototypes are samples and heterogeneous samples are δ-separated. Therefore, we only have to prove that all prototypes of the same label are also δ-separated.

We assume that $\mathbf{p}$ and $\mathbf{q}$ are prototypes of the same label. As CNN is a sequential process, its prototypes are constructed in linear order. Without loss of generality, we assume that $\mathbf{p}$ is constructed before $\mathbf{q}$; hence, there must be a prototype $\mathbf{m}$ that is constructed before $\mathbf{q}$, $l(\mathbf{m}) \neq l(\mathbf{q})$, and $\|\mathbf{q}-\mathbf{p}\| \geq \|\mathbf{q}-\mathbf{m}\|$. Now, since $\mathbf{q}$ and $\mathbf{m}$ have different labels, $\|\mathbf{q}-\mathbf{m}\| \geq \delta$. Combining these two facts, we obtain $\|\mathbf{q}-\mathbf{p}\| \geq \delta$.

We define a ball of radius r centered at $\mathbf{w}$ as $B(\mathbf{w}, r) = \{\mathbf{x} : \|\mathbf{x}-\mathbf{w}\| < r\}$. Let the prototypes be $\{\mathbf{p}_i\}_{i=1}^k$. Since they are δ-separated from each other, all the balls $B(\mathbf{p}_i, \delta/2)$ are non-overlapping. Moreover, the union of these balls is contained in a ball of radius $R+\delta/2$ (Figure 2). So, we must have $k(\delta/2)^d < (R+\delta/2)^d$, or $k < (2R/\delta+1)^d$. □

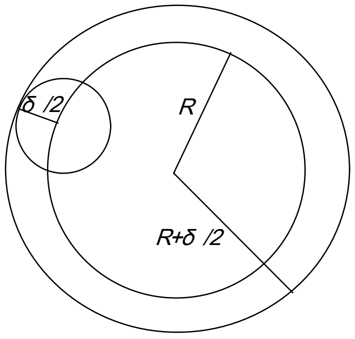

Figure 2. If all samples are contained in a ball of radius *R*, then all balls of radius *δ/2* centered at a sample are included in a ball of radius *R+δ/2*.

From this lemma and the corollary to Theorem 6, we derive the following.

Theorem 9 *Let {g_n} be a sequence of classifiers using the 1-NN rule based on CNN prototype data pairs. The boundedness and non-zero separation conditions ensure the consistency of {g_n}.*

4 Adaptive Prototype Learning Algorithms

An adaptive prototype learning algorithm is similar to CNN in that it adds as many prototypes as necessary until all samples have been absorbed. APL, however, differs from CNN in two respects: the absorption criterion and the nature of prototypes. In CNN, all prototypes are samples, whereas prototypes in APL can be samples or the weighted averages of samples. We denote prototypes that are samples as instance-based prototypes (IBPs) to differentiate them from cluster-based prototypes (CBPs), which are the weighted averages of samples. First, we develop a special type of APL algorithm for IBPs and prove its consistency under the conditions that ensure the consistency of CNN. We then propose a more complex type of APL that combines IBPs and CBPs, after which we address APL's convergence and consistency properties.

4.1 Generalized CNN

The instance-based APL includes CNN as a special case. For this reason, we denote it as a generalized CNN, or GCNN. The difference is that GCNN employs a strong absorption criterion, in contrast to the weak criterion employed by CNN. According to CNN, a sample **x** is absorbed if

$$\|\mathbf{x}-\mathbf{q}\|-\|\mathbf{x}-\mathbf{p}\|>0, \tag{1}$$

where **p** and **q** are prototypes, **p** is the nearest homogeneous prototype to **x**, and **q** is the nearest heterogeneous prototype to **x**. For GCNN, however, we adopt the following criterion:

$$\|\mathbf{x}-\mathbf{q}\|-\|\mathbf{x}-\mathbf{p}\|>\rho\delta_n, \qquad \rho\in[0,1). \tag{2}$$

We say that a sample is *weakly* absorbed if it satisfies (1), and *strongly* absorbed if it satisfies (2). Note that (1) corresponds to the case where ρ = 0 in (2). Adopting (2) makes it possible to improve the classifier by optimizing ρ. The question of how to optimize ρ is addressed in Section 6.

We now describe the steps of GCNN.

G1 *Initiation*: For each label y, select a y-sample as an initial y-prototype.

G2 *Absorption Check*: Check whether each sample is strongly absorbed (absorbed, for short). If all samples are absorbed, terminate the process; otherwise, proceed to the next step.

G3 *Prototype Augmentation*: For each y, if any unabsorbed y-samples exist, select one as a new y-prototype; otherwise, no new prototype is added to label y. Return to G2 to proceed.

In G1, a y-sample is selected as follows. We let each y-sample cast a vote to its nearest y-sample, and select the one that receives the highest number of votes. In G3, an unabsorbed y-sample is selected as follows. Let $\Psi_y = \{\mathbf{x}_i: l(\mathbf{x}_i)=y \ \&\ \mathbf{x}_i \text{ is unabsorbed}\}$. We let each member of Ψ_y cast a vote for the nearest member in this set. The selected y-sample is the member of Ψ_y that receives the highest number of votes.

Lemma 10 *GCNN prototypes satisfy the following properties. (a) For each prototype* **p***, no heterogeneous sample can be found in* $B(\mathbf{p},\delta_n)$. *(b) For any two heterogeneous prototypes* **p** *and* **q***,* $\|\mathbf{p}-\mathbf{q}\|\geq\delta_n$. *(c) For any two homogeneous prototypes* **m** *and* **n***,* $\|\mathbf{m}-\mathbf{n}\|>(1-\rho)\delta_n$.

Proof: Propositions (a) and (b) follow from the fact that GCNN prototypes are samples and the separation between any two heterogeneous samples is at least δ_n. To prove (c), let two homogeneous prototypes **m** and **n** be given, and let **m** be constructed before **n**. Since **n** is not absorbed by the time it is taken as a prototype, there exists a heterogeneous prototype **q** such that $\|\mathbf{n}-\mathbf{q}\|-\|\mathbf{n}-\mathbf{m}\|\leq\rho\delta_n$ or, equivalently,

$$\|\mathbf{n}-\mathbf{m}\|\geq\|\mathbf{n}-\mathbf{q}\|-\rho\delta_n. \tag{3}$$

Since **n** and **q** are heterogeneous, by (a), **n** cannot lie in $B(\mathbf{q},\delta_n)$. Thus,

$$\|\mathbf{n}-\mathbf{q}\|\geq\delta_n. \tag{4}$$

Combining (3) and (4), we obtain $\|\mathbf{n}-\mathbf{m}\|\geq\delta_n-\rho\delta_n=(1-\rho)\delta_n$. □

We define the number of iterations as the number of times G2 has been executed. The following lemma states that the number of iterations cannot exceed a certain magnitude. Let R_n be the radius of the smallest ball containing all samples in X_n.

Lemma 11 *The number of GCNN prototypes cannot exceed* $[(2R_n+\delta_n)/(1-\rho)\delta_n]^d$. *Moreover, GCNN converges within a finite number of iterations.*

Proof: Lemma 10 ensures that homogeneous and heterogeneous GCNN prototypes are separated by certain constants. Using this fact and a similar argument to that in the proof of Lemma 8, we conclude that the number of GCNN prototypes cannot exceed $[(2R+\delta_n)/(1-\rho)\delta_n]^d$. Since at least one prototype is created at each iteration, the number of iterations cannot exceed this number either. □

Now, under the conditions of boundedness and non-zero separation, we can also show that the number of GCNN prototypes is bounded from above by $[(2R+\delta_n)/(1-\rho)\delta_n]^d$, with R replacing R_n. Since $[(2R+\delta_n)/(1-\rho)\delta_n]^d \le [(2R+\delta)/(1-\rho)\delta]^d$, the number of GCNN prototypes is bounded from above by a constant independent of *n.* The consistency of GCNN follows from the same argument that demonstrates the consistency of CNN.

Theorem 12 *Under the conditions of boundedness and non-zero separation, GCNN is consistent.*

4.2 Linear Adaptive Prototype Learning

Having explained GCNN, we are ready to describe a more complex type of APL that can take a mixture of IBPs and CBPs as its prototypes. To differentiate it from GCNN, and from another version of APL to be described later, we denote this algorithm as linear APL (LAPL).

Recall that the consistency of GCNN derives from the separation of prototypes. We wish to obtain a similar separation between LAPL prototypes, but the addition of CBPs raises some problems.

The first problem is the separation required for heterogeneous prototypes. While a δ_n separation can be easily maintained by any two heterogeneous IBPs, it may not be maintained so easily by two heterogeneous CBPs. Therefore, we require the separation to be $f\delta_n$, where $f \in [0,1]$. How we determine the optimal value of *f* is discussed in Section 6.

The next problem is the absorption criterion. For LAPL, we adopt the following:

$$\|\mathbf{x}-\mathbf{q}\| - \|\mathbf{x}-\mathbf{p}\| > \rho f\delta_n, \quad \text{for } \rho \in [0,1). \tag{5}$$

How to optimize *ρ* is also addressed in Section 6.

The third problem is how to maintain a positive separation between *all* LAPL prototypes. To achieve this objective, we specify the following requirements.

(C1) For each prototype **p**, no heterogeneous sample exists in $B(\mathbf{p}, f\delta_n)$.

(C2) For any two heterogeneous prototypes **p** and **q**, $\|\mathbf{p}-\mathbf{q}\| \ge f\delta_n$.

(C3) For any two homogeneous prototypes **m** and **n**, $\|\mathbf{m}-\mathbf{n}\| > (1-\rho)f\delta_n$.

Thus, in the transition from GCNN to LAPL, we have systematically changed δ_n to $f\delta_n$. We now state the LAPL algorithm, and prove that the prototypes derived from it satisfy (C1), (C2), and (C3). We first describe the general scheme of the algorithm, and then provide the technical details. The steps of LAPL are:

H1 *Initiation*: For each label *y*, initiate a *y*-prototype as the average of all *y*-samples. If this prototype does not satisfy (C1), (C2), and (C3), we apply the prototype adjustment module (described later in this section).

H2 *Absorption Check*: Check whether each sample has been absorbed. If all samples have been absorbed, terminate the process; otherwise, proceed to the next step.

H3 *Prototype Refreshment*: For each un-satiated label *y* (i.e., some *y*-samples are unabsorbed), select an unabsorbed *y*-sample. We then apply a clustering algorithm to construct clusters, using the selected *y*-sample and all existing *y*-prototypes as seeds. The centers of the resultant clusters are new *y*-prototypes. If these prototypes do not satisfy (C1), (C2), and (C3), we apply the prototype adjustment module (described later in this section). Return to H2 to proceed.

We now provide the technical details.

Selection of Unabsorbed Samples in H1 and H3. The selection procedures in H1 and H3 are the same as those in G1 and G3.

Clustering Algorithms in H3. Any clustering algorithm can be used. For the experiment described in this paper, we use the *k*-means (KM) and the fuzzy *c*-means (FCM) clustering algorithms, both of which are applied to training samples of the same label. Thus, if there are *m* labels in the training data, we apply the algorithms *m* times. Details of the methods are as follows.

The KM method (Lloyd, 1982; Max, 1960; Linde et al. 1980) derives a locally optimal solution to the problem of finding a set of cluster centers $\{\mathbf{c}_i\}_{i=1}^{p}$ that minimizes the objective function

$$\sum_{j=1}^{n} \min_{i=1,\ldots,p} \| \mathbf{c}_i - \mathbf{x}_j \|^2 . \tag{6}$$

KM's iterative process is performed as follows. Setting seeds as the initial cluster centers, we add each sample to the cluster whose center is nearest to it. We then reset the center of each cluster as the average of all the samples that fall in that cluster. To ensure rapid convergence of KM, we require that the process stops when the number of iterations reaches 30, or the membership of the clusters remains unchanged after the previous iteration.

In FCM (Bezdek, 1981; Höppner et al., 1999), the objective function to be minimized is

$$\sum_{j=1}^{n} \sum_{i=1}^{p} u_{ij}^{m} \| \mathbf{c}_i - \mathbf{x}_j \|^2 , \quad \text{for } m \in (1, \infty) \tag{7}$$

under the constraint

$$\sum_{i=1}^{p} u_{ij} = 1 , \qquad \text{for } j = 1, 2, ..., n, \tag{8}$$

where u_{ij} is the membership grade of sample $\mathbf{x}_j$ to prototype $\mathbf{c}_i$. Using the Lagrangian method, we can derive the following equations:

$$u_{ij} = \frac{\left(1 / \| \mathbf{c}_i - \mathbf{x}_j \|\right)^{\frac{2}{m-1}}}{\sum_{k=1}^{p} \left(1 / \| \mathbf{c}_k - \mathbf{x}_j \|\right)^{\frac{2}{m-1}}} , \tag{9}$$

$$\mathbf{c}_i = \frac{\sum_{j=1}^{n} u_{ij}^{m} \mathbf{x}_j}{\sum_{j=1}^{n} u_{ij}^{m}} , \tag{10}$$

for *i* = 1, 2, …, *p*, and *j* = 1, 2, …, *n* respectively. FCM is a numerical method that finds a locally optimal solution for (9) and (10). Using a set of seeds as the initial solution for $\{\mathbf{c}_i\}_{i=1}^{p}$, the algorithm computes $\{u_{ij}\}_{i,j=1}^{p,n}$ and $\{\mathbf{c}_i\}_{i=1}^{p}$ iteratively. To ensure rapid convergence of FCM, we require that the process stops when the number of iterations reaches 30, or $\sum_{i=1}^{p} \| \mathbf{c}_i^{old} - \mathbf{c}_i^{new} \| = 0$.

The prototype adjustment module is used to adjust the location of prototypes if they do not satisfy the separation conditions (C1), (C2), and (C3).

Prototype Adjustment in H1. The purpose of this module is to replace prototypes that violate (C1) or (C2) with those that do not. Note that there is only one prototype per label in H1, so we do not need to worry about (C3). There are two steps in this stage.

Step 1: If we find a CBP **p** that violates (C1), which requires that no heterogeneous sample exists in $B(\mathbf{p}, f\delta_n)$, we replace **p** with a sample of the same label. The replacement sample is an IBP and is selected in exactly the same way as a seed is selected in G1.

Step 2: If we find a CBP **p** that violates (C2), which requires that $\|\mathbf{p}-\mathbf{q}\| \geq f\delta_n$ for any other prototype **q**, we replace **p** with an IBP of the same label. We perform this operation iteratively, until the desired separations hold between CBPs, and between CBPs and IBPs.

We now prove that after these two steps, all prototypes satisfy (C1) and (C2). We first prove that, after Step 1, all prototypes satisfy (C1). By assumption, all CBPs satisfy (C1) after this step. Also, all IBPs satisfy (C1), since all heterogeneous samples are δ_n-separated from them and $\delta_n \geq f\delta_n$. We now prove that, after Step 2, all prototypes satisfy both (C1) and (C2). It is clear that all prototypes satisfy (C1) at the end of this step; and each CBP maintains $f\delta_n$-separations from other prototypes by assumption. Also, since each IBP maintains δ_n-separations from other IBPs and $\delta_n \geq f\delta_n$, each IBP maintains $f\delta_n$-separations from other IBPs.

Prototype Adjustment in H3. This module adjusts prototypes in two steps.

Step I: A set of prototypes of the same label is called a *pack*. When a pack consists of CBPs that satisfy (C1) and they are $(1-\rho)f\delta_n$-separated from each other, as required by (C3), we preserve that pack. Otherwise, we replace it with the set of seeds from which the CBPs were derived.

Step II: Two packs are said to be $f\delta_n$-separated if any two prototypes drawn from them are $f\delta_n$-separated. When we find two packs that are not $f\delta_n$-separated, we replace one of them with the set of seeds from which its prototypes were derived. We perform this operation iteratively until the remaining packs are $f\delta_n$-separated.

We now show that, after Step I, all prototypes satisfy (C1) and (C3). For convenience, we call preserved packs *P-packs* and replacement packs *R-packs*. An R-pack consists of existing prototypes, called X-prototypes, and an unabsorbed sample, called a U-prototype. By induction, all X-prototypes meet (C1) and (C3). The U-prototype, denoted as **u**, also satisfies (C1), because heterogeneous samples are δ_n-separated from each other. It remains to show that **u** is $(1-\rho)f\delta_n$-separated from all X-prototypes. This fact follows from a similar argument to that for Lemma 10(c), so we omit the proof. We conclude that all the prototypes satisfy (C1) and (C3).

We now prove that, after Step II, all prototypes satisfy (C1), (C2), and (C3). It is clear that all prototypes satisfy (C1) and (C3) at the end of this step. It remains to prove that they also satisfy (C2), that is, heterogeneous prototypes are $f\delta_n$-separated. We want to show that all heterogeneous prototypes in the R-packs are $f\delta_n$-separated. As noted earlier, these prototypes consist of X-prototypes and U-prototypes. By induction, heterogeneous X-prototypes are $f\delta_n$-separated. Heterogeneous U-prototypes are also $f\delta_n$-separated, as noted before. U-prototypes are also $f\delta_n$-separated from heterogeneous X-prototypes, since all X-prototypes satisfy (C1). Thus, at the end of Step II, all prototypes satisfy (C2).

The $f\delta_n$-separation between heterogeneous prototypes and the $(1-\rho)f\delta_n$-separation between homogeneous prototypes imply the convergence of LAPL and also its consistency under the conditions of boundedness and non-zero separation. Note that the above conclusions do not hold for the case where $f = 0$, which we deal with in Section 5.

Theorem 13 *The LAPL terminates within a finite number of iterations, provided that $f \in (0,1]$, $\rho \in [0,1)$, and $m \in (1,\infty)$.*

Theorem 14. *Let $\{g_n\}$ be a sequence of classifiers using the 1-NN rule based on LAPL prototype data pairs. The conditions of boundedness and non-zero separation ensure the consistency of $\{g_n\}$, provided that $f \in (0,1]$, $\rho \in [0,1)$, and $m \in (1,\infty)$.*

5 Kernelized Adaptive Prototype Learning Algorithms

Let $\Phi : R^d \to H$ be a function that maps from the d-dimensional Euclidean space to a Hilbert space, whose dimension dim(H) may be infinite. In a kernelized adaptive prototype learning algorithm, the goal is to build prototypes in H. To this end, we first transform the given observed data $\{\mathbf{x}_j\}_{j=1}^n$ into $\{\Phi(\mathbf{x}_j)\}_{j=1}^n$. When either KM or FCM is used to compute prototypes, each prototype in H is of the form

$$\mathbf{c}_i = \sum_{j=1}^n u_{ij}^m \Phi(\mathbf{x}_j) / \sum_{j=1}^n u_{ij}^m ,$$

where $\mathbf{c}_i$ and u_{ij} were introduced in (6), (7), and (8). When KM is used, $m \equiv 1$. Moreover, $u_{ij} = 1/n_i$ provided that the j^{th} sample falls in the i^{th} cluster, whose population size is n_i; otherwise, $u_{ij} = 0$. When FCM is used, we compute u_{ij} according to (9) in which the distance now becomes a kernel-based distance, to be defined below.

If $\dim(H) = \infty$, $\mathbf{c}_i$ cannot be expressed in vector form. Even when dim(H) < ∞, it can be computationally expensive to find an explicit form of $\mathbf{c}_i$. Fortunately, we can compute the distance between $\Phi(\mathbf{x}_j)$ and $\mathbf{c}_i$ directly, provided there exists a kernel function (Mercer, 1909; Girosi, 1998)

$$K(\mathbf{x},\mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle, \quad \text{for } \mathbf{x},\mathbf{y} \in R^d .$$

When such a function exists, we obtain the kernel-based distance as

$$\| \mathbf{c}_i - \Phi(\mathbf{x}_j) \|_{Ker}^2 = \langle \mathbf{c}_i - \Phi(\mathbf{x}_j), \mathbf{c}_i - \Phi(\mathbf{x}_j) \rangle$$
$$= \langle \mathbf{c}_i, \mathbf{c}_i \rangle - 2\langle \mathbf{c}_i, \Phi(\mathbf{x}_j) \rangle + \langle \Phi(\mathbf{x}_j), \Phi(\mathbf{x}_j) \rangle . \tag{11}$$

Moreover,

$$\langle \mathbf{c}_i, \mathbf{c}_i \rangle = \frac{\sum_{k=1}^n \sum_{l=1}^n u_{ik}^m u_{il}^m K(\mathbf{x}_k, \mathbf{x}_l)}{\left(\sum_{k=1}^n u_{ik}^m \right)^2}, \tag{12}$$

$$\langle \mathbf{c}_i, \Phi(\mathbf{x}_j) \rangle = \frac{\sum_{k=1}^n u_{ik}^m K(\mathbf{x}_k, \mathbf{x}_j)}{\sum_{k=1}^n u_{ik}^m}, \tag{13}$$

and

$$\left\langle \Phi(\mathbf{x}_j), \Phi(\mathbf{x}_j) \right\rangle = K(\mathbf{x}_j, \mathbf{x}_j). \tag{14}$$

From $\{\| \mathbf{c}_i - \Phi(\mathbf{x}_j) \|_{Ker}^2\}_{i,j=1}^{p,n}$, we derive $\{u_{ij}\}_{i,j=1}^{p,n}$ according to the appropriate formula in the clustering algorithm being used. Since we do not want to express the prototypes explicitly, we use $\{u_{ij}\}_{i,j=1}^{p,n}$ to represent them instead. From the prototypes, we can compute $\{\| \mathbf{c}_i - \Phi(\mathbf{x}_j) \|_{Ker}^2\}_{i,j=1}^{p,n}$ again, using (11)-(14). This iterative process stops if the number of iterations reaches 30, or $\max_{i,j} | u_{ij}^{old} - u_{ij}^{new} | < 0.001$. These steps represent the kernelized versions of KM and FCM, depending on which definition of $\{u_{ij}\}_{i,j=1}^{p,n}$ is used. The kernelized version of FCM was proposed and studied by Wu, Xie and Yu (2003) and Kim et al. (2005).

There is also a kernelized version of GCNN, but we do not consider it in this paper, for the reason to be given in Section 7. We denote the kernelized version of LAPL simply as KAPL, which is derived from LAPL by replacing KM or FCM with an appropriate kernelized version. In addition, we make the following changes. First, the initial y-prototype in KAPL should be the average of all $\{\Phi(\mathbf{x}) : l(\mathbf{x}) = y\}$. Using (11)-(14), we compute the distance of each $\Phi(\mathbf{x})$ to this prototype. Second, we apply the prototype adjustment module in KAPL to separate prototypes. Prototype separation, however, does not imply the convergence of KAPL, since it may have to deal with data in a space of infinite dimensions.

To ensure the convergence of KAPL, we modify the prototype adjustment module as follows. As in LAPL, we adopt the necessary operations to create the desired prototype separation. However, prior to these operations we check if each prototype in a pack has a non-empty domain of attraction (DOA), where the DOA of a y-prototype $\mathbf{p}$ is the set of all y-samples $\Phi(\mathbf{x})$ for which $\mathbf{p}$ is the nearest prototype. Recall that we employ a clustering algorithm to create a pack of prototypes, using an unabsorbed sample $\Phi(\mathbf{u})$ and some other prototypes as seeds. If any prototype in a pack has an empty DOA, we replace that pack with the pack of prototypes constructed earlier. In this case, $\Phi(\mathbf{u})$ is called a *futile* sample. If a sample is declared *futile* in an iteration, it will not be taken as a sample in any later iteration.

Theorem 12 *The KAPL algorithm converges within a finite number of iterations.*

Proof: The number of futile samples is bounded from above, since it cannot exceed n, that is, the number of samples. We assume that the last futile sample is created at iteration i, with $i \le n$. If all samples are absorbed at the end of i, the proof is complete; otherwise, more prototypes will be created, all with non-empty DOAs. The number of unabsorbed samples must decrease to zero, or else the number of DOAs would eventually exceed the number of samples, which would be an absurd result. □

Note that if we treat futile samples in LAPL in the same way, we can prove the convergence of LAPL in the setting where there is no guarantee of prototype separation.

Theorem 13 *Adopting the prototype adjustment module used in KAPL, LAPL converges for $f = 0$ and $m \in (1, \infty)$.*

6 Soft Adaptive Prototype Learning Algorithms

The versions of APL proposed thus far are designed to continue constructing prototypes until all training samples are absorbed or, equivalently, the training error declines to zero. These could be called hard versions of APL. Insistence on a zero training error, however, runs the risk of overfitting. Another approach, called the *soft alternative*, maintains the error rate at a level that enhances the generalization power of the resultant classifier. The optimal error rate can be determined in a cross validation task, which is also needed to find the optimal values of the parameters. All versions of APL involve some parameters; for example, they all involve f and ρ, which regulate prototype separation (cf. (2), (5), (C1), (C2), and (C3)). Moreover, if FCM is used to compute cluster centers, there is another parameter m (cf. (7), (9), and (10)) to consider. In addition, some parameters in KAPL are used to define the kernel-based distance. For example, when the RBF kernel

$$K(\mathbf{x},\mathbf{y}) = \exp(-\gamma \|\mathbf{x}-\mathbf{y}\|^2) \tag{15}$$

is used to define the distance, there is an additional parameter γ, whose range is assumed to be $(0,\infty)$.

To search for the optimal values of the parameters, we perform cross-validation. As all the parameters are assumed to be independent, we must evaluate all combinations of them and determine which one is the most suitable for the task. When a combination of parameter values Q is given, we build prototypes on K-1 folds of data, which serves as the training data, and measure the test accuracy on the remaining fold of data, which serves as the validation data. We determine the optimal training error rate associated with Q as follows.

Given a set of training data and a set of validation data and assuming that the latter is the k^{th} fold of the data, k = 1, 2, …, K, we construct prototypes and record the following information. First, for a given level of e, we record the lowest number of iterations $n_k(e,Q)$ at which the training error rate falls below e. We also compute the validation accuracy rate $v_k(e,Q)$ for all the prototypes obtained at the end of iteration $n_k(e,Q)$. Let $v(e,Q) = \sum_{k=1}^{K} v_k(e,Q)/K$. The optimal training error rate is then

$$e_{opt}(Q) = \arg\max_{e} v(e,Q).$$

Note that once we have constructed prototypes to achieve a training error e_1, we do not need to start from the scratch to obtain a lower training error e_2. Instead, we continue to construct more prototypes until e_2 is reached. At the end of this process, we obtain $v(e,Q)$ for all e and thus $v(e_{opt}(Q),Q)$. When we have done this for all Q, we obtain the optimal Q as

$$Q_{opt} = \arg\max_{Q} v(e_{opt}(Q),Q) .$$

One additional parameter that needs be optimized is the number of k nearest prototypes, which we use in a voting mechanism to determine the label of a test sample. If a tie occurs, we classify the sample according to the nearest prototype. The optimal value of k should be evaluated in the cross-validation applied to the other parameters.

7 Experimental Results

To evaluate the APL algorithms and compare their performance with that of alternative methods, we use 12 benchmark data sets retrieved from the UCI databases (Newman et al., 1998). The results are described in three subsections. The first describes the four types of APL. In the second subsection, we compare the performance of GCNN with six instance-based prototype algorithms proposed in the literature. Then, in the third subsection, we compare the performance of the four APLs with SVM and k-NN. Note that many of the methods, including ours, require that the data must be bounded. One way to meet this requirement is to normalize all the feature values to [0,255], which can be done by the following linear transformation:

$$x \mapsto \frac{(x-v)\times 255}{V-v} \ ,$$

where x is a given feature value, V is the maximum value of the feature, and v is the minimum value. All experimental results reported in this section were obtained using an Intel Pentium 4 CPU 3.4GHz with a 1GB RAM.

7.1 Evaluation of APLs

The four types of APL are listed in Table 1. The first one is GCNN. The other three types of APL are: fuzzy linear APL (*f*-LAPL), crisp kernelized APL (*c*-KAPL), and fuzzy kernelized APL (*f*-KAPL). We use "*f*-" to indicate that the clustering algorithm employed is FCM, and "*c*-" to indicate that the technique is KM. In the experiments, the soft versions of the four APLs are used. Although we can consider the kernelized version of GCNN using RBF as the kernel function, this version of GCNN gains only slightly higher testing accuracy, at the expense of a much higher number of prototypes, than GCNN. So we choose not to discuss it. We do not discuss *c*-LAPL either, since it usually has a lower performance than *f*-LAPL.

In Table 2, we show the parameters used in the four types of APL and also the values of the parameters whose combinations are considered in our experiments. The values result from a trade-off between the demand for accuracy and the need to reduce the computation time. When a combination, *Q,* of parameter values is given, we have to record $v(e,Q)$ for certain values of *e*. In our experiments, the values of *e,* at which we record $v(e,Q)$, are percentages that start from 0% and increase by some increments until they reach 30%. All the percentages are listed in Table 3. The 12 benchmark data sets retrieved from the UCI databases are listed in Table 4, which also shows the number of labels, the number of samples, the number of features per sample, and the number of folds into which we divide the samples during cross validation.

	Assumed Distance
GCNN	Euclidean
f-LAPL	Euclidean
c-KAPL	RBF
f-KAPL	RBF

Table 1. The four types of APL studied in our experiments.

	Values	GCNN	*f*-LAPL	*c*-KAPL	*f*-KAPL
f	0., .1, .25, .5, .75, 1.		√	√	√
ρ	0., .1, .25, .5, .75, .99	√	√	√	√
m	1.05, 1.1, 1.2, 1.3, 1.4		√		√
γ	$a\times10^{-b}$; $a = 1, 2, ..., 9$; $b = 4, 5, ..., 7$			√	√

Table 2. Parameters: their value range, and the types of APL that involve them; "√" indicates that the parameter is used in that type of APL. The parameters *f* and ρ appear in (2), (5), (C1), (C2), and (C3); *m* appears in (7); and γ appears in (15).

Values of *e*
0%, 1%, 2%, 3%, 4%, 5%, 7.5%, 10%, 20%

Table 3. The values of *e* at which we record $v(e,Q)$.

	Number of Labels	Number of Samples	Number of Features	Number of Folds
Iris	3	150	4	5
Wine	3	178	13	5
Glass	6	214	9	5
Ionosphere	2	351	34	10
Cancer	2	683	9	10
Zoo	7	101	16	5
Heart	2	270	13	5
TAE	3	151	5	5
BUPA Liver Disorders (BLD)	2	345	6	5
New Thyroid	3	215	5	5
SPECTF	2	267	44	5
Ecoli	8	336	7	5

Table 4. Information contained in the 12 data sets.

In Table 5, we show three performance measures of the four APLs, namely, the accuracy rate, the training time, and the condensation ratio. Given that *K*-fold cross-validation is conducted, the accuracy rate (AR) is the average accuracy over all validation data sets, each of which is one of the *K* folds; the training time (TT) is the sum of the training times of all training data sets, each of which consists of *K*-1 folds; and the condensation ratio (CR) is the average prototype-to-sample ratios obtained from all training data. Note that for most types of APL, we drop the decimal parts of their training times, since they are relatively insignificant to the integer parts. At the bottom of Table 5, we also show the average of the three measures over the 12 data sets. The boldface figures indicate that the performance of the corresponding method is the best of all the methods applied to the given data set.

The averaged figures in Table 5 show that, in terms of training time, the four APLs are ranked in the following order: GCNN, *f*-LAPL, *c*-KAPL, and *f*-KAPL. The number of all possible combinations of parameter values is the major factor that affects the amount of training time. If we divide the total training time by the above number, then the temporal differences among the four algorithms are reduced drastically, as shown in Table 6. Since APL training under different combinations of parameter values is conducted independently, some fashion of parallel computing, such as cluster computing or grid computing, would help reduce the training time.

GCNN requires the least amount of training time because it picks samples as prototypes, thereby avoiding the rather costly computation of clustering. The *c*-KAPL and *f*-KAPL algorithms, on the other hand, employ kernelized versions of KM and FCM respectively, which are relatively slow. In terms of accuracy, the order of the four APLs is exactly the opposite of that for the training time.

DATA SET		GCNN	*f*-LAPL	*c*-KAPL	*f*-KAPL
Iris	AR	96.62	97.95	98.63	98.40
	TT	0.8	30	16,225	81,334
	CR	9.6	10.33	54.00	5.83
Wine	AR	98.06	99.02	99.02	99.56
	TT	1	144	24,378	175,647
	CR	21.7	18.40	20.22	92.98
Glass	AR	69.39	71.26	72.23	72.73
	TT	1.37	314	18,906	108,891
	CR	48.5	35.98	44.98	22.90
Ionosphere	AR	89.07	91.46	95.88	95.87
	TT	9.45	8,010	399,693	3,078,420
	CR	17.5	5.63	4.56	6.05
Cancer	AR	97.5	97.79	97.50	97.79
	TT	3.34	2,301	496,817	5,265,013
	CR	17.9	4.44	12.74	19.70
Zoo	AR	97.66	97.66	97.66	97.66
	TT	0.83	11	21,666	135,346
	CR	23.2	18.32	22.77	24.50
Heart Rate	AR	85.57	86.90	85.83	86.43
	TT	1.56	1,134	72,925	607,436
	CR	42.6	21.67	35.83	23.98
TAE	AR	63.21	62.47	65.22	65.61
	TT	0.95	229	18,682	133,157
	CR	43.2	51.82	45.86	46.85
BLD	AR	65.93	67.34	67.72	70.52
	TT	2.4	3,379	232,211	1,378,124
	CR	47.9	35.87	74.13	23.33
New Thyroid	AR	97.31	97.76	98.57	99.05
	TT	0.92	135	19,289	134,671
	CR	8.7	12.79	3.72	10.00
SPECTF	AR	83.55	85.63	86.13	87.04
	TT	9.1	8,820	167,428	1,363,339
	CR	28.3	30.15	50.37	28.37
Ecoli	AR	86.44	86.81	86.18	87.06
	TT	1.32	920	35,151	216,235
	CR	24.6	31.85	48.51	27.98
AVERAGE	AR	85.86	86.84	87.55	88.14
	TT	2.3	2,119	126,948	1,056,468
	CR	27.5	23.10	34.81	27.71

Table 5. The performance of the four APLs, where AR = Accuracy Rate (%), TT = Training Time (sec), and CR = Condensation Ratio (%).

	Number of Combinations	Total Training Time	Training Time per Combination
GCNN	6	2.3	0.38
f-LAPL	155	2,119	13.67
c-KAPL	1,116	126,948	113.75
f-KAPL	5,580	1,056,468	189.33

Table 6. The number of all possible combinations of parameter values, the total training time, and the training time per combination for the four types of APL.

	Time to Compute δ_n (sec)	GCNN Run Time (sec)	Ratio (%)
Average	0.012	0.38	3.2

Table 7. The average amount of time to compute δ_n, the average run time of GCNN, and their ratio.

These findings suggest that the high accuracy rates of APLs are derived at the expense of a rather high computational cost. Hence, there is a tradeoff between accuracy rates and training costs, which allows users to choose the most suitable APL based on the size of their problems, their computing resources, and the degree of accuracy they require. There are two reasons for this tradeoff. First, the cluster-based approach has higher generalization power than the instance-based approach, since it picks the weighted averages of samples as prototypes and they are relatively immune to noise. Second, the RBF-based approach has higher generalization power than the Euclidean-based approach. To understand why this is so, we note that for very small γ, the RBF distance between $\mathbf{x}$ and $\mathbf{y}$ is approximately $2\gamma \| \mathbf{x}-\mathbf{y} \|^2$. This means that the RBF distance covers the Euclidean distance as a special case, and using the RBF distance may allow us to find a better-performing classifier than the one we obtain by using the Euclidean distance.

Recall that when applying any APL algorithm we must first compute δ_n, the minimum distance between heterogeneous samples. One may be curious about the ratio of the computing time for δ_n to the run time of APL. In fact, the ratio is 3.2% for GCNN (Table 7) and much less for the other types of APL.

The reason for such a small ratio is as follows. If the number of training samples is n, then the time complexity of computing δ_n is in the order of n^2, while the time complexity of conducting APL training is in the order of n^3. To confirm the latter fact, we note that APL training takes no more than n iterations. Within each iteration, checking the absorption criterion takes no more than n^2 steps, and clustering takes no more than $30\times n^2$ steps (if cluster-based prototypes are required), where 30 is the maximum number of iterations allowed in a clustering algorithm. Furthermore, the space complexity of APL training is in the order of n^2 at most.

LAPL and KAPL are associated with parameters f and ρ, which appear in the absorption criterion (5) and requirements (C1), (C2), and (C3) (cf. Section 4). We were curious to know how the parameters' values affect the prototypes built in the training process, so we studied the training of *f*-LAPL on the 12 data sets. We assume that all parameters, except f, are fixed at certain values. The absorption criterion requires that a training sample should be closer to its nearest homogeneous prototype than to its heterogeneous prototype by at least $f\rho\delta_n$. If we raise the value of f, we increase the likelihood of a sample becoming unabsorbed so that more prototypes would have to be built. This fact is reflected in Table 8, which shows that the average condensation ratio increases as the value of *f* increases. What happens when we fix the values of all parameters except ρ? By raising the value of ρ, we also make the absorption criterion more difficult to satisfy and therefore increase the number of prototypes that need to be built. This fact is reflected in Table 9.

f	0.00	0.10	0.25	0.50	0.75	1.00
Average Condensation Ratio (%)	22.18	23.26	24.56	27.83	30.70	34.92

Table 8. Average condensation ratio of *f*-LAPL over the 12 data sets for various values of *f* when $m = 1.1$, $\rho = 0.5$, and $e = 0$.

ρ	0.00	0.10	0.25	0.50	0.75	0.99
Averaged Condensation Ratio (%)	22.25	23.22	24.53	27.83	30.81	34.31

Table 9. Average condensation ratio of *f*-LAPL over the 12 data sets for various values of ρ when $m = 1.1$, $f = 0.5$, and $e = 0$.

7.2 Comparison of GCNN with Some Instance-Based Learning Algorithms

As noted earlier, GCNN differs from LAPL and KAPL in that it adopts samples as prototypes. It is thus one of the methods, called instance-based learning algorithms, which reduce an entire set of training samples to a subset, while maintaining as much generalization power as possible. For this reason, we compare GCNN with some of the methods that have been proposed in the literature.

Two approaches can be adopted in IBL algorithms. The first is incremental, so it starts with a null set and gradually adds samples as prototypes. Both CNN and GCNN are incremental algorithms. For comparison purposes, we also include a primitive version of GCNN, called pGCNN. It is similar to GCNN, except that the value of parameter *f* is fixed at 0. Note that pGCNN is not the same as CNN. In pGCNN, we select unabsorbed samples through a voting procedure (cf. Section 4) and the training error rate *e* is determined by cross-validation (cf. Section 6). In CNN, however, unabsorbed samples are selected randomly and *e* is fixed at 0.

The second approach is decremental, so it starts with the entire set of samples and gradually removes samples that are considered properly "protected" by the retained ones. For algorithms of this type, we include DROP1 to DROP5 (Wilson and Martinez, 2005) and ICF (Brighton and Mellish, 2002) for comparison. They differ from each other in the way samples are ordered for removal, and in the criterion for removing samples. For further details, readers should refer to the cited references. We used the code provided by Wilson and Martinez (2005) for DROP1 to DROP5, and implemented our own codes for ICF.

For all the methods, we apply cross-validation, similar to that used for the APLs, whereby the 12 data sets are divided into the same number of folds (cf. Table 4). Moreover, in measuring the test accuracy, we use the top-*k* nearest prototypes with *k* being determined in the cross-validation (cf. Section 6). Table 10 shows the performance of all the instance-based methods, with the averaged results shown at the bottom of the table. From the latter results, we observe that GCNN achieves the best accuracy among all the compared methods. In general, the incremental methods have lower training costs than the decremental methods. The only exception is GCNN, which is little slower than ICF. On the other hand, the incremental methods build more prototypes than the decremental methods. Among the incremental methods, GCNN achieves a higher accuracy rate than the other two methods, at the expense of building more prototypes and a higher training cost. Meanwhile, pGCNN constructs fewer prototypes and has a lower training cost than GCNN,

DATA SET		Incremental Methods			Decremental Methods					
		CNN	pGCNN	GCNN	DROP1	DROP2	DROP3	DROP4	DROP5	ICF
Iris	AR	93.07	96.62	96.62	92.65	95.71	95.23	95.23	95.53	95.04
	TT	0.08	0.11	0.8	0.11	0.19	0.16	0.16	0.14	0.11
	CR	13.67	10.1	9.6	6.6	9.2	10.5	10.5	8.9	22
Wine	AR	96.02	96.31	98.06	92.97	93.15	93.42	93.42	98.06	92.81
	TT	0.09	0.12	1	0.41	0.41	0.36	0.36	0.56	0.13
	CR	15.31	10.3	21.7	7.5	12.6	12.1	12.1	8.1	11.1
Glass	AR	65.20	67.54	69.39	59.02	65.83	66.23	67.12	64.19	64.09
	TT	0.11	0.1	1.37	0.38	0.33	0.41	0.39	0.53	0.17
	CR	50.12	36.5	48.5	20.3	27.1	18.8	24.3	23.5	22.3
Ionosphere	AR	87.76	87.16	89.07	77.13	88.16	86.70	88.06	88.06	81.20
	TT	0.16	0.30	9.45	5.4	6.6	8.18	7.8	13.98	0.48
	CR	23.01	15	17.5	5.9	10.1	5.3	8.2	9	3.7
Cancer	AR	96.47	97.21	97.5	96.32	96.47	96.03	96.47	96.32	96.47
	TT	0.14	0.30	3.34	21.1	34.2	34.8	30.2	21.9	0.67
	CR	10.05	7.5	17.9	2.1	5	3	3.7	3.9	2.5
Zoo	AR	97.19	96.32	97.66	94.97	93.46	92.53	93.82	90.43	90.59
	TT	0.09	0.14	0.83	0.33	0.28	0.25	0.25	0.28	0.16
	CR	15.25	11.6	23.2	14.8	17	18.4	18.8	15	44.3
Heart	AR	83.88	81.95	85.57	77.36	82.10	79.73	80.31	81.09	76.41
	TT	0.11	0.17	1.56	0.89	0.97	1.09	1	1.2	0.14
	CR	41.11	30	42.6	11	16.1	11.1	12.8	13.5	14.3
TAE	AR	58.16	61.69	63.21	51.08	51.15	51.36	53.63	55.64	52.12
	TT	0.1	0.13	0.95	0.09	0.09	0.11	0.11	0.07	0.11
	CR	61.26	42.7	43.2	25.6	27.1	23	24.3	28.8	26.6
BLD	AR	65.95	65.41	65.93	57.38	60.34	59.98	62.75	63.92	60.22
	TT	0.15	0.23	2.4	0.56	0.5	0.63	0.55	0.7	0.12
	CR	58.84	42	47.9	23	29.9	19.3	25.2	23.4	18
New Thyroid	AR	94.89	96.28	97.31	90.83	93.85	95.27	94.42	93.61	93.55
	TT	0.09	0.13	0.92	0.27	0.36	0.33	0.33	0.45	0.14
	CR	13.14	9.8	8.7	6.2	11.3	7.2	8.4	7.6	8
SPECTF	AR	79.60	83.55	83.55	77.27	74.99	79.98	74.29	76.13	76.26
	TT	0.17	0.38	9.1	2.59	3.17	3.28	2.9	3.86	0.25
	CR	46.35	20.9	28.3	10	16.5	9.1	11.7	11.7	10.1
Ecoli	AR	83.97	83.58	86.44	81.22	85.94	83.53	86.64	84.36	83.17
	TT	0.14	0.17	1.32	0.98	0.84	1.44	1.28	1.36	0.25
	CR	36.01	17.5	24.6	9.5	14.6	12.00	12.7	12.2	11.3
AVERAGE	AR	83.51	84.47	85.86	79.02	81.76	81.67	82.18	82.28	80.16
	TT	0.12	0.19	2.3	2.76	4	4.25	3.78	3.75	0.23
	CR	32.01	21.2	27.5	11.9	16.4	12.5	14.4	13.8	16.2

Table 10. The performance of three incremental methods and six decremental methods.

and yields higher accuracy and generates fewer prototypes than CNN. Both GCNN and pGCNN generate fewer prototypes than CNN, because their training error rate e can be non-zero, while it is fixed at zero for CNN.

Since pGCNN is a special case of GCNN with $\rho = 0$, comparison of their accuracy rates offers us an opportunity to examine the sensitivity of GCNN to the parameter values. The difference between the average accuracy rates is 1.39%, but for the Ecoli and Heart data sets, the differences increase to 2.86% and 3.62% respectively, showing that the search for the optimal parameter values can be very useful. A similar situation is found with other types of APL.

7.3 Comparison of LAPL and KAPL with *k*-NN and SVM

To further evaluate the performance of the four APLs, we run two other alternative learning methods: *k*-NN, and SVM. Once again, for both methods, we apply cross-validation, similar to that used for APLs. For SVM, we employ the soft-margin version with the RBF kernel. Recall that the RBF function involves a parameter γ . In SVM, the value range of γ is taken as $\{a\times10^{-b}$: $a = 1, 2, \ldots, 9$ and $b = 3, 4, \ldots, 6\}$, which differs from that of KAPL by a factor of 10. Also, since the soft-margin version of SVM is used, there is an additional parameter C, which serves as a penalty factor for SVM training errors whose value range is taken as $\{10^c$: $c = -1, 0, \ldots, 5\}$. We use the LIBSVM toolkit (Hsu and Lin, 2002) to train SVM. For *k*-NN, the optimal value of *k* is determined during cross-validation, in much the same way that we optimize the *k* nearest prototypes for use in the voting procedure to determine the label of a test sample (cf. Section 6).

One crucial difference between SVM and APL is the way of dealing with multiclass data sets, that is, data sets comprised of more than two class types. Since SVM only deals with one binary classification at a time, we need to use a decomposition scheme when applying it to multiclass data sets. We employ one-against-others (Bottou et al., 1994) in our experiment. In other words, if there are *m* class types in total, we train *m* SVM classifiers, each of which classifies a sample as *A* or not *A*, where *A* is one of the *m* class types. One-against-one (Knerr et al., 1990; Platt et al., 2000) is an alternative decomposition scheme that allows us to train $m(m\text{-}1)/2$ classifiers. In our experience, the one-against-others scheme usually yields comparable or better accuracy rates than the one-against-one approach; however, the training cost is higher. For APLs, on the other hand, we construct prototypes for all class types simultaneously. Thus, in our experiments, there is *no* decomposition scheme for APLs.

The accuracy rates and training times of all the methods are given in Table 11. The boldface numbers have the same meaning as before, while the underlined numbers are the accuracy rates that are lower than the corresponding SVM results. As usual, we list the averaged results over all the 12 data sets at the bottom of the table. From the last results, we observe that all the APLs outperform *k*-NN in terms of accuracy; and GCNN is faster in training than SVM, but it is less accurate. The other three APLs incur higher training costs than SVM, but yield higher accuracy rates.

8 Conclusion

We have proposed a number of adaptive prototype learning algorithms that construct prototypes out of training samples. They differ in the use of samples or the weighted averages of samples as prototypes, and in the use of the Euclidean distance or a kernel-based distance. The algorithms can be further strenghened by allowing a non-zero training error rate, which improves the test accuracy. Our experiments, in which four types of APL were applied to 12 benchmark data sets, confirm the algorithms' efficacy in terms of test accuracy compared to many instance-based learning algorithms, the *k*-NN rule, and SVM.

DATA SET		APL Methods				Alternative Methods	
		GCNN	*f*-LAPL	*c*-KAPL	*f*-KAPL	*k*-NN	SVM
Iris	AR	96.62	97.95	98.63	98.40	97.03	96.47
	TT	0.8	30	16,225	81,334		70.52
Wine	AR	98.06	99.02	99.02	99.56	97.64	98.97
	TT	1	144	24,378	175,647		90.88
Glass	AR	69.39	71.26	72.23	72.73	70.40	69.43
	TT	1.37	314	18,906	108,891		299.68
Ionosphere	AR	89.07	91.46	95.88	95.87	86.72	95.08
	TT	9.45	8,010	399,693	3,078,420		362.20
Cancer	AR	97.5	97.79	97.50	97.79	96.91	97.06
	TT	3.34	2,301	496,817	5,265,013		321.92
Zoo	AR	97.66	97.66	97.66	97.66	96.55	95.86
	TT	0.83	11	21,666	135,346		153.64
Heart	AR	85.57	86.90	85.83	86.43	83.77	84.83
	TT	1.56	1,134	72,925	607,436		130.00
TAE	AR	63.21	62.47	65.22	65.61	57.78	64.23
	TT	0.95	229	18,682	133,157		605.24
BLD	AR	65.93	67.34	67.72	70.52	63.90	71.19
	TT	2.4	3,379	232,211	1,378,124		1181.68
New Thyroid	AR	97.31	97.76	98.57	99.05	96.31	97.78
	TT	0.92	135	19,289	134,671		78.80
SPECTF	AR	83.55	85.63	86.13	87.04	79.90	81.48
	TT	9.1	8,820	167,428	1,363,339		143.88
Ecoli	AR	86.44	86.81	86.18	87.06	87.33	88.13
	TT	1.32	920	35,151	216,235		421.88
AVERAGE	AR	85.86	86.84	87.55	88.14	84.52	86.71
	TT	2.3	2,119	126,948	1,056,468		321.72

Table 11. The performance of the four APLs, *k*-NN and SVM.

Acknowledgements

We wish to thank Po-Han Kuo and Chien-Hsing Chou for providing some of the experimental results.

References

P. Bartlett and J. Shaw-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*. MIT Press, 1999.

J. C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum, New York, 1981.

L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, L. Jackel, Y. LeCun, U. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of classifier methods: A case study in handwriting digit recognition. In *Proc. Int. Conf. Pattern Recognition*, pages 77–87, 1994.

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

C. Cortes and V. Vapnik. Support vector machines. In *Machine Learning*, 20:1-25, 1995.

T. Cover and P. Hart. Nearest neighbor pattern classification. In *IEEE Trans. Information Theory*, 13:21-27, 1967.

N. Cristianini and J. Shawe-Taylor. *An introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

F. Devi and M. Murty. An incremental prototype set building technique. In *Pattern Recognition*, 35:505–513, 2002.

L. Devroye and L. Györfi. *Nonparametric Density Estimation: The L1 View*. John Wiley, New York, 1985.

L. Devroye, L. Györfi, A. Krzyżak, and G. Lugosi. On the strong consistency of nearest neighbor regression function estimates. In *Annals of Statistics*, 22:1371-1385, 1994.

L. Devroye, L Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, New York, 1996.

E. Fix and J. L. Hodges. Discriminatory analysis. Nonparametric discrimination: Consistency properties. Technical Report 4, Project Number 21-49-004, USAF School of Aviation Medicine. Randolph Field, Texas, 1951.

E. Fix and J. L. Hodges. Discriminatory analysis: small sample performance. Technical Report 11, Project Number 21-49-004, USAF School of Aviation Medicine. Randolph Field, Texas, 1952.

E. Fix and J. L. Hodges. Discriminatory analysis. Nonparametric discrimination: Consistency properties. In B. Dasarathy, editor, *Nearest Neighbor Pattern Classification Techniques*, pages 32-39, IEEE Computer Society Press, Los Alamitos, 1991.

E. Fix and J. L. Hodges. Discriminatory analysis: small sample performance. In B. Dasarathy, editor, *Nearest Neighbor Pattern Classification Techniques*, pages 40-56, IEEE Computer Society Press, Los Alamitos, 1991.

F. Girosi. An equivalence between sparse approximation and support vector machines. In *Neural Computation*, 10(6):1455-1481, 1998.

P. Hart. The condensed nearest neighbor rule. In *IEEE Trans. Information Theory*, 14:515-516, 1968.

F. Höppner, F. Klawonn, R. Kruse, and T. Runkler. In *Fuzzy Cluster Analysis: Methods for Classification, Data Analysis and Image Recognition*. John Wiley & Sons, New York, 1999.

C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. In *IEEE Transactions on Neural Networks*, 13(2):415-425, 2002.

D.-W. Kim, K.Y. Lee, D. Lee, K. H. Lee. Evaluation of the performance of clustering algorithms in kernel-induced feature space. In *Pattern Recognition*, 38:607-611, 2005.

S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited: A stepwise procedure for building and training a neural network. In J. Fogelman, editor, *Neurocomputing: Algorithms, Architectures and Applications*, Springer-Verlag, New York, 1990.

T. Kohonen. *Self-Organization and Associated Memory*. Springer-Verlag, Berlin, 1988.

T. Kohonen. Statistical pattern recognition revisited. In R. Eckmiller, editor, *Advanced Neural Computers*, North-Holland, Amsterdam, pages 137-144, 1990.

Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. In *Pattern Recognition*, 19:84-95, 1980.

S. Lloyd. Least squares quantization in PCM. In *IEEE Trans. Information Theory*, 28:129-137, 1982.

J. Max. Quantizing for minimum distortion. In *IEEE Trans. Information Theory*, 6:7-12, 1960.

T. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. In *Philosophical Transactions of the Royal Society of London*, Series A, 209:415-446, 1909.

D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI Repository of machine learning databases, Irvine, CA: University of California, Department of Information and Computer Science, 1998. [http://www.ics.uci.edu/~mlearn/ MLRepository.html].

J. C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin DAG's for multiclass classification. In *Advances in Neural Information Processing Systems*, 12:547-553, MIT Press, Cambridge, 2000.

S. Salzberg. A nearest hyperrectangle learning method. In *Machine Learning*, 6:251-276, 1991.

B. Schölkopf, C. J. C. Burges, and A. J. Smola. In *Advances in Kernel Methods — Support Vector Learning*. MIT Press, Cambridge, MA, 1999.

C. Stone. Consistent nonparametric regression. In *Annals of Statistics*, 5:595-645, 1977.

V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Theory of Probability and Its Applications*, 16:264-280, 1971.

V. Vapnik and A. Chervonenkis. Ordered risk minimization I. In *Automation and Remote Control*, 35:1226-1235, 1974.

V. Vapnik and A. Chervonenkis. Ordered risk minimization II. In *Automation and Remote Control*, 35:1403-1412, 1974.

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

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

Z.-D. Wu, W.-X. Xie and J.-P. Yu. Fuzzy c-means clustering algorithm based on kernel method. In *Fifth Intern. Conf. Computational Intelligence and Multimedia Applications*, pages 1–6, 2003.

L. Zhao. Exponential bounds of mean error for the nearest neighbor estimates of regression functions. In *Journal of Multivariate Analysis*, 21:168-178, 1987.