8 Large Margin Adaptive 1-Nearest-Neighbour Classifiers

Abstract- This chapter introduces a learning strategy to design a set of prototypes for a 1-nearest-neighbour (NN) classifier. In learning phase, we transform the 1-NN classifier into a maximum classifier whose discriminant functions use the nearest models of a mixture. Then the computation of the set of prototypes is viewed as a problem of estimating the centres of a mixture model. However, instead of computing these centres using standard procedures like the EM algorithm, we derive to compute then a learning algorithm based on minimising the misclassification accuracy of the 1-NN classifier with high confidence of correct classification (or large margin) on the training set. One possible implementation of the learning algorithm is presented (*gaussian learn1NN*). It is based on the online gradient descent method and the use of radial gaussian kernels for the models of the mixture. As we show, *gaussian learn1NN* tends to use support vectors (Schölkopf, 1997) for computing class borders. Experimental results using hand-written NIST databases show the superiority of the proposed method over Kohonen's LVQ algorithms.

Index Terms- Nearest Neighbour Classifiers, Large Margin Classifiers, Support Vector Learning, Online gradient descent, Learning Vector Quantization, Hand-written Character Recognition.

Abbreviations- NN- Nearest Neighbour, LVQ- Learning Vector Quantization.

1. Introduction

Nearest neighbour (NN) methods are still among the most simple and successful for many (real-world) pattern recognition problems (Michie et al., 1994). The problem arises when the number of prototypes becomes large, since storage and computational requirements can be prohibitively expensive. Then we need to use techniques for reducing the number of prototypes like editing and condensing algorithms (see (Ripley, 1996)§6.2 or (Darasay, 1991) for a review on this topic) or other adaptive strategies like Kohonen's LVQ algorithms (Kohonen, 1996). This latter class of learning algorithms usually remains more powerful than editing and condensing algorithms. Several reasons can be argued to explain the good results of LVQ algorithms in real-world problems. First, the equilibrium points of these algorithms can ensure that NN prototypes' labels agree with the majority class of training data that falls in each Voronoi cell (see e.g. (Bermejo, 2000)). Hence, the resulting NN classifier is a data-dependent partitioning method (Devroye et al., 1996)§21. Besides, they let prototypes take arbitrary values and consequently allow designing a more flexible class of classifiers ((Devroye et al., 1996)§19.3). However one of the most favourable strategies for designing prototypes is to minimise the empirical classification error produced on the training set (p.311, (Devroye et al., 1996)) and LVQ algorithms do not explicitly address this minimisation. Other recent approaches also advocate the maximisation of the confidence (or margin) on the correct classifications (Smola et al., 1999).

This chapter introduces a learning strategy to design a set of prototypes for a 1-nearest-neighbour (NN) classifier based on minimising the training misclassification accuracy with large margin. Our approach reformulates the design of 1-NN's prototypes as a problem of estimating the centres of a mixture model since the NN rule is equivalent to a maximum classifier whose discriminant functions use the nearest models of a mixture for each class. Nevertheless, we do not compute these centres using standard method for mixtures like the EM algorithm (Dempster et al., 1977) since the goal of the classifier must be the minimisation the training classification error with high confidence (or large margin). Consequently, we introduce a loss function to compute these centres that ensure this minimisation. We present one implementation of the learning algorithm based on the online gradient descent method and the use of radial gaussian kernels for the models of the mixture that we call hereafter *the gaussian learn1NN algorithm.*

In the next section, we briefly review NN classifiers. Section 3 presents our approach to the design of the prototypes 1-NN classifiers. In Section 4 we introduce the gaussian

learn1NN algorithm. Section 5 shows experimental results comparing our algorithm with Kohonen's LVQ algorithms using the NIST hand-written character database (Garris et al., 1997). Section 6 shows the similarities of gaussian learn1NN and support vector learning (Schölkopf, 1997). The important topic of model selection is addressed in section 7. Finally, some discussion and conclusion are given in sections 8 and 9.

2. Nearest neighbour classifiers

Given a pattern **x** to classify, the K-nearest-neighbour classification rule is based on applying the following algorithm

i) Find the K nearest patterns to **x** in the prototypes set $\mathbf{P}=\{(\mathbf{m}_i, \mathbf{y}_i), i=1...M\}$ where $\mathbf{m}_i$ is a prototype that belongs to one of the classes and $\mathbf{y}_i$ is an class indicator variable. (The nth coefficient of $\mathbf{y}_i$ is equal to 1 and the other coefficients are equal to 0 when $\mathbf{m}_i$ belongs to the nth class.)

ii) Decide the classification by a majority vote amongst these K.

We have as typical design choices: the metric *d* to measure closeness between patterns, the number of neighbours *K*, the set of prototypes **P** and its size *M*. The most usual similarity measure $d(\mathbf{x},\mathbf{y})$ is the Euclidean distance $d(\mathbf{x},\mathbf{y})=\|\mathbf{x}-\mathbf{y}\|^2$. However other measures can be used like the Mahalanobis distance or even measures that have been learned with training sets (Friedman, 1994). K can be automatically selected using a validation set, although K=1 is a common choice due to

1) Euclidean 1-NN classifiers form class boundaries with piecewise linear hyperplanes so they can be used to solve a large class of classifiers since any border can be approximated by a series of hyperplanes defined locally.
2) Most of the learning algorithms that compute **P** from training data work with 1-NN classifiers (see below).

Finally, the design of the set of prototypes is the most difficult and challenging task. The most simple election is to select the whole training set $\mathbf{T}=\{(\mathbf{x}_j, \mathbf{y}_j), j=0...N-1\}$ (where $\mathbf{x}_j$ is a random sample of X and $\mathbf{y}_j$ is the class indicator variable) as **P**. Nevertheless, this simple choice requires big memory and execution requirements in large databases so in practice a reduced set of prototypes of size M (with M<<N) is a mandatory. There are three main techniques to reduce the number of stored prototypes:

1) **Condensing algorithms**. Since only training data that are near class border are useful for classification, condensing procedures aim to keep those points from training data which form class boundaries (e.g. Hart's condensed 1-NN classifier (Hart, 1968)).
2) **Editing algorithms**. They retain those training patterns that fall inside class borders that are estimated with the same training set. These algorithms tend to form homogeneous clusters since only the points that are at the centre of natural groups in data are retained (e.g. (Wilson, 1972)).
3) **Clustering algorithms**. It is also feasible to use any clustering algorithm (e.g. K-means) to form a set of labelled prototypes. First, we obtain a set of unlabeled prototypes from training data using the clustering algorithm. These prototypes then can be used to divide the input space in M nearest-neighbour cells. Finally, we can assign labels to prototypes according to a majority vote of training data in each cell (see (Devroye et al., 1996)§21.5). However it is also possible to compute labelled centroids using a one-step learning strategy like Kohonen's LVQ algorithms do. E.g. the equilibrium points of LVQ1 are a particular kind of labelled class centroids which ensure that the resulting Euclidean 1-nearest-neighbour classifier discriminates according to a majority vote of training data in each Voronoi region (Bermejo, 2000).

Clustering algorithms seems preferred to condensing and editing algorithms since if we let that **P** has arbitrary values, prototypes are not constrained to training points and then a more flexible class of classifiers can be designed ((Devroye et al., 1996)§19.3). However one of the most favourable strategies for designing prototypes is to minimise the empirical classification error produced on the training set **T** (p.311, (Devroye et al., 1996)), and clustering methods do not explicitly address this minimisation. Other recent approaches like large margin classifiers (Smola et al., 1999) also address the maximization of the confidence on the correct classification since generalization can be then improved.

3. Learning with 1-NN classifiers as an adaptive mixture of nearest models

3.1. Maximum classifiers, NN classifiers and Bayes error.

Maximum classifiers (Duda & Hart, 1973) use a set of so-called discriminant functions (one for each class) to classify a pattern **x** according to the following rule:

$$\mathbf{x} \in \text{Class } i \Leftrightarrow d_i(\mathbf{x}) = \min_{j=1,\ldots,C} d_j(\mathbf{x}) \qquad (1)$$

where $\{d_j(\mathbf{x}), j=1,...,C\}$ are the discriminant functions. These functions can be unbounded on X. However if $\{d_j(\mathbf{x})\}$ map $\Re^d \rightarrow [0,1]$ subject to the constrained $\sum_{j=1}^{C} d_j(\mathbf{x}) = 1$ then the resulting functions can be interpreted as probabilities measures.

The statistical goal of learning in a classification procedure is (in the case of equally misclassification costs) to design a classifier c with the minimum expected misclassification rate or probability of error $P(E_c)=E_x[E_c \backslash \mathbf{x}]$. In maximum classifiers, $P(E_c)$ is minimised when the discriminant function of class i, d_i, is the posterior probability of belonging to class i given a random vector **x**, $P(C_i \backslash \mathbf{x})$ (Ripley, 1996). However a good estimation of $P(C_i \backslash \mathbf{x})$ is only needed near class borders to ensure that the maximum classifier approximates the Bayes rule. K-Nearest neighbour classifiers estimate $P(C_i \backslash \mathbf{x})$ with a ratio between K_i (the number of prototypes that belong to class i among K nearest neighbours of **x**) and K (Bishop, 1995). For the special case of K=1, a different view can be given. Prototypes must be placed during learning to approximate Bayes decision boundaries. If the metric is the Euclidean distance, the approximation is done with piecewise linear boundaries, which are formed with each segment corresponding to the perpendicular bisector between two nearest prototypes of different classes. Nevertheless, as we will show in the following section, 1-nearest-neighbour classifiers can be also derived from kernel estimation. This alternative formulation help us to derive a useful learning rule based on minimising the number of misclassifications with large margin.

3.2. Maximum classifiers with kernel estimators

As we have just seen a possible way of constructing maximum classifiers is estimating posterior class probabilities. Since $P(C_i \backslash \mathbf{x})=E[\mathbf{y}_i \backslash \mathbf{x}]$ we can employ non-parametric kernel regressors (Scott, 1992)§8 to estimate these probabilities and use them to design the set of discriminant functions as follows:

$$d_i(\mathbf{x}) = \frac{\sum_{j=1}^{N} y_{ij} G(\mathbf{x} - \mathbf{x}_j; \gamma)}{\sum_{j'=1}^{N} G(\mathbf{x} - \mathbf{x}_{j'}; \gamma)} \quad i = 1,..., C \tag{2}$$

where kernel G is a bounded and even function on X that is peaked about **0**, γ denotes the locality control parameters associated to G. This estimation is derived from a locality principle,

that is any function using Taylor's theorem can be locally approximated with a constant. In fact, kernel estimators of equation (2) are the solution of the following local weighted least-squared estimation problem:

$$d_i(\mathbf{x}) = \arg\min_a \sum_{j=1}^{N} G(\mathbf{x} - \mathbf{x}_j; \gamma)(a - y_{ij})^2 \quad i = 1,...,C \tag{3}$$

where *a* is a constant.

Applying Bayes formula $P(C_i|\mathbf{x}) = p(\mathbf{x}|C_i)P(C_i)/p(\mathbf{x})$ where $p(\mathbf{x})$ is the density function of random vector X, $p(\mathbf{x}\backslash C_i)$ is the density function of class i and $P(C_i)$ is the prior probability of belonging to class i, then the set of discriminant functions of equation (2) can also be derived using the following non-parametric kernel density estimates (Scott, 1992)§6.

$$\hat{p}(\mathbf{x}|C_i) = \frac{1}{N_i}\sum_{j=1}^{N} y_{ij} G(\mathbf{x} - \mathbf{x}_j; \gamma), \quad \hat{P}(C_i) = \frac{N_i}{N}, \quad \hat{p}(\mathbf{x}) = \frac{1}{N}\sum_{j'=1}^{N} G(\mathbf{x} - \mathbf{x}_{j'}; \gamma) \quad i = 1,...,C \tag{4}$$

where N_i is the number of training samples that belong to class i.

The shape of the kernel G determines different solutions to the same original problem. If $G=G_H$ where G_H is a square kernel whose width is adjusted to contain exactly k samples then equation (2) is the *K-NN* algorithm. On the other hand, if $G=G_S$ where G_S is a smooth kernel with a width modulated by a locality parameter σ then (2) is the *Parzen window* estimator (also known as the Nadaraya-Watson estimator (Tarter & Lock, 1993) (Ripley, 1996)§6.1). Furthermore, equation (2) with $G=G_S$ can be considered as a particular form of normalised radial basis functions (RBFs) (Bishop, 1995)§5 since

$$d_i(\mathbf{x}) = \sum_{j=1}^{C} w_{ji}\, \Phi_j(\mathbf{x}) \quad \text{with } w_{ji} = 1(i=j)\hat{P}(C_j) \text{ and } \Phi_j(\mathbf{x}) = \frac{\hat{p}(\mathbf{x}|C_j)}{\hat{p}(\mathbf{x})} \quad i = 1,...,C \tag{5}$$

where 1(u) is the indicator which is 1 if condition is true and 0 otherwise and the probability and density estimators of equation (4).

Finally, we can obtain the 1-NN algorithm if equation (2) with $G=G_S$ is approximated by

$$d_i(\mathbf{x}) \approx \frac{G(\mathbf{x}-\mathbf{x}_W^i;\gamma)}{\sum_{j=1}^{C} G(\mathbf{x}-\mathbf{x}_W^j;\gamma)} \quad i=1,...,C \tag{6}$$

where $\mathbf{x}_W^j$ is the nearest training pattern to $\mathbf{x}$ that belong to class j using the distance metric d and $G(\mathbf{x}-\mathbf{x}_W^j;\gamma)=G(d(\mathbf{x},\mathbf{x}_W^j);\gamma)$. Note that equations (6) and (1) induce the same region partitions than the NN algorithm of section 2 for K=1. Since equation (6) is a simplification of Parzen windows, the 1-NN algorithm is then a poor estimator of posterior class probabilities. However in classification, accuracy is only important in class borders. Therefore, maximum classifiers with simplified Parzen windows (a.k.a. 1-NN classifiers) are useful since. if d is the Euclidean distance, the induced class borders are built with local piecewise hyperplanes, so it is possible to approximate locally a vast class of problems, as we have seen before.

3.3. From kernel regressors to adaptive mixture estimation

Maximum classifiers that use kernel regressors have an important drawback since the whole training set must be retained to compute the discriminant functions. However, replacing the kernel estimation with mixture models (McLachlan & Basford, 1988) can simply extend this approach and solve the memory requirements of kernel regression since the total number of mixture models can be typically much smaller than the training set size. Suppose we approximate each class density function with the following mixture model

$$\hat{p}(\mathbf{x}|C_i)=\sum_{j=1}^{M_i} w_{ij}\, G(\mathbf{x}-\mathbf{w}_j^i;\gamma) \text{ with } w_{ij}=\frac{1}{M_i} \quad i=1,...,C \tag{7}$$

where M_i is the number of the mixture models of class i, $\mathbf{w}_j^i$ are the centres of the mixture models of class i and G(**u**) is defined as before. If $\hat{P}(C_i)=M_i/M$ where M is the total number of mixture models, then the density funcion p(**x**) can be approximated by

$$\hat{p}(\mathbf{x})=\sum_{i=1}^{C}\hat{p}(\mathbf{x}|C_i)\hat{P}(C_i)=\frac{1}{M}\sum_{i=1}^{C}\sum_{j=1}^{M_i} G(\mathbf{x}-\mathbf{w}_j^i;\gamma) \tag{8}$$

Finally, the discriminant functions as estimators of posterior class probabilities give

$$d_i(\mathbf{x}) = \hat{P}(C_i|\mathbf{x}) = \frac{\sum_{j=1}^{M_i} G(\mathbf{x} - \mathbf{w}_j^i; \gamma)}{\sum_{i'=1}^{C} \sum_{j=1}^{M_{i'}} G(\mathbf{x} - \mathbf{w}_j^{i'}; \gamma)} \quad i = 1,...,C \tag{9}$$

Equation (9) is a particular normalised RBF based on mixture models. The total number of mixture models M will be typically much smaller than the number of training points N, and the mixture models centres $\mathbf{w}_i^j$ are no longer constrained to coincide with the training points. The centres of the mixture models could be estimated using unsupervised procedures (as the EM algorithm (Dempster et al., 1977)) for each class or using standard supervised learning procedures based on the minimisation of the sum-of-squares error or the cross-entropy error (Bishop, 1995)§6.

3.4. Adaptive mixture of nearest models for learning with 1-NN classifiers.

We can simplify equation (9) as follows

$$d_i(\mathbf{x}) \approx \frac{G(\mathbf{x} - \mathbf{w}_W^i; \gamma)}{\sum_{i'=1}^{C} G(\mathbf{x} - \mathbf{w}_W^{i'}; \gamma)} \quad i = 1,...,C \tag{10}$$

where $\mathbf{w}_W^j$ is the nearest centre of the mixture model to $\mathbf{x}$ that belong to class j using the distance metric d (e.g. $d = \|\mathbf{x} - \mathbf{w}_W^j\|^2$) and $G(\mathbf{x} - \mathbf{w}_W^j; \gamma) = G(d(\mathbf{x}, \mathbf{w}_W^j); \gamma)$. Now, the discriminant functions of the maximum classifier are only formed with the nearest mixture model of each class. Note that equations (10) and (1) form a 1-NN classifier whose prototypes are the centres of a mixture model that are not constrained to training points. Besides, the number of the prototypes of a 1-NN classifier (M) is much smaller than the number of training points. With this formulation, the design of the set of prototypes of 1-NN classifiers is transformed into a problem of computing the discriminant functions of a maximum classifier based on the nearest mixture models that approximate class density functions. As before, the centres of mixture models (that is the prototypes of the 1-NN classifier) could be estimated using standard adaptive procedures. However, one of the most advantageous strategies for designing prototypes for classification purposes is to minimise the empirical classification error

contained on the training set **T** with large margin. In the next points, we will introduce and analyse the loss function whose global minimum points ensure that the number of misclassifications in the training set was minimised with a high confidence on the correct classifications.

3.4.1. The Loss Function

A natural measure of performance to drive learning could be the number of misclassifications. However this measure is not a good direct candidate since it is a binary function and consequently can cause problems to optimisation methods (e.g. those procedures based on gradient descent (Duda & Hart, 1973)). As a loss function, we propose a modification of the cross-entropy error function for multiple classes:

$$L = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=1}^{C} y_{ji}\, d_j(\mathbf{x}_i) \tag{11}$$

The absolute minimum with respect to the $\{d_j(\mathbf{x_i})\}$ occurs when all the training points are **correctly classified with the highest confidence** (or maximum margin), that is when $d_j(\mathbf{x_i})=y_{ji}$ for all j and i. An interesting property of equation (11) is that the training set's outliers cause a lower impact than in the cross-entropy error ((Bishop, 1995)§6.9) case. (Since outliers typically can provoke a very low activation of their corresponding discriminant function, $\log d_{j'}(\mathbf{x}_{outlier})$ with j'=class label($\mathbf{x}_{outlier}$) gives a very negative value so the cross-entropy error function is very sensitive to outliers.) We will see in the next section when the learning equations are derived for a particular choice of the kernel G, how the inclusion of the logarithmic function in equation (11) modifies the form of the cross-entropy function and thus the dynamics of the learning system.

In the limit in which the size N of the training set goes to infinity, we can substitute the sum over patters of equation (11) with the following integral

$$\langle L\rangle = \lim_{N\to\infty} L = -\int_{\Re^d}\sum_{j=1}^{C} y_j(\mathbf{x})\, d_j(\mathbf{x})\, p(\mathbf{x})\, d\mathbf{x} \tag{12}$$

where p(**x**) is the probability density of the input space. Since y_j=**1**(**x**∈ class j) where **1**(u) is the indicator function which is 1 if condition u is met and 0 otherwise. Equation (12) can be decomposed as

$$\langle L \rangle = -\sum_{j=1}^{C} \left(\int_{R_j} d_j(\mathbf{x}) p(\mathbf{x}|C_j) P(C_j)\, d\mathbf{x} + \int_{R_j} \sum_{\substack{i=1 \\ i \neq j}}^{C} d_i(\mathbf{x}) p(\mathbf{x}|C_i) P(C_i)\, d\mathbf{x} \right) \tag{13}$$

where p(**x**\$C_i$) are the class densities, P(C_i) are the prior probabilities and R_j are the decision region of class j, that is the input region in which d_j has the highest value among discriminant functions. The solution to the minimization problem of equation (13) is d_j=1 and p(**x**\$C_j$)P($C_j$)=máx p(**x**\$C_i$)P($C_i$) for all the points in each R_j. Then the minimum of equation (13) gives

$$\langle L_{min} \rangle = -\sum_{j=1}^{C} \int_{R_j} p(\mathbf{x}|C_j) P(C_j)\, d\mathbf{x} \tag{14}$$

Then the Bayes error P(E_B)=1+<L_{min}>. In practice, the absolute value of <L_{min}> will converge to the maximum probability of correct classifications when the number of prototypes was large enough to hold d_j=1 for all the points in each R_j.

4. The Gaussian Learn1NN Algorithm

In this section we derive and study the learning algorithm based on online gradient descent using the loss function of equation (11) and a radial gaussian function for the kernel G of the mixture models.

4.1. Online gradient descent for 1-NN classifiers

In real-life pattern recognition problems training sets are usually large and high dimensional. Therefore, algorithmic simplicity of the learning procedure is a necessity to get a solution using moderated computational resources. Online gradient descent algorithms (Bottou, 1998) are one of the most simple computational approaches to the learning problem. Empirical evidence supports the use of on-line algorithms over batch versions since that convergence speed is accelerated when training data is redundant, a typical situation in real-life data (p.264 (Bishop, 1995), (Bengio, 1991)). Another typical argument in favour of on-line versions is that they can be considered as a 'noisy' version of batch algorithms so they could escape from a local minimum easier. Furthermore, large-sample convergence of these

algorithms can be studied using the stochastic approximation theory (Benveniste at al., 1990)(Bottou, 1998).

Our goal in the learning phase is to compute a set of labelled codebooks $\mathbf{C}_j=\{\mathbf{w}_i^j,\ i=1,...,M_j\}$ j=1,...,C, one for each class (with $\sum_{j=1}^{C} M_j = M$), for a 1-nearest-neighbour classifier that minimises a estimator of the expected loss function,

$$\langle L\rangle\left(\{C_j\}_{j=1}^{C}\right)=E_X\left[Q\left(\mathbf{x},\{C_j\}_{j=1}^{C}\right)\right] \tag{15}$$

where $Q\left(\mathbf{x},\{C_j\}_{j=1}^{C}\right)$ is here $\sum_{j=1}^{C} y_j\, d_j(\mathbf{x})$ according to equation (12). Since $E_X[\bullet]$ is unknown, an empirical estimator is formed with a (training) set of random samples pairs $\mathbf{T}=\{(\mathbf{x}_j,\ \mathbf{y}_j),$ j=0...N-1}. Then the online gradient learning algorithm estimates $E_X[\bullet]$ using the instantaneous loss function $Q\left(\mathbf{x},\{C_j\}_{j=1}^{C}\right)$. Each step of this algorithm consists of picking up (cyclically or randomly) one sample pair from the training set **T** and applying the following update equation

$$\mathbf{w}_i^j[k]=\mathbf{w}_i^j[k-1]-\eta_i^j[k]H\left(\mathbf{w}_i^j[k-1],\mathbf{x}[k]\right)\quad k>0 \tag{16}$$

where **x**[k] is a sample of the training set (e.g. **x**[k]=$\mathbf{x}_{\text{K-1 mod N}}$ for cyclic sampling) and the learning rates $\eta_i^j[k]$ are positive numbers which are usually made to decrease monotonically with discrete time k (or they can be also constant with time). The term $H(\mathbf{w}_i^j, \mathbf{x})$ is defined as

$$H\left(\mathbf{w}_i^j,\mathbf{x}\right)=\begin{cases}\nabla_{\mathbf{w}_i^j} Q\left(\mathbf{x},\{C_j\}_{j=1}^{C}\right) & \text{when differentiable}\\ 0 & \text{otherwise}\end{cases} \tag{17}$$

4.2. The gaussian learn1NN algorithm

Let $G\left(d(\mathbf{x},\mathbf{w});\gamma\right)=\exp\left(-\|\mathbf{x}-\mathbf{w}\|^2/2\sigma\right)$, $L=-\sigma/N\sum_{i=0}^{N-1}\sum_{j=1}^{C} y_{ji}\, d_j(\mathbf{x}_i)$ and $\eta_i^j[k]=\eta$. We have modified the loss function for analytical convenience since the added constant does not affect the desired points of convergence. Hence the pattern version of gradient descent over L gives

$$H(\mathbf{w}_i^j, \mathbf{x}) = \begin{cases} -d_j(\mathbf{x})(1-d_j(\mathbf{x}))(\mathbf{x}-\mathbf{w}_i^j) & \text{if } \mathbf{x} \in \text{class j and } \mathbf{w}_i^j = \mathbf{w}_W^j \\ d_j(\mathbf{x})d_{j'}(\mathbf{x})(\mathbf{x}-\mathbf{w}_i^j) & \text{if } \mathbf{x} \in \text{class j' and } \mathbf{w}_i^j = \mathbf{w}_W^j \\ 0 & \text{otherwise} \end{cases} \quad (18)$$

where $\mathbf{w}_W^j$ is the nearest prototype of class j to x in the Euclidean sense, that is $\mathbf{w}_W^j = \arg\min_{\mathbf{w}_i^j} \left\|\mathbf{x}-\mathbf{w}_i^j\right\|^2$. Finally the learning equation (16) is

$$\begin{aligned} &\text{if } \mathbf{x}[k] \in \text{class j}, \mathbf{w}_i^j[k-1] = \mathbf{w}_W^j[k-1] \\ &\quad \mathbf{w}_i^j[k] = \mathbf{w}_i^j[k-1] + \alpha\, d_j[k-1](\mathbf{x}[k])(1-d_j[k-1](\mathbf{x}[k]))(\mathbf{x}[k]-\mathbf{w}_i^j[k-1]) \\ &\text{if } \mathbf{x}[k] \in \text{class l}, \mathbf{w}_i^j[k-1] = \mathbf{w}_W^j[k-1] \\ &\quad \mathbf{w}_i^j[k] = \mathbf{w}_i^j[k-1] - \alpha\, d_j[k-1](\mathbf{x}[k])d_l[k-1](\mathbf{x}[k])(\mathbf{x}[k]-\mathbf{w}_i^j[k-1]) \\ &\text{otherwise} \\ &\quad \mathbf{w}_i^j[k] = \mathbf{w}_i^j[k-1] \end{aligned} \quad (19)$$

where $\mathbf{x}[k] = \mathbf{x}_{(k-1) \bmod N}$ $k \geq 0$ if we pick up cyclically. Note that we can also derive a simplified learning rule for the case $\sigma \to \infty$. Then all the discriminant functions tend to 1/C (where C is the number of classes) since $\exp(-\bullet/2\sigma) \to 1$.

4.3. Study of large-sample convergence

General convergence proofs for the online gradient descent (Benveniste at al., 1990)(Bottou, 1998) can be directly applied to the above learning algorithm. Given a training set of infinite size, the necessary (but not sufficient; see (Bottou, 1998)) conditions to ensure the convergence of the iterative equation (19) to a local minimum of the loss function (equation (11)) are the following:

$$\text{i) } E_X[H(\mathbf{w}_i^j, \mathbf{x})] = \nabla_{\mathbf{w}_i^j} L(\{C_j\}_{j=1}^C) \quad (20)$$

$$\text{ii) } \left\|H(\mathbf{w}_i^j[k], \mathbf{x}[k+1]) - H(\mathbf{w}_i^j[k-1], \mathbf{x}[k])\right\| << \left\|H(\mathbf{w}_i^j[k], \mathbf{x}[k+1])\right\| \quad (21)$$

Condition (20) can be rewritten using (17) and explicit integration operators as

$$\int H\left(\mathbf{w}_i^j, \mathbf{x}\right) p(\mathbf{x}) d\mathbf{x} = \{\text{when differentiable}\} = \int \nabla_{\mathbf{w}_i^j} Q\left(\mathbf{x}, \{C_j\}_{j=1}^C\right) p(\mathbf{x}) d\mathbf{x} \stackrel{?}{=}$$
$$\stackrel{?}{=} \nabla_{\mathbf{w}_i^j} \int Q\left(\mathbf{x}, \{C_j\}_{j=1}^C\right) p(\mathbf{x}) d\mathbf{x} \qquad (22)$$

The integration and differentiation operator can be swapped if the maximal slope of Q is conveniently bounded. This happens when Q is differentiable and gradient Q is integrable. In our case, the instantaneous loss function Q is not differentiable on points located on the Voronoi boundaries induced by the subsets of prototypes of each class but, since the iterative algorithm have zero probability to reach these points (Bottou, 1998), condition (20) is met. (In any case, if a point falls in these regions during learning, we can simply take another sample.)

Condition (21) checks that the constant step alpha is small enough to ensure convergence to a local minimum (Benveniste at al., 1990). However, checking this condition can be computational expensive. Instead, we can use a simple strategy. If we use the early stopping technique to avoid overtraining (that is to stop at a minimum of the validation set), we can use several values of alpha and choose that alpha which gets to the deepest minimum of the classification error of the validation set.

4.3. Properties of gaussian Learn1NN

4.3.1. The Gaussian loss function.

Equations (10) and (11) with $G(d(\mathbf{x}, \mathbf{w}); \gamma) = \exp\left(-\|\mathbf{x} - \mathbf{w}\|^2 / 2\sigma\right)$ yield the gaussian loss function L_{gauss} which is minimised to compute the prototypes of a Euclidean 1-nearest-neighbor classifier. Like the smoothness parameter h in kernel regression ((Scott, 1992)§8) and kernel classification ((Devroye et al., 1996)§10), the parameter σ controls how local the discriminant functions $\{d_i\}$ are, that is how many training data contributes to form the value of the discriminant functions at a given test point $\mathbf{x}$. (In other words, σ regulates the degree of smoothness of the solution. So if σ is large, the value of $d_i(\mathbf{x})$ is softly interpolated though the sum of many more gaussians centred at the training points.) Then, σ manipulates the number of local maximums of $\{d_i\}$ and consequently the number of local minimum points of L_{gauss}. Let us illustrate the effect of σ with an example. Suppose we have 10 training points that belongs to classes a and b: -1.0 (a), 0.5 (a), 1.0 (b), 2.5 (a), 3.0 (a), 4.0 (b), 4.2 (b), 5.0 (a), 6.0 (b), 7.0 (b). This classification problem can be solved with only two prototypes w_A (a) and

w_B (b) and the frontier point $F=(w_A+w_B)/2$. The minimum classification error is achieved when $3 \le F \le 4$ and $w_A < w_B$. Then test patterns will be classified to class a when $x<F$. Figure 1 shows the contour levels of L_{gauss} for several values of σ and the line F=3.5. The attraction basins of L_{gauss} are in the region $w_A<w_B$. As expected, σ controls the number of local minimum points of L_{gauss}. E.g. there are 9 local attraction basins, one for each frontier F between two training points when σ=0.325. As σ grows the local minimum points disappear and the global attraction basins is moved toward the point $(-\infty,+\infty)$ subject to the constraint F=3.5.

4.3.2. The minimum points of L_{gauss}.

Suppose that we only have one prototype for each class. Then $\nabla_{\mathbf{w}^j} L_{gauss} = 0$ yields

$$\mathbf{w}^j = \frac{\sum_{i=0}^{N-1} y_{ji}\, d_j(\mathbf{x}_i)\left(1-d_j(\mathbf{x}_i)\right)\mathbf{x}_i - \sum_{i=0}^{N-1}\sum_{\substack{k=1\\k\neq j}}^{C} y_{ki}\, d_j(\mathbf{x}_i) d_k(\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=0}^{N-1} y_{ji}\, d_j(\mathbf{x}_i)\left(1-d_j(\mathbf{x}_i)\right) - \sum_{i=0}^{N-1}\sum_{\substack{k=1\\k\neq j}}^{C} y_{ki}\, d_j(\mathbf{x}_i) d_k(\mathbf{x}_i)} \quad j=1...C \tag{23}$$

According to equation (23), prototypes only depend on few training samples. Since prototypes are a function of weighted data, only those training points that have a significant activation of their corresponding weight function contribute to form prototypes. Each prototype $\mathbf{w}^j$ are formed with the following subset of training data:

i) Samples belonging to the class j which are near the class border since the weight function of these samples is $d_j(1-d_j)$ and reaches its maximum for $d_j=0.5$.

ii) Samples belonging to any other class which are near the border of class j since the weight function of these samples is $d_j d_k$ and reaches its maximum for $d_j=0.5$ and $d_k=0.5$.

Remember that the minimum points of L_{gauss} ensure a minimum number of misclassifications. Consequently, the set of prototypes that solve $\nabla_{\mathbf{w}^j} L_{gauss} = 0$ are formed with the sub-set of training samples near class borders which are hard to classify, that is *hard boundary points* (Breiman, 1998). Let us review again the above example to see that optimal prototypes depend on hard boundary point. Figures 1 and 2 show that the global minimum point (w_A,w_B) of L_{gauss} is $(-\infty,+\infty)$ subject to the constraint F=3.5. Clearly F only depends on points 3.0 (a) and 4.0 (b) which are the hardest training points to classify since they are the nearest data to the

frontier point that minimises the classification error. But why is the minimum point of L_{gauss} achieved at (-∞,+∞) with F=3.5? As we pointed out before, the absolute minimum of L force $d_j(\mathbf{x_i})=y_{ji}$ for all j and i. This is not possible if the best possible solution misclassify some training samples. Then the global minimum of L ensures the minimum number of misclassifications and besides forces those samples $\{\mathbf{x_i}\}$ correctly assigned to class j with $d_j(\mathbf{x_i})=1$. In our problem, the discriminant functions are $d_a(x)=1/(1+\exp(-\beta u))$ and $d_b(x)=1-d_a(x)$ where $\beta=2\sigma^{-1}(w_B-F)$ and u=F-x. Thus, L_{gauss} has a global minimum when $d_a(x_i)=1$ for all $x_i<3.5$ that belongs to class a and $d_a(x_i)=0$ for all $x_i>3.5$ that belongs to class b. This force $\beta\rightarrow\infty$ so $w_B\rightarrow\infty$ subject to the constraint $(w_A+w_B)/2=3.5$. Then $d_a(x)=\mathbf{1}(x<3.5)$ where **1** is the indicator function and the test samples are classified, according to the 1-nearest-neighbour rule, to class a if x<3.5 and to class b if x>3.5.

4.3.3. L_{gauss} vs. the gaussian cross-entropy error.

The proposed loss function has the same absolute minimum points than the cross-entropy error function for multiple classes,

$$L_{cross-entropy}=-\frac{1}{N}\sum_{i=0}^{N-1}\sum_{l=1}^{C}y_{li}\ln d_l(\mathbf{x}_i) \tag{24}$$

However, the cross-entropy loss function is a much more sensitive function. It penalises those solutions that make a very low activation of some training patterns in their discriminant function. These patterns include outliers and some patterns that are wrongly classified and far from class borders. Suppose that the training pattern $\mathbf{x}_i$ belonging to class j has a very low activation of $d_j(\mathbf{x}_i)$ for a given solution of the set of prototypes. Then the term $-y_{ji}\ln d_j(\mathbf{x}_i)$ take a very large value and accordingly this solution is strongly penalised. Figures 3a and 3b shows the contour levels of equation (24) using the radial gaussian kernel. As one can see, the minimum point of the loss function depends on sigma and is subject to F≈3.5. Those solutions with very low activation of the discriminant function are penalised so w_A and w_B are chosen to built a function $d_a(F)=1/2$ that has a slow pendent around the point F. Let us now introduce the additional training sample -7.0 (b) to this problem. (The new problem has the same solution than before: 3≤F≤4 and $w_A<w_B$.) The inclusion of this point does not affect the minimum points of L_{gauss} as we can see in figures 2c and 2d. However, $L_{gauss\ cross\text{-}entropy}$ is strongly affected (figures 3c,3d and 4). In fact the solution that minimises the number of misclassification is vanished and

the new minimum points are $w_A=w_B$. The sample -7.0 penalises the solution F=3.5 since $d_b(-7)<<1/2$. Then the only feasible solution is $w_A=w_B$ since $d_a=d_b=1/2$ for all the training points and accordingly $L_{cross\text{-}entropy}$ gives a low value (ln(2)=0.70). There is another way to stress the differences between our proposed loss function and the cross-entropy error. If we derive the on-line gradient equations for the cross-entropy error function, we obtain the equation (18) divided by $d_j(\mathbf{x})$. This modification in the learning equations cause the prototypes to be modified for training data belonging to the same class that are not near class borders.

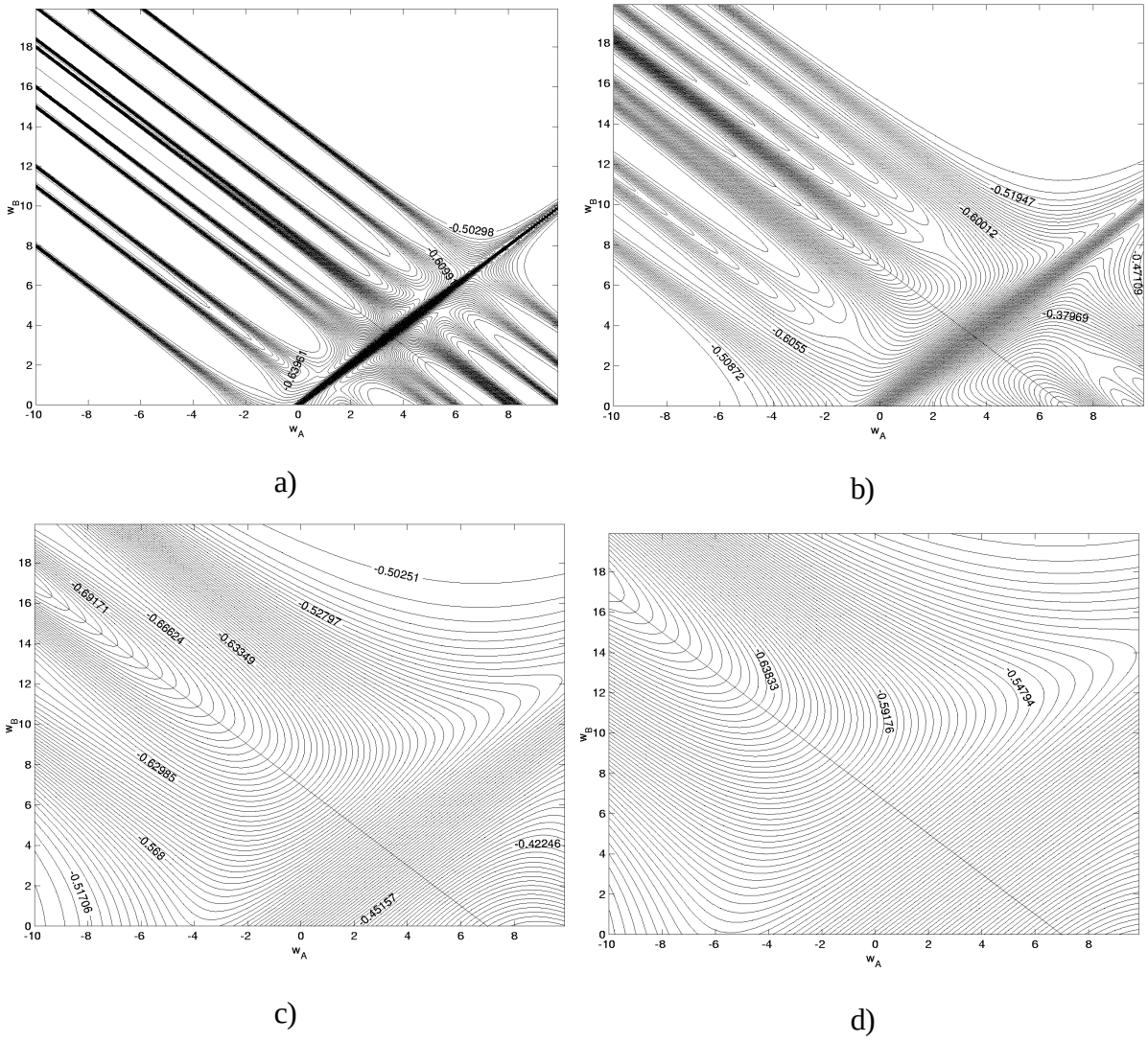
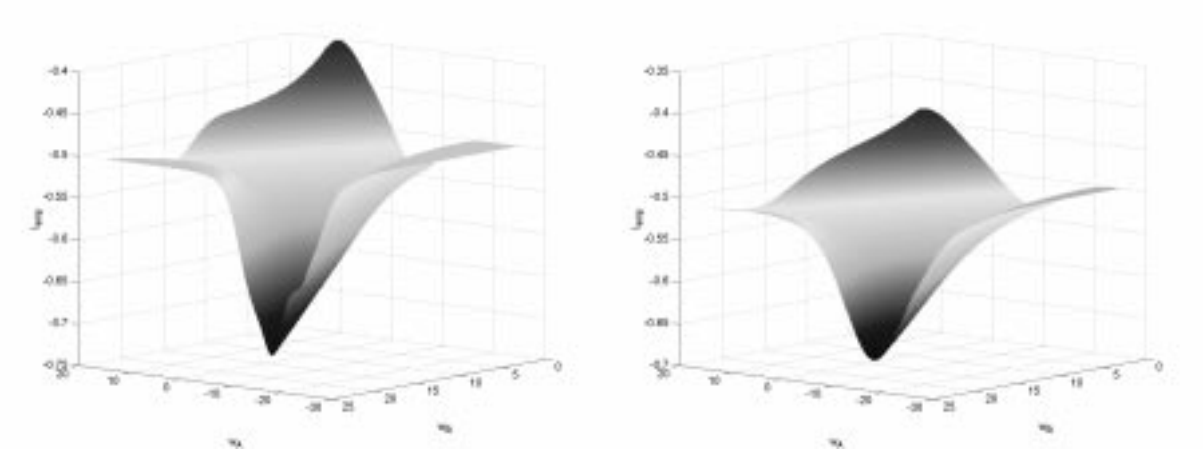
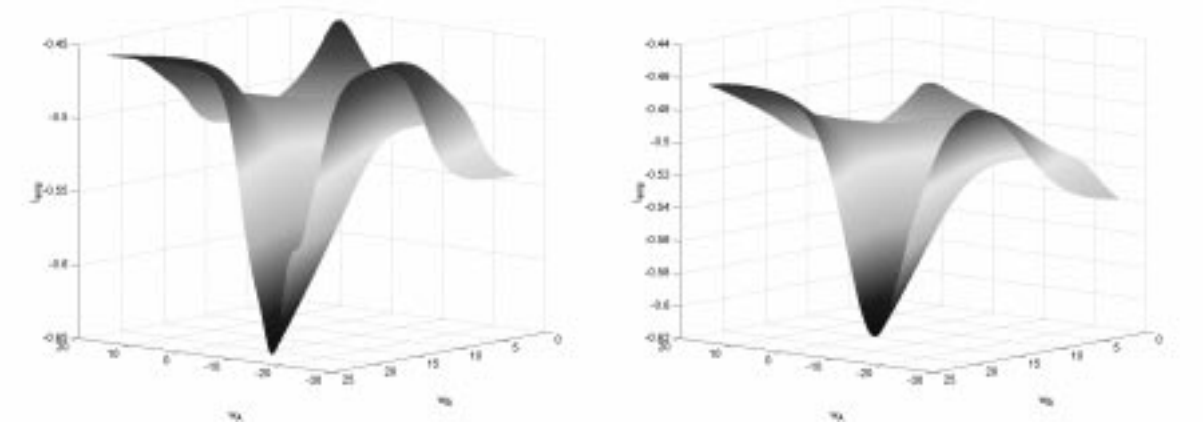


Fig. 1. Contour levels of L_{gauss} for the example and the solution line F=3.5: a) σ=0.325, b) σ=2.5, c) σ=12.5 and d) σ=25.

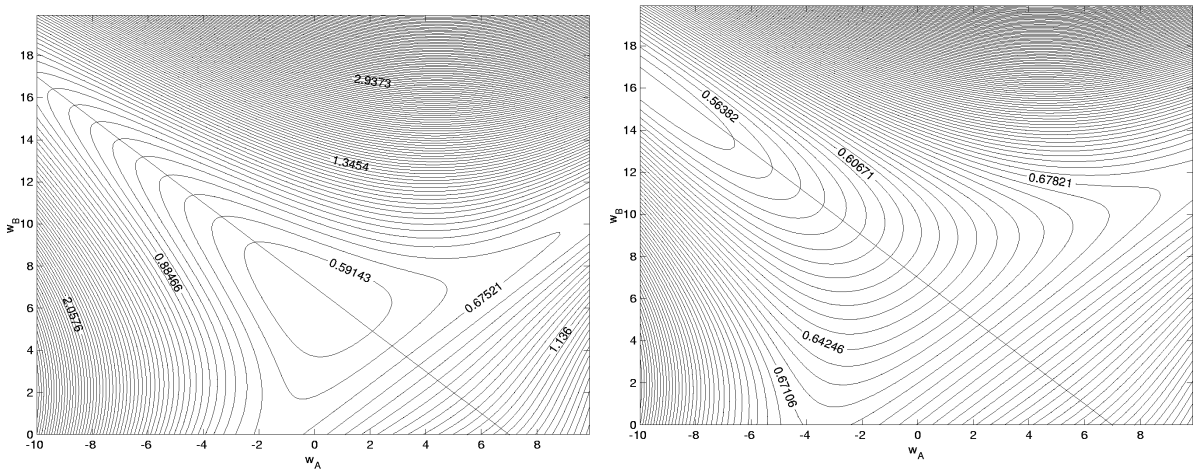


a) b)

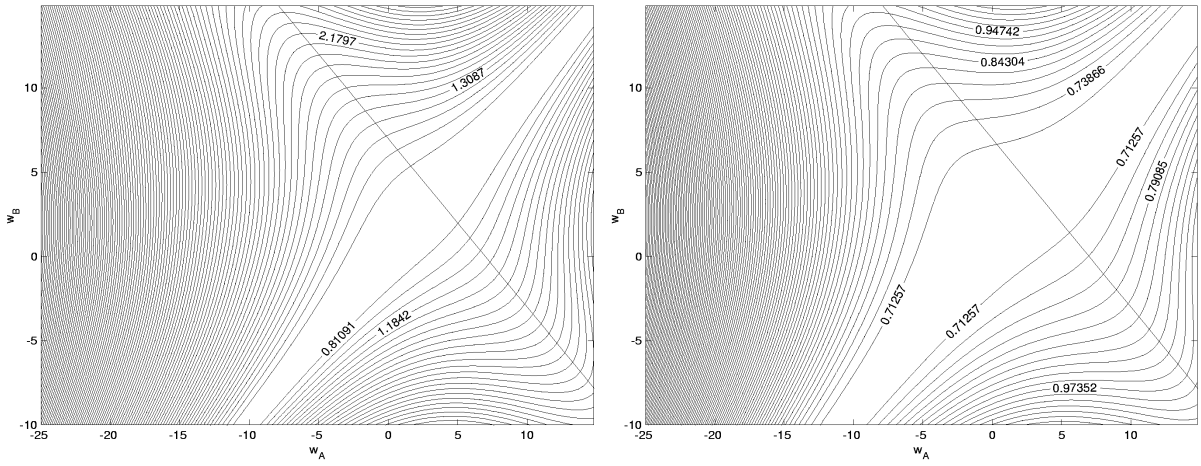


c) d)

Fig. 2 The loss function L_{gauss} for the same example of figure 1: a) σ=25, b) σ=50; and for the same example with the inclusion of an outlier: c) σ=25, d) σ=50.

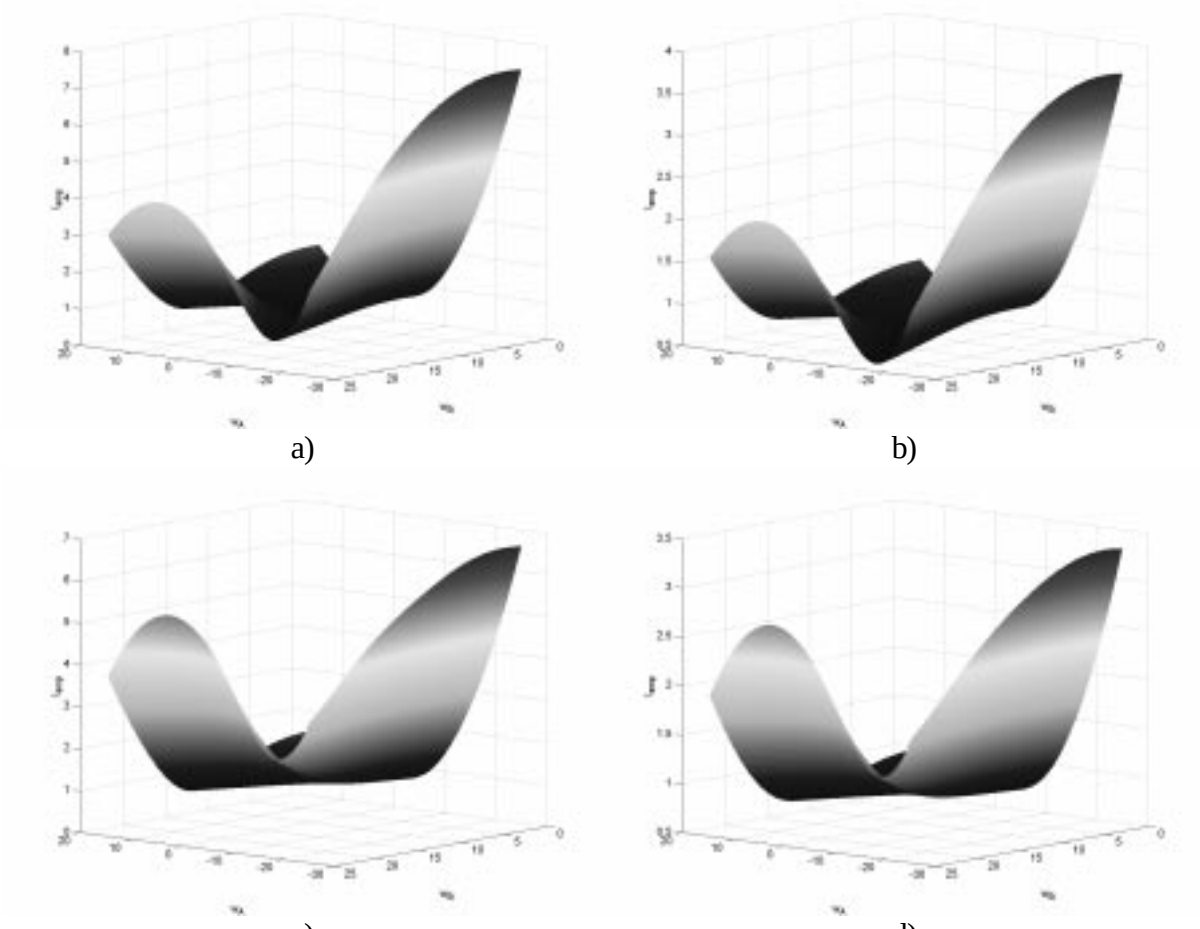


a) b)



c) d)

Fig. 3. Contour levels of $L_{\text{gauss cross-entropy}}$ for the example and the solution line F=3.5: a) σ=12.5 and b) σ=50; and for the same example with the inclusion of an outlier: c) σ=12.5 and d) σ=50.



a) b)

c) d)

Fig. 4. The loss function $L_{\text{gauss cross-entropy}}$ for the same example of figure 1: a) σ=25, b) σ=50; and for the same example with the inclusion of an outlier: c) σ =25, d) σ=50.

5. Experimental Results

In this section, we compare LVQX algorithms (where X is 1, 2 and 3) with our proposal using the LVQ_PAK (Kohonen et al., 1995). We have employed the NIST (uppercase & lowercase) hand-written characters database (Garris et al., 1997) and the synthetic problem appeared in (Ripley, 1994).

5.1. Hand-written character problems

5.1.1. NIST Database and its pre-processing

This hand-written database can be found in (Garris et al., 1997) in directories train/hsf_4, train/hsf_6 and train/hsf_7. We have split directory train/hsf_7 in two sets (one for validation and one for test) preserving the original class distribution in data. The other two directories were chosen for training (see table 1). These images (32x32 pixels) have been preprocessed with a Principal Component Analyzer that extracts 64 components from each image (NIST's mis2evt utility). The correlation matrix of the PCA was computed using the training set.

Database	Training Set Size	Validation Set Size	Test Set Size
Upper	24420	1031	10453
Lower	24205	1078	10914

Table 1. Size of NIST hand-written sets.

5.1.2 Simulations

Ten runs for each classification problem, learning algorithm and several sizes of the set of prototypes (or codebook) have been made. We have initialized the codebooks using *eveninit* and *balance* programs. *Eveninit* selects an equal number of prototypes to each class. These prototypes are training points that fall inside an estimation of class borders. (A training point is said to fall inside class borders if this sample is well classified with a K-NN classifier using the rest of the training set.) The set of prototypes selected with *eveninit* is then passed through the Balance program. In balance, the medians of the shortest distance between the prototypes of each class are computed. Then, the program adds new prototypes or deletes old ones if the distances turn out very divergent for different classes. Finally, *balance* runs OLVQ1 one epoch. See (Kohonen et al., 1995) for additional information. Once the initial value of the prototypes is computed, we have applied every learning algorithm until classification error in

validation set increases or stands. (We have monitored this error every 200000 steps. In the case of LVQX algorithms, every time we monitored the error, we restart its value). The optimal parameters of algorithms have been estimated using the validation set.

5.1.3 Results

Table 2 summarises the average classification error of uppercase and lowercase test sets for Gaussian learn1NN + the 1-NN classification rule and 1-NN classifiers whose codebooks are designed using Kohonen's LVQ algorithms. Each result is the average of ten trials for each learning algorithm. Gaussian learn1NN always outperforms LVQ algorithms for each codebook size. In uppercase and lowercase recognition, best results are achieved when codebook size is 800. On average, our learning system correctly recognised 93,68% of the uppercase hand-written letters and 86,68% of the lowercase hand-written characters. On the hand, LVQ2, the best of the LVQ algorithms, only achieves 91,74% and 85,97% respectively. Differences between our procedure and LVQ algorithms are notably greater for codebooks of small size. This behaviour could be related with the very different nature of both approaches. LVQ algorithms compute a particular kind of cluster centroids. In the LVQ1 algorithm, these centres tend to class means (e.g. those centroids computed for each class with the K-means) as the number of prototypes decreases. Then class boundaries are poorly approximated. The other two LVQ algorithms are also affected by this behaviour in less degree. On the other hand, Gaussian learn1NN places prototypes to minimise the training classification error with large margin. So the algorithm always seeks the best possible solution given a fixed size of the set of prototypes. Figure 6 also shows the standard deviation in the test error. All the learning algorithms tend to exhibit a decrease of this statistic as the codebook size is increased. Our algorithm has the lowest variance in 4 out of 8 cases. LVQ3 only wins in two cases. Finally, LVQ1 and LVQ2 achieve the best deviation in one case. In the situations where Learn1NN is not the winner, it does not greatly differ from the other non-winner LVQ algorithms.

In figure 5 we can see average validation error using our learning procedure for several values of the parameter σ. Experimental results display better classification error for small values of σ. As section 4 shows, σ controls the number of local minimum points of the loss function but also the region where the attraction basin to these points is located. In the example of section 4, the attraction basins of the global minimum is pushed towards the point $(-\infty,+\infty)$ as σ grows. Since class regions are unbounded for the problem, this behaviour does not affect the solution. However if we increase σ in problems where class regions are

bounded, minimum points can be pushed outside these regions and the solution can be degraded as simulations seems to corroborate.

No. of prototypes	Upper				Lower			
	Lvq1	Lvq2	Lvq3	*Gaussian learn1NN*	Lvq1	Lvq2	Lvq3	*Gaussian learn1NN*
100	19.3 (1.03)	13.36 (1.39)	16.02 (1.3)	12.29 (0.96)	25.02 (0.53)	21.5 (1.36)	22.29 (1.15)	19.14 (0.98)
200	15.08 (1.1)	11.29 (0.85)	11.4 (0.63)	9.91 (1.13)	20.28 (0.81)	17.15 (0.63)	18.39 (1.25)	16.67 (1.03)
400	11.11 (0.59)	9.06 (0.47)	9.08 (0.488)	7.22 (0.465)	17.19 (0.36)	15.14 (0.48)	15.54 (0.48)	14.52 (0.28)
800	9.34 (0.484)	7.25 (0.29)	8.26 (0.54)	**6.32** (0.23)	15.76 (0.37)	14.03 (0.35)	14.43 (0.3)	**13.32** (0.43)

Table 2. Experimental Results for the NIST databases. We show for each algorithm the average test error and the standard deviation over ten runs.

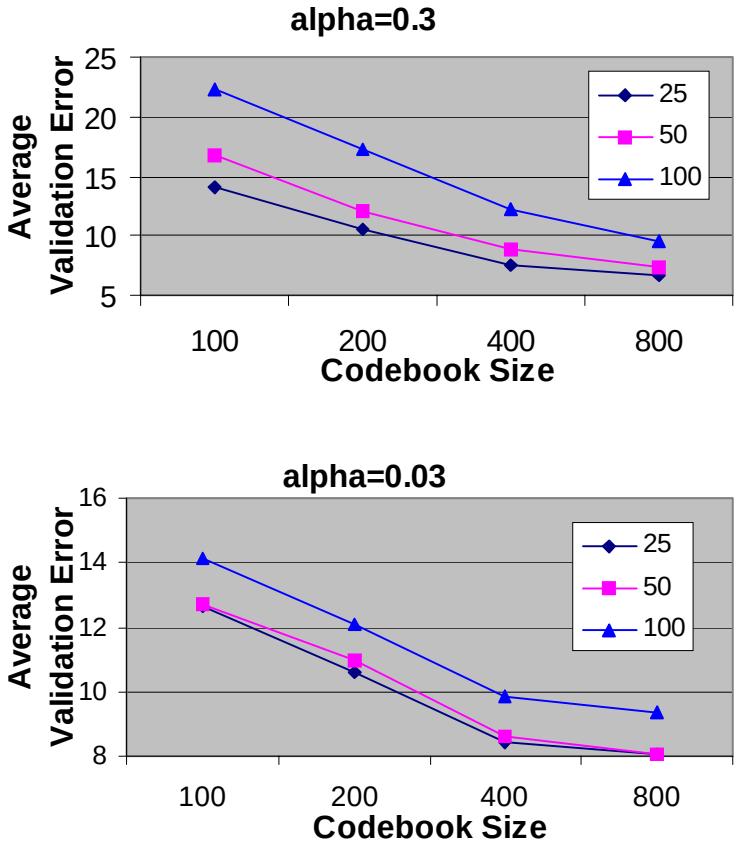


Fig.5. The effect of parameter σ in the classification accuracy. We show the average validation error (over ten runs) for α=0.3 and 0.03, several codebook sizes and σ=25, 50 and 100.

5.2. Ripley's synthetic problem.

The synthetic data appeared in (Ripley, 1994) is a two-class problem where each population is an equal mixture of two 2-D gaussian clusters. We use a training set and a test set of sizes 250 and 1000 respectively. These sets are the same than those reported in (Ripley, 1994). The learning phase is stopped when iterative algorithms achieved a local minimum of the training misclassification accuracy. (More precisely, we check the training classification error each 200 epochs to decide if learning must be stopped.) Since we do not use any validation set to stop learning, the overtraining phenomena can be present. The parameters of the algorithms have been estimated using an additional validation set of size 250. We only report results on learn1NN and LVQ1 since the other algorithms give worse accuracy. The same number of prototypes was assigned to each class. 100 runs were performed varying the ordering of data in memory and the initial conditions of the set of prototypes. The initial prototypes were chosen using the LVQ_PAK's eveninit program (Kohonen, 1995). The Bayes rule gives for this problem an accuracy of 8%. Table 3 shows the misclassification rate for several sizes of the set of prototypes. The best average result is achieved by learn1NN for 32 prototypes (8.764%). However both learning systems achieve the same absolute minimum (8,2%) that is superior to the best result reported in (Ripley, 1994): the 8.3 % of the 5-NN classifier after multiedit. The best result for Learn1NN with 2 prototypes (10.2%) outperforms linear discriminant (10.8%) and logistic discriminant (11.4%) in (Ripley, 1994) and the LVQ1 with 2 prototypes (27.8%). (The interest in this comparison is that all three classifiers have the same approximation capabilities.) Notice in figures 6 and 7 that the solution of learn1NN for 32 prototypes does not overfit training data (in fact it is a solution that could be implemented with fewer protoypes) while LVQ1's does. Moreover LVQ1 is more sensitive to initial conditions as the number of prototypes grows since the standard deviation is greater than learn1NN's.

No. of prototypes	Learn1NN				LVQ1			
	Average	Variance	Max.	Min.	Average	Variance	Max.	Min.
2	10.908	0.489	11.7	10.2	27.963	0.058	28.1	27.8
4	9.327	0.3278	11.2	8.5	**8.801**	0.01	8.9	8.8
6	9.212	0.2066	9.6	8.6	9.363	0.598	11	8.4
8	9.163	0.2236	9.6	8.4	9.694	1.0	16.4	**8.2**
16	8.846	0.2607	9.6	8.3	10.197	1.57	16.6	**8.2**
32	**8.764**	0.2456	9.5	**8.2**	11.503	1.795	16.2	9.3
64	8.93	0.4694	10.6	8.3	12.3	1.521	15.7	9.9

Table 3. Experimental Results for the NIST databases. We show for each algorithm the average test error, the standard deviation and the maximum and minimum achieved over 100 runs. We mark in bold type the best average and total solutions in both learning algorithms.

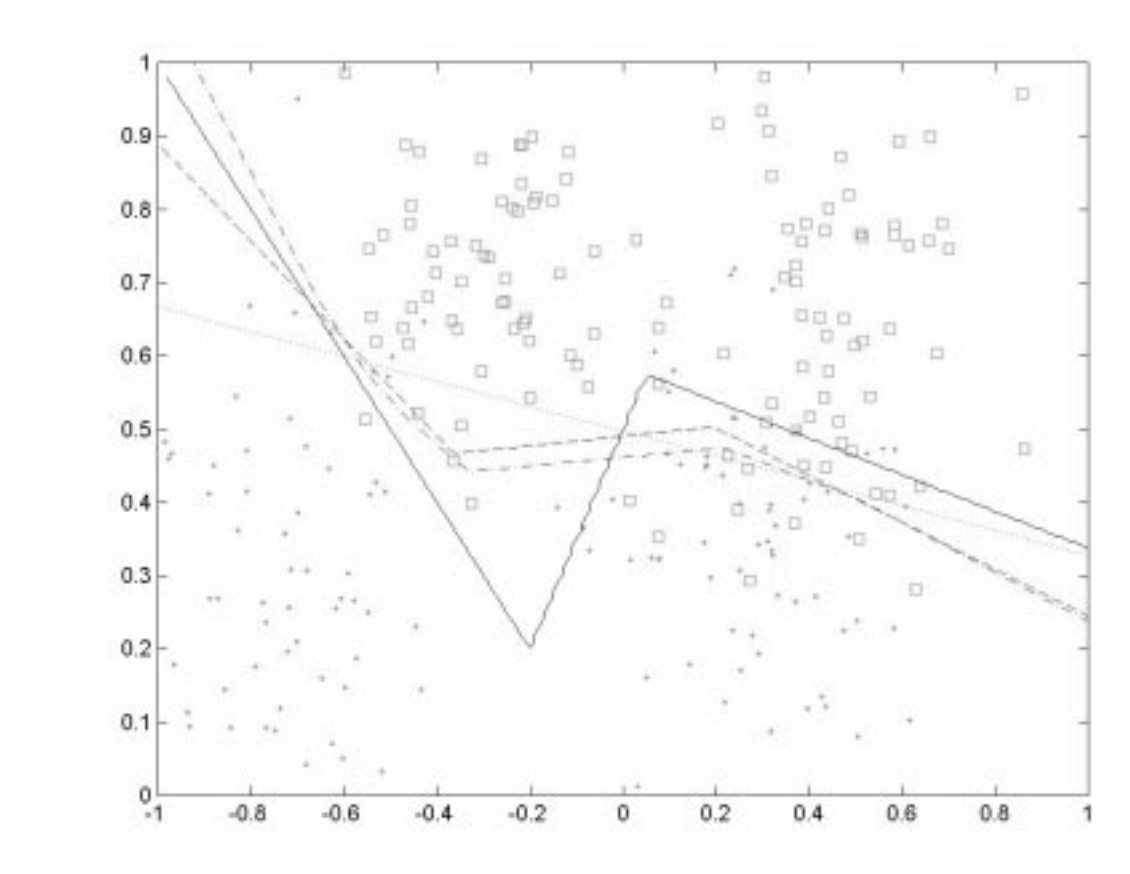


Fig.6. Ripley's synthetic training set (the two classes are denoted by dots and squares) with the Bayes border (solid line) and the class borders computed with lear1NN for 2 (dotted line), 6 (dashed line) and 32 (dashdot line) prototypes. The test error for these classifiers was 10.7%, 9.4% and 8.4 % respectively. Note that the NN classifier with 32 prototypes do not overfit training data.

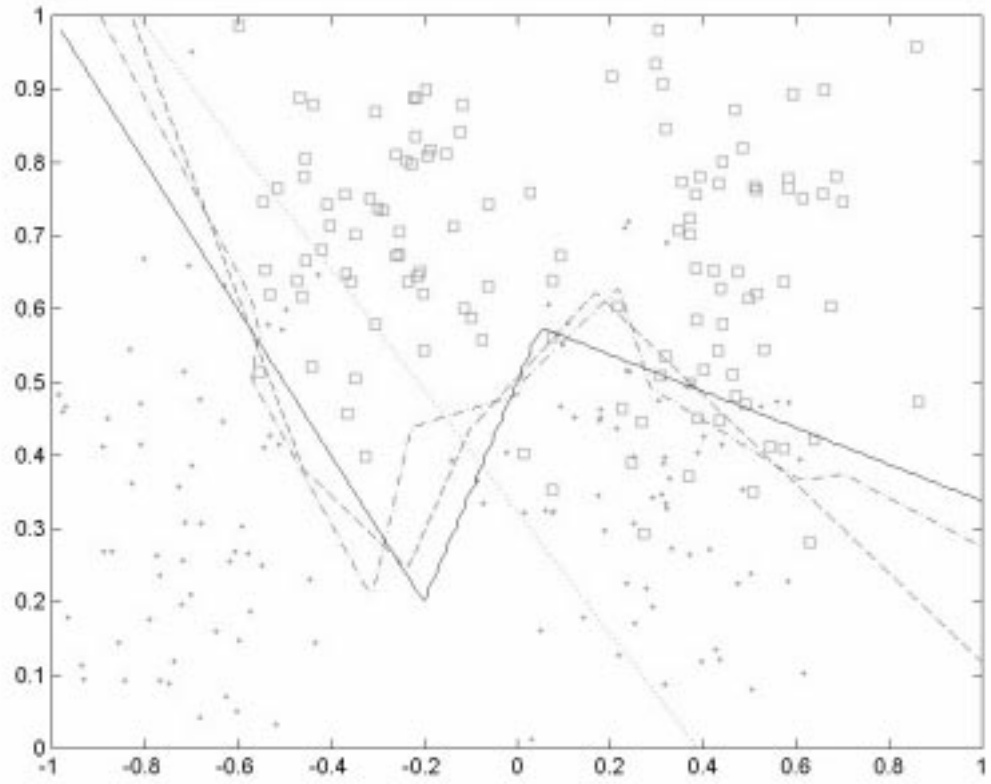


Fig.7. Ripley's synthetic training set with the Bayes border (solid line) and the class borders computed with LVQ1 for 2 (dotted line), 6 (dashed line) and 32 (dashdot line) prototypes. The test error for these classifiers was 27.7%, 8.6% and 9.4 % respectively. Note that the LVQ-based NN classifier with 32 prototypes tends to overfit training data while the classifier with 2 prototypes achieves a very poor solution.

6. Gaussian Learnn1NN, Support Vector Learning and Large Margin Classifiers

Figure 8a shows a two-class classification problem that is linearly separable. (Dots and squares indicate the two classes.) Consequently, a hyperplane $w^T x+b$ (where T denotes transpose) can be properly placed to achieve 0% misclassification rate. In this case, the solution is not unique since there are many lines that achieve this accuracy. For instance, the perceptron algorithm (Nilsson, 1990) could compute any of them since it halts when gets to the solution region and for different initial points it will arrive to this region at different points. However, it would desirable to place a line in the "middle" between the two classes since then one could expect a better generalization performance. We show why with an example. Suppose a test pattern x that is a slight variation of a training point x_i (e.g. $x=x_i+\varepsilon$ for a small ε). Then the test point might be assigned to the same class than x_i since x is arbitrarily close to it (considering a smoothness assumption). Now consider x_i is an extreme (or outlying) point of one of the classes (e.g. the point (6,8)). If the line is near x_i then we cannot ensure that x is classified to the same class. Hence, the optimal hyperplane (OH) (optimal in the sense that a slightly variation in training points induce the same solution) must be placed to achieve the largest separation between the extreme points. This hyperplane can be computed using the OH algorithm (addendum I (Vapnik, 1982), (Cortes, 1995)§3). It can be shown (e.g. (Burges, 1999)) that the OH only depends on the extreme training points. These vectors are also called *support vectors* since the solution vector w^* can be expanded in terms of them as a linear combination. Accordingly (p.25 (Cortes, 1995)),

$$\mathbf{w}^* = \sum_{\mathbf{x}_i \in X} \alpha_i \mathbf{x}_i - \sum_{\mathbf{x}_i \in \Xi} \beta_i \mathbf{x}'_i \qquad (25)$$

where $\{\alpha_i >0\}$, $\{\beta_i >0\}$ are the Lagrange multipliers of the two sets of support vectors X and Ξ (see e.g. (Cortes, 1995),(Burges, 1999) for additional details). In figure 8a, the support vectors are marked with a circle (points (6,8) and (9,5)) and the OH is the continuos line $x_2=x_1-1$. On the other hand, the dashed line is the class border computed with the gaussian learn1NN algorithm for 1-NN classifier with only two prototypes. This border (almost) coincides with the OH. In fact, this surprising result is a simple consequence of the nature of the equilibrium points of the gaussian learn1NN. As we pointed before, the computed prototypes only depend on those points

with an activation (of the corresponding discriminant function) near 0.5. Figure 8b shows the discriminant function d of class 'dots' where blue and red denotes a low and high activation respectively. Since this function has an abrupt transition between "0" and "1" levels around the OH, only support vectors have a significant contribution of the term d(1-d) in equation 23. Hence, the resulting frontier line only depends on support vectors. Figures 8c and 8d show the results for the same problem, but now the number of prototypes of the 1-NN classifier is 12 and 50 respectively. Since the training set has only 11 points, the solutions might overfit data. Again, the learning algorithm fins a solution with 0 % classification error. However, overfitting is not present. The capacity of the learning machine is being controlled since the solution is only formed with two lines. In spite of the great number of parameters, the learning algorithm finds a solution with a restricted number of effective parameters since many prototypes converge to the same point (see figure 10a).

Let us now introduce a classification problem with a minimum classification error greater than zero, which can be solved with a line. Figure 9a shows this problem. We consider again the extreme points that are near the optimal class border (that is, the class borders that achieves the minimum error). These points (marked with a circle) are located at the line $x_2=x_1+0.5$ for squares and at $x_2=x_1-2.5$ for dots. The OH that uses these points is the continuos line $x_2=x_1-1$. Our algorithm converges to it again (the discontinuous line in figure 9a). Thus, only extreme points that ensure the minimum classification error are used to form class boundaries. Note that in our problem there are clusters of squares and dots far from the optimal border. These training points are irrelevant to form class boundaries and our algorithm does not use them since the activation of their corresponding discriminant function is very high or very low (see figures 9b, 9c and 9d). The overfitting phenomenon is not present again since the algorithm finds a solution with a small number of effective parameters (see figure 10b).

These striking results can be explained if we interpret our algorithm in the context of large margin classifiers (Smola et al., 1999). Denote by $d:\Re^p \rightarrow [0,1]$ the discriminant function used for a two-class pattern recognition problem. Then the margin is defined by

$$\rho_d(\mathbf{x},z)=z\left(d(\mathbf{x})-1/2\right) \tag{26}$$

where z is +1 if **x** belongs to class 1 and –1 otherwise. When $\rho_d>0$, it indicates the margin by which **x** is classified correctly. A negative value of ρ_d agrees with a misclassification. Besides, we define the minimum margin of the classifier over the whole training set as

$$\rho_d = \min_{i=0...N-1} \rho_d(\mathbf{x}_i, z_i) \tag{27}$$

The minimum margin measures the "worst " classification on the training set **T**. For problems with small overlapping between classes, it could be useful to build classifiers with large margin (Smola et al., 1999). (E.g. the solution of this max-min margin optimization in problems with no class overlapping is the OH.) However when class overlap is big, large margin classifiers are determined entirely by those misclassified samples that are furthest from the optimal class border. However if we compute a solution that minimises the training classification error, it would be interesting that the resulting classifier will also have a large margin near the optimal class border. Hence, we could define a local minimum margin

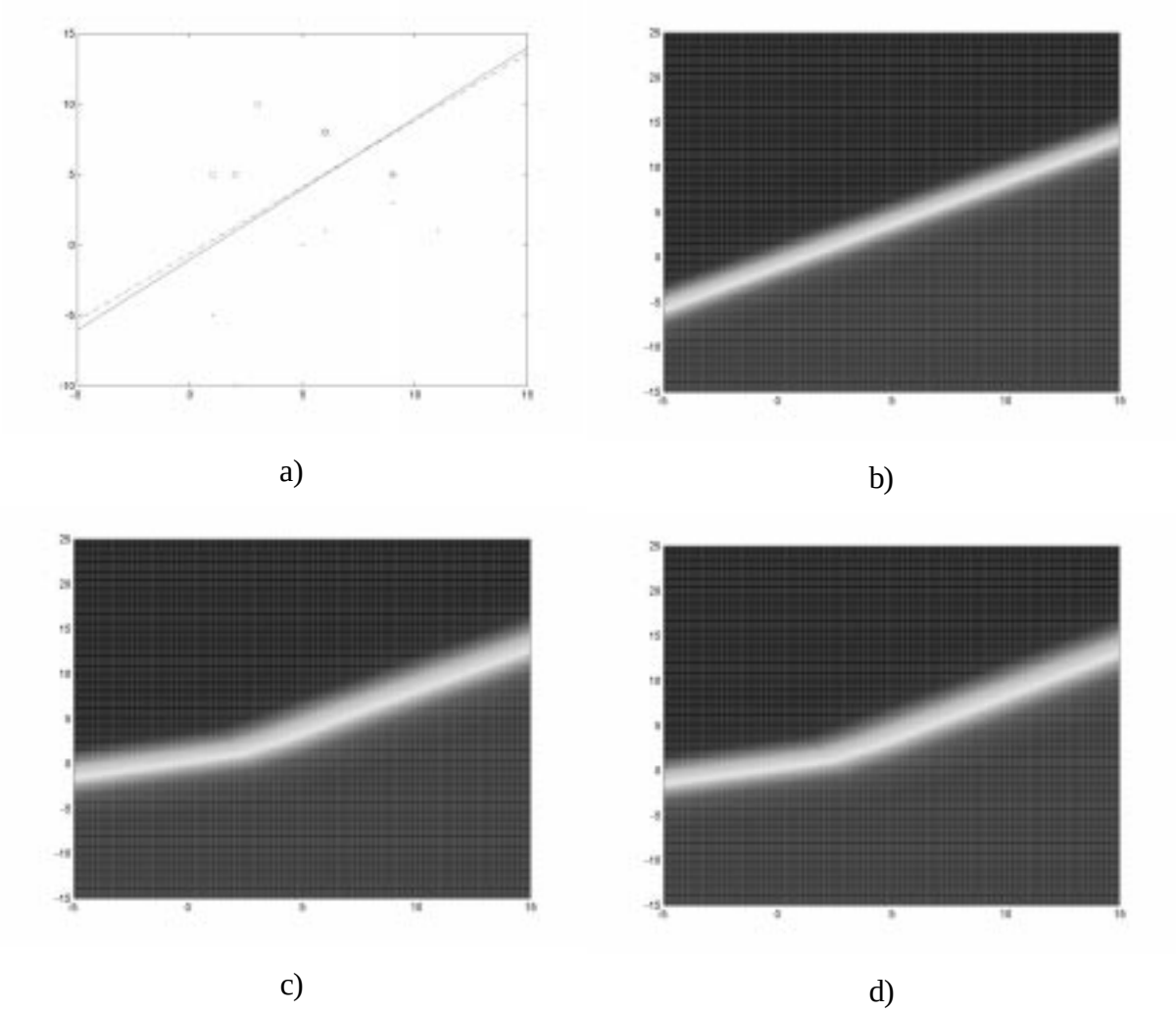
$$\rho_d^l = \min_i \rho_d(\mathbf{x}_i, z_i), \mathbf{x}_i \in \Gamma \tag{28}$$

where the set Γ is formed with those training samples near class borders. Then we search a solution that minimises the training error with subject to the constraint max ρ_d^l. Intuitively, this leads in Euclidean nearest neighbour classifiers to place a series of OH's in regions where the training error is minimised. As the above examples show gaussian learn1NN exhibit this kind of solution. Let us see why it happens. In the case of a two-class problem the loss function that gaussian learn1NN minimises (equation 11) gives

$$L = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=1}^{2} y_{ji}\, d_j(\mathbf{x}_i) = \{d_1(\mathbf{x}) = 1 - d_2(\mathbf{x}) = d(\mathbf{x})\} = -\frac{1}{N}\sum_{i=0}^{N-1} z_i\, d(\mathbf{x}_i) + A \tag{29}$$

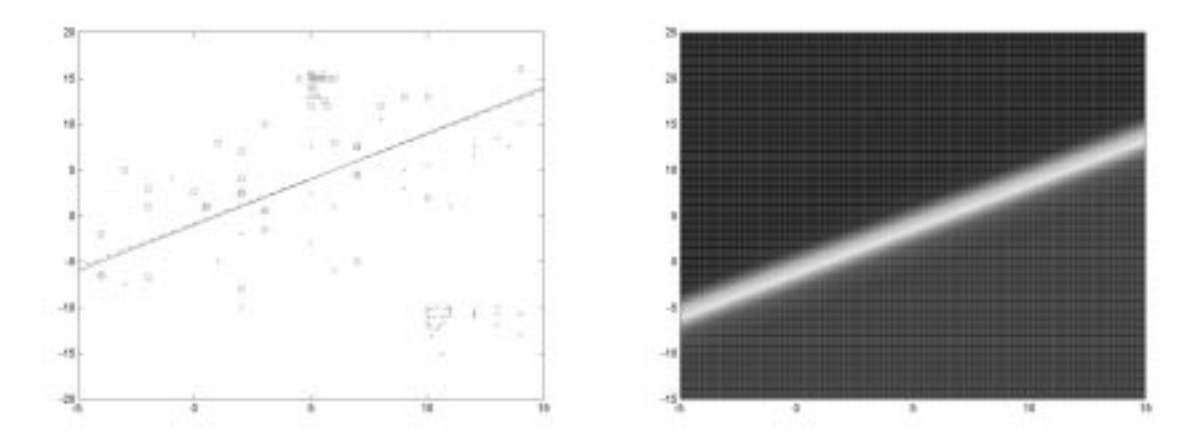
where A is a constant. Consequently, our algorithm maximises the margin over the whole training set. The absolute minimum of L (and the training error) is reached when $d(\mathbf{x}_i)\rightarrow 1$ when $z_i = 1$ and $d(\mathbf{x}_i)\rightarrow 0$ when $z_i = -1$ that is $d(\mathbf{x}_i)=1(z_i=1)$. As we see in the above examples the algorithm builds a $d(\mathbf{x})$ that effectively minimises the training error but also tends to behave in this way around the class boundary (that is, $d(\mathbf{x}_i)\rightarrow 1$ when $\mathbf{x}_i$ belongs to the decision region of

class 1 and $d(\mathbf{x}_i) \rightarrow 0$ otherwise). On the other hand, the equilibrium points (or the solution hyperplanes) of learn1NN (equation 23) only depend on the nearest training samples to class boundaries because only these pattern have a significant contribution of the term $d(\mathbf{x}_i)(1-d(\mathbf{x}_i))$ since $d(\mathbf{x}_i)$ is around 0.5 for these patterns. As we have seen experimentally, these equilibrium hyperplanes converge to the OH formed with the outlying patterns. This happens because the outlying points of both classes have the same activation level.

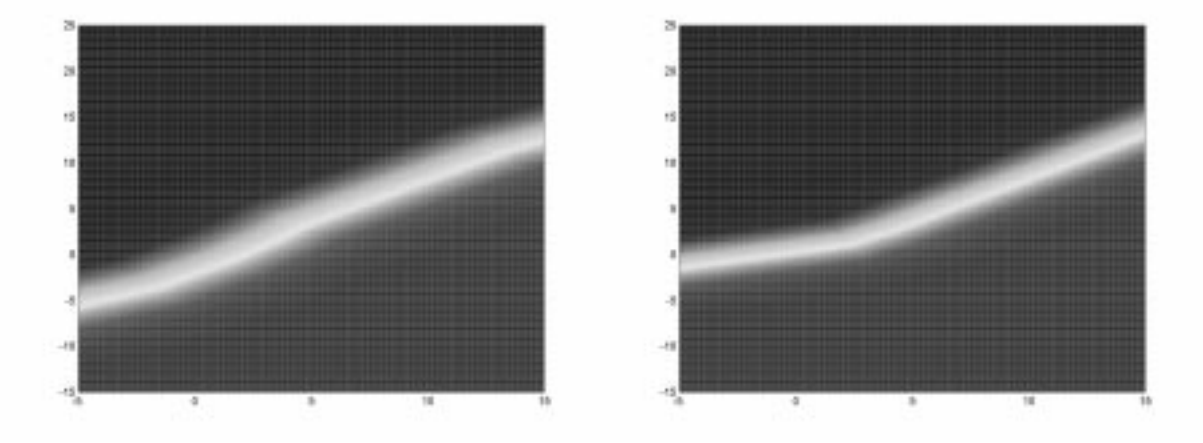


a) b) c) d)

Fig.8. a) A linear separable two-class problem with P(Error)=0. We show the training data (dots and squares), the support vectors (circles), the optimal margin hyperplane (continuous line) and the hyperplane computed with gaussian learn1NN (discontinuous line); b), c), d) the shape of decision surface for 2, 12 and 50 prototypes respectively.

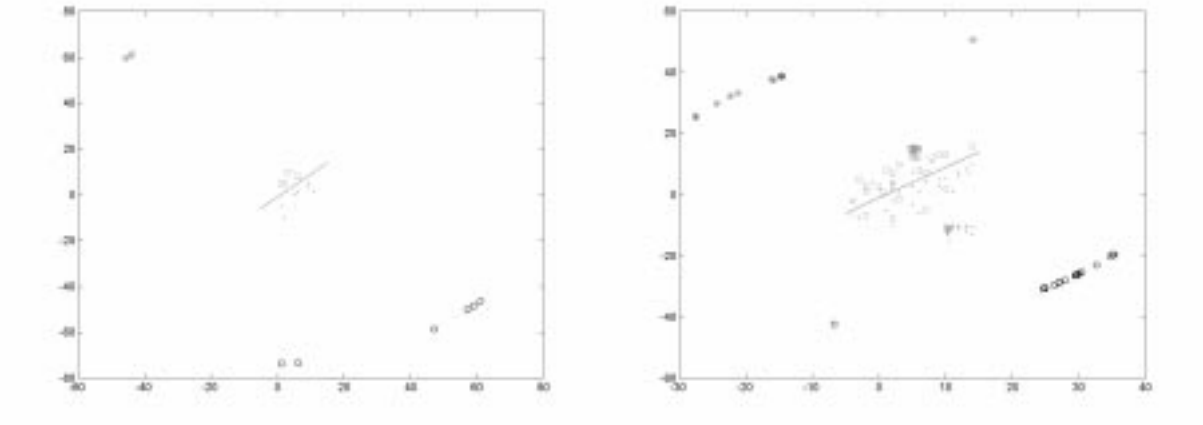


a) b)



c) d)

Fig.9. a) A linear separable two-class problem with $P(E_{min})>0$. We show the training data (dots and squares), those training points analogous to support vectors (circles), the optimal margin hyperplane using the above data (continuous line) and the hyperplane computed with gaussian learn1NN (discontinuous line); b), c), d) the shape of decision surface for 2, 12 and 50 prototypes respectively.



a) b)

Fig.10. The fifty prototypes computed with gaussian learn1NN for the problems of figs. 8 and 9. Training data are showed near the solution line while prototypes are those points far from it.

7. Model Selection

The set of prototypes **P** has been computed with the learn1NN algorithm to minimise the training classification error of the resulting 1-NN classifier. Consequently, we can bound the estimation error (i.e.. the error produced for using a finite training set) for the case C=2 (two-class problems) (Devroye et al., 1996)§19.3, §12.3. Moreover, we could use the structural risk minimisation (SRM) principle ((Vapnik, 1982), (Devroye et al., 1996)§18) to avoid overfitting and to select the best classifier from a pool of NN classifiers of increasing capacity. (For problems with C≥2, we could solve C two-class problems and applying the SRM principle in each sub-problem.) Using the SRM principle, first we form a finite number of sets of classifiers where the members of each set have the same size of **P** and train the whole pool of classifiers with e.g. *learn1NN* to minimise the training classification error, given a training set of size N. Then we choose a classifier $\tilde{\phi}_{N,j}$ from every set j $C_{NN}^{(j)}$ (formed with NN classifiers of size M_j where $M_j<M_{j+1}$) which achieves the minimum training classification error $\hat{L}_N(\phi)$over the whole set. Finally, we select the classifier ϕ_N^* from the sequence $\{\tilde{\phi}_{N,j},\ j\geq 1\}$ minimising the complexity penalised term estimate $\tilde{L}_N(\tilde{\phi}_{N,j})$ over j≥1 that is defined as

$$\tilde{L}_N(\tilde{\phi}_{N,j}) = \hat{L}_N(\tilde{\phi}_{N,j}) + r(j,N) \tag{30}$$

The complexity penalty r(j,N) is

$$r(j,N) = \sqrt{\frac{32}{N} V_{C_{NN}^{(j)}} \ln(eN)} \tag{31}$$

where $V_{C_{NN}^{(j)}}$ is the VC dimension of the NN classifiers with M_j prototypes. Since the N-th shatter coefficient of the class of 1-NN nearest classifiers C_{NN} ((Devroye et al., 1996)§19.3) is

$$S(C_{NN},N) = 2^M N^{(d+1)M(M-1)/2} \tag{32}$$

where M is the number of prototypes and d is the input dimension, the VC dimension of this class of classifiers can be computed solving (e.g. numerically)

$$S\left(C_{NN}, V_{C_{NN}}\right) = 2^{V_{C_{NN}}} \tag{33}$$

However, the complexity penalty introduced in the equation (31) gives a very large value for many real-databases (e.g. NIST databases). (See in figure 11 the term r(j,N) for a database of 10000 samples of dimension two.) Thus, the SRM principle is not useful in these cases. Then we must employ standard procedures like the early stopping technique (that is to stop at a local minimum of the validation error) to avoid overfitting and select the best classifier from the pool of computed classifiers minimising the validation error.

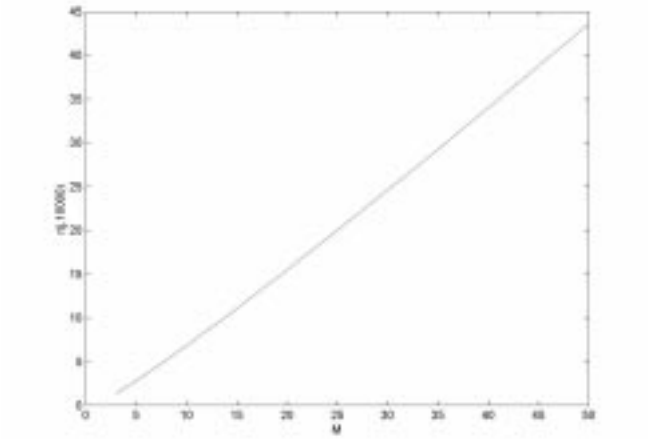


Fig.11. The penalised term r(j,N) of the SRM principle for N=10000 and d=2. Note that r(j,N) is useful only for a very small number of prototypes since it quickly increases to large values.

8. Discussion

8.1. 1-NN classifiers as maximum classifiers based on mixtures of the nearest models.

As we show, the design of a reduced set of prototypes **P** for a 1-nearest-neighbour classifier can be transformed into a problem of estimating the centres of C mixture models (one for each class). In the learning phase, we use a maximum classifier whose discriminant functions are determined with the nearest models of each class. Since the models of the mixture are local, they are placed around a centre point. Moreover, the contour levels of these models are radial since they employ the Euclidean distance to the centre for computing the activation of a pattern. Thus the resulting maximum classifier induces the same classification rule than Euclidean 1-NN classifiers and the centres of the mixture models can be considered

as the set of prototypes **P**. Then we could compute these centres with any typical method for mixture models (see (McLachlan & Basford, 1988)). However, the best solution for classification purposes is to design a set of prototypes that minimises the training classification error (Devroye et al., 1996). Accordingly, we have introduced a loss function that ensures this minimisation and have derived a learning equation based on online gradient descent. Moreover, training samples are classified wit a large margin so generalization performance could be improved. In the recall phase, the 1-nn classification rule is applied using the prototypes computed with the learning algorithm. However the maximum classifier could also be used here to produce fuzzy outputs for post-processing purposes. This is an additional feature of our approach.

8.2. Properties of Gaussian Learn1NN.

Our approach to the design of 1-NN's prototypes is general enough to allow deriving many different learning equations that use different kernel functions, distance metrics and optimization methods. In this chapter, we have introduced the most direct approach that employs radial gaussian kernels, the Euclidean metric and online gradient descent. As we have shown, the locality parameter of radial gaussian kernels (σ) controls how many local minimum of the cost function exist. If σ is large enough then the probability of reaching a deeper minimum of the loss function is augmented. The minimum points of the gaussian cost function L_{gauss} are based only on those training data near those class borders that minimise the training classification error. This is an important property. In classification, the essential thing is to get right class borders. If we know in advance training data near class boundary, we could use only this subset to build class borders (Breiman, 1998). Support vector learning (Schölkopf, 1997) is an important example that on this property since it only retains those boundary points (the support vectors) in class borders. In section 6 we have shown experimentally that gaussian learn1NN use the support vectors to compute a series of optimal margin hyperplanes. Besides, learn1NN seems biased towards smooth solutions since the algorithm ignores the excess of parameters and consequently overfitting is not present (at least in the reported experiments). Moreover, the excess of parameters can help to improve training and generalization error as it has also empirically showed in multi-layer perceptrons trained with the back-propagation algorithm (Lawrence, 1997). (Section 5.2 shows that in the Ripley's problem our learning algorithm achieves the best solution for 32 prototypes. However the resulting class border could be implemented with only six prototypes. Hence the

number of effective parameters is much smaller than 32.) A simple explanation of this behaviour was suggested by (Kröse & van der Smagt, 1996) §4.6: the addition of extra parameters can decrease the probability of being trapped in a local minimum while the effective number of parameters is the same. Accordingly, the excess of parameter only helps to achieve a deeper minimum while the number of effective parameters (that is the number of parameters that reflect the approximating power of the implemented solution) seems to be constant when a critical number of free parameters is exceeded.

8.3. Learn1NN and RBF's.

We have introduced 1-NN classifiers (that use the whole training set for the set of prototypes **P**) as a maximum classifier whose discriminant functions are a simplification of a particular kind of Normalised RBF's. Hence, 1-NN classification can be seen as a simplified kernel classification rule (Devroye et al., 1996)§10. Furthermore, we have extended our approach considering the case where the samples that form **P** are no longer constrained to training points and their size (M) are much smaller than the number of training data (N). This expansion consists of replacing the non-parametric estimation with a series of mixture models defined for each class. These models are parameterised with a local kernel $G(d(\mathbf{x},\mathbf{w});\gamma)$ where d is the distance metric, **w** is the centre and γ is a locality parameter. Since the 1-NN classification rule can be derived using the nearest mixture model of each class using the metric d, the centres of the mixture models are consequently the set **P**. Then we have derived an adaptive algorithm (learn1NN) to compute them for minimising the training classification accuracy. Hence, the resulting 1-NN classifier can be considered a particular normalised RBF's based on mixture models. However, our approach is very far from standard RBF's (Bishop, 1995)§5. E.g. the centres of standard RBF's and **P** are computed in a very different way. Compare equations (19) with the equations of updating the centres of RBF's using supervised learning (e.g. p.191(Bishop, 1995) and p.267 (Haykin, 1994)) or unsupervised learning (p.188-189 (Bishop, 1995)). In fact, these disagreements are based on the use on different architectures and cost functions for different purposes: while RBF's estimates posterior class probabilities using localised basis function near their centres, we use prototypes to estimate Bayes class borders according to the 1-NN classification rule. Another difference is the effect of sigma in gaussian RBF's and in 1-NN classifiers trained with gaussian learn1NN. In gaussian RBF's, the width parameter σ affects the quality of the approximation (Mel & Omohundro, 1991). Given a fixed number of centres, a very small value of σ causes under-smoothing. On the other hand, if σ is

very big the solution is over-smoothed. Consequently, the optimal value of sigma depends on the number of centres. In our case, σ only affects in the learning phase where it controls the degree of smoothness of the loss function (e.g. the number of local minimum points). However, σ does not affect the approximation capabilities of the classifier that only depends on M.

8.4. Learn1NN and Kohonen's LVQ algorithm.

L*earn1NN* is focussed on placing the prototypes to minimise the training classification error achieving a large margin on correct classifications. Hence, prototypes are only arranged to form class borders. On the other hand, Kohonen's LVQ algorithms compute a particular kind of cluster centroids useful for classification purposes (e.g. LVQ1(Bermejo, 1999)). However, LVQ's prototypes are placed over all the input space (not only near class borders). These differences are stressed in the learning equations. In the *gaussian learn1NN*, the nearest prototypes of each class (C prototypes) are updated in each step of the algorithm. On the other hand, LVQ1 only considers the nearest prototype of **P** and LVQ2 and LVQ3 consider the two nearest prototypes. If σ is finite, the *gaussian learn1NN* equations resemble a fuzzified version of the update equation of LVQ algorithms since the step size α is modulated by a factor that depends on the current value of the discriminant functions. The inclusion of this adaptive factor makes that the update of a learn1NN's prototype notably differs from the update of a LVQ's prototype since our prototypes are only updated if training patterns are near class borders. Consider now the case of $\sigma \to \infty$ in *gaussian learn1NN*. Then, the nearest prototype that agrees with the class label of the current training pattern come near it a constant factor $\alpha\left(1/C - 1/C^2\right)$ where C is the number of classes. The other nearest prototypes move away it a constant factor α/C^2. Suppose that C=2 for comparison purposes. Then gaussian learn1NN's updating factors are both $\alpha/4$. Consequently, the updating equations of learn1NN will coincide with LVQ2's when the winning prototypes of gaussian learn1NN and LVQ2 are the same. Clearly, this only will happen in some regions near class borders.

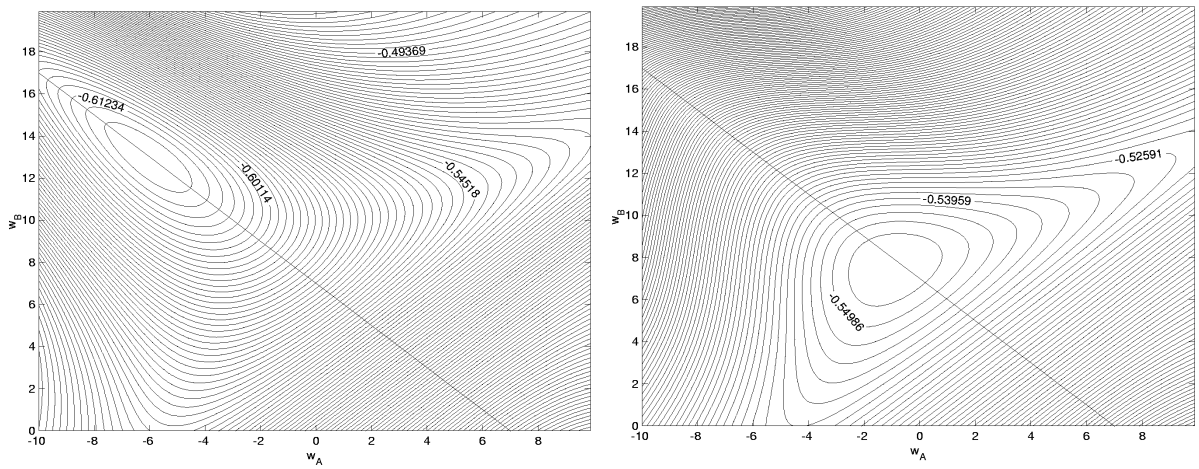
8.5. Regularization in learn1NN.

In the examples of section 4 we have seen that the equilibrium points of the loss function L tend to be located at infinitum subject to the constraint that the solution point belongs to some line. Hence, the iterative algorithm does not converge in finite time to the solution and the values of the prototypes grow indefinitely. However, the attraction basin of an

equilibrium point of learn1NN is, given a large value of sigma, a very abrupt valley located at the solution line. From a practical point of view, we can stop when the training error stops decreasing since the solution hyperplanes are then achieved. This, however, can require executing the learning algorithm a large number of steps. An alternative solution can consist in penalising solutions with a large value of the prototypes. In the examples of section 4 that use two prototypes, a simple regularizing term can be added to the loss function as follows

$$L_\lambda = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=1}^{C} y_{ji}\, d_j(\mathbf{x}_i) + \frac{\lambda}{2}\left\|\mathbf{w}_A - \mathbf{w}_B\right\|^2 \tag{34}$$

We can see in figure 12 for the example of section 4.3, how the inclusion of a term that measures the distance between the prototypes bounds the attractors of L_λ to a finite value while it preserves the desired solution F=3.5.



a) b)

Fig.11. Contour levels of L_λ for the example of section 4.3.with σ=25 and the line F=3.5: a) λ=0.0001 and b) λ=0.0005.

9. Conclusion

We have presented a learning strategy to design a set of prototypes for a 1-NN classifier based on minimising the training classification error. Our approach reformulates the design of 1-NN's prototypes as a problem of estimating the centres of a mixture model since the NN rule is

equivalent to a maximum classifier whose discriminant functions use the nearest models of a mixture for each class. Nevertheless, since the goal of the classifier must be the minimisation the training classification error, we do not compute these centres using standard method for mixtures. Instead, we introduce a loss function to compute these centres that ensure this minimisation. We have shown several properties of this cost function like robust behaviour in presence of outliers and its equivalence to the averaged margin function for two-class problems. One implementation of the learning algorithm based on the online gradient descent method and the use of radial gaussian kernels for the models of the mixture (*the gaussian learn1NN algorithm*) have been introduced. As section 4 and 6 displays, this learning algorithm places hyperplanes that are completely determined by the patterns closest to them. Furthermore, an equivalence between Vapnik's Optimal margin hyperplane algorithm (Vapnik, 1982) and ours have been established. For two-class problems, our algorithm uses the support vectors of the training set to form optimal margin hyperplanes. Finally, we notice in the reported experiments that the learning algorithm is biased toward smooth solutions. So gaussian learn1NN ignores the excess of prototypes if the solution can be solved with fewer prototypes. This prevents the resulting NN classifier to overfit training data.

References

Bengio, Y. (1991). Artificial Neural Networks and Their Application to Sequence Recognition, Ph.D. thesis, McGill University, Department of Computer Science.

Benveniste, A., Métivier, M., & Priouret, P. (1990). Adaptive Algorithms and Stochastic Approximations, Berlin: Springer-Verlag.

Bermejo, S. (2000). Finite-Sample Convergence Properties of the LVQ1 algorithm, the BLVQ1 algorithm and the GLVQ1 algorithm, in Bermejo, S. Learning with Nearest Neighbour Classifiers, Ph.D. Thesis, Barcelona: Universitat Politècnica de Catalunya.

Bishop, C. M. (1995). Neural Networks and Pattern Recognition, Oxford: Oxford University Press.

Bottou, L. (1998). Online Learning and Stochastic Approximations, in David Saal (Ed.) Online Learning and Neural Networks, Cambridge: Cambridge University Press.

Breiman, L. (1998). Half-&-Half Bagging and Hard Boundary Points, Technical Report No.534, Berkley: University of California, Department of Statistics.

Burges, C. J. C. (1999). A Tutorial on Support Vector Machines for Pattern Recognition, Machine Learning, ?, 1-43.

Cortes, C. (1995). Prediction of Generalization Ability in Learning Machines, Ph.D. Thesis, New York: University of Rochester, Department of Computer Science.

Darasay, B. V. (Ed.) (1991). Nearest Neighbor Pattern Classification Techniques, Los Alamitos, LA: IEEE Computer Society Press.

Dempster, A.P., Laird, N. M. & Rubin, D.B. (1977). Maximum Likelihood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society, series B, 39, 1-38.

Devroye, L., Györfi, L. & Lugosi, G. (1996). A Probabilistic Theory of Pattern Recognition, Berlin: Springer-Verlag

Duda, R. O. & Hart, P. E. (1973). Pattern Classification and Scene Analysis, New York: John Wiley Interscience.

Friedman, J. (1994). Flexible Metric Nearest Neighbor Classification, Technical Report, Stanford, CA: Stanford University, Department of Statistics and Stanford Linear Accelerator Center.

Garris, M. et al. (1997). NIST Form-Based Handprint Recognition System (Release 2.0), National Institute of Standards and Technology.

Hart, P.E. (1968). The Condensed Nearest Neighbor Rule, IEEE Transactions on Information Theory (Corresp.), 14, 515-516.

Haykin, S. (1994). Neural Networks: A Comprehensive Foundation, Englewood Cliffs, NJ: Macmillan College Publishing Company.

Kröse, B., & van der Smagt, P. (1996). An Introduction to Neural Networks, Eighth edition. The University of Amsterdam. Available electronically in http://www.fwi.uva.nl/research/neuro

Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J., & Torkkola, K., LVQ_PAK. The Learning Vector Quantization Program Package. Version 3.1, Helsinki: Helsinki University of Technology, Laboratory of Computer and Information Science.

Kohonen, T. (1996). Self-organizing Maps, 2nd Edition, Berlin: Springer-Verlag.

Lawrence, S., Giles, C. L. & Tsoi, A. C. (1997). Lessons in Neural Network Training: Overfitting May be Harder than Expected, Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97, 540-545, Menlo Park, CA: AAAI Press.

Mel, B.W., & Omohundro, S. M. (1991). How Receptive Field Parameters Affect Neural Learning, in Lippmann, R. P., Moody, J. E., & Touretzky, D. S. (Eds.) Advances in Neural Information Processing Systems 3, Boston, MA: Morgan Kaufmann Publishers.

Michie, D., Spiegelhalter, D. J., & Taylor, C.C. (Eds.). (1994). Machine Learning, Neural and Statistical Classification, London: Ellis Horwood: London, 1994.

McLachlan, G. J., & Basford, K. E. (1988). Mixture Models. Inference and Applications to Clustering, New York: Marcel Dekker.

Nilsson, N. J. (1990). The Mathematical Foundations of Learning Machines, Boston, MA: Morgan Kaufmann Publishers (previously published in 1965).

Ripley, D. (1994). Neural Networks and Methods for Classification, Journal of the Royal Statistical Society, Series B, 56, p. 409-456

Ripley, D. (1996). Pattern Recognition and Neural Networks, Cambridge: Cambridge University Press.

Schölkopf, B. (1997). Support Vector Learning. Ph.D. Thesis, Berlin: Informatik der Technischen Universität Berlin.

Smola, A., Barlett, P., Schölkopf B., & Shuurmans. (1999). Introduction to Large Margin Classifiers, in Smola, Barlett, P. Schölkopf, B. and Shuurmans, (Eds.) Advances in Large Margin Classifiers Boston, MA: MIT Press.

Scott, W. (1992). Multivariate Density Estimation: Theory, Practice and Visualization, New York: John Wiley & Sons.

Tarter, M. E., & Lock, M. D. (1993). Model-Free Curve Estimation, New York: Chapman & Hall.

Vapnik, V. (1982). Estimation of Dependencies based on Empirical Data, New York: Springer-Verlag.

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