2 Survey on Learning with Nearest Neighbour Classifiers

"The function that describes data well and belongs to a set of functions with low capacity will generalise well regardless of the dimensionality of the input space."

Vladimir Vapnik (p.240; Bishop, 1998)

"Mathematical theory is not critical to the development of machine learning.

But scientific inquiry is.

INQUIRY = sensible and intelligent efforts to understand what is going on. For example:

- ❑ Mathematical heuristics
- ❑ Simplified analogies
- ❑ Simulations
- ❑ Comparisons of methodologies
- ❑ Devising new tools
- ❑ Theorems where useful (rare!)
- ❑ Shunning panaceas

...

It makes research more interesting to know there is no one universally best method. What is best is data dependent. Sometimes "least glamorous" methods such as nearest neighbor are best. We need to learn more about what works best where. But emphasis on theory often distracts us from doing good engineering and living with data."

Leo Breiman (p.14-15; Wolpert, 1995)

Abstract- This chapter presents a brief introduction to the work developed in the thesis.

Index Terms- Soft K-Nearest-Neighbour Classifiers, Nearest-Neighbour Classifiers, Large Margin Classifiers, Oriented Principal Component Analysis, Local Stabilization, Ensemble Learning, Batch Learning Vector Quantization (LVQ) algorithms, Generalised LVQ1, Batch LVQ1, Dynamic LVQ algorithms, Kohonen's LVQ algorithms, Finite-sample Convergence, Online gradient descent, Hand-written Character Recognition..

1.Introduction

This chapter introduces several results of our work. Section 2 reviews the problem of learning pattern recognition. Section 3 examines several properties of nearest neighbour classifiers. Finally, section 4 gives a cohesive introduction to the toolbox of learning algorithms developed in the thesis, emphasising and unifying only those fundamental features of our work. The reader is encouraged to read the following chapters for having a complete and deeper view of our work.

2. Learning Pattern Recognition

Pattern recognition (PR) addresses the problem of synthesising artificial systems that group input data (also called patterns) into categories (or classes). Many PR problems are *ill defined* (i.e. there is no mathematical theory that can properly cope with them) so they cannot be solved completely by handcrafted algorithms and then the designer must use a **learning machine**. Typically this learning device automatically synthesises some of the parts that compose a pattern recogniser from a set of labelled examples belonging to the recognition problem (**learning from examples**).

Many modules can be integrated to form a pattern recogniser depending on the complexity of the problem. However, the core of any pattern recogniser is typically composed of a **feature extractor** and a **classifier**. The feature extractor reduces the input data by measuring certain invariant “features” or “properties”. The classifier uses then these features to make the decision of assigning the input pattern to a class.

The **feature extractor** is often handcrafted since it is rather specific to the problem. However, the current tendency is rely more on learning devices that *automatically extract features* and less on manual feature extraction of discriminatory information. Recent research efforts integrate *the feature extractor into the classifier* for performing a global training of both systems since separate training does not usually give the best possible solution. Most widely-used automatic feature extraction methods include statistical techniques like **principal component analysis** (PCA) (Jolliffe, 1986).

The **classifier** must assign input patterns to one of the pre-defined categories and consequently divides the input space into class regions. It is often general-purpose and trainable. An omnipotent classifier could solve any recognition problem without the help of the feature extractor. However, the use of a feature extractor is a mandatory in practical applications since it reduces the design complexity of the classifier. This augments the probability of having a better generalization performance since the learning device can estimate the parameters of the system with greater reliability.

There are many **classification methods** proposed in the literature. However, no method is universally superior to the others (that is, the best all problems). Only a method can be superior to the others for a particular problem or (in average) for a benchmark composed of several problems. This makes more interesting the problem of PR and focuses the research on designing learning machines with good generalization in a great variety of problems. Currently, the most distinguished classification methods (e.g. methods that are very useful for practical applications) are **non-parametric** (i.e. they do not assume any statistical distribution in data) like **neural networks**, **nearest neighbour classifiers**, **support vector machines** and **classification trees (CARTs)**.

In the next section, we will review two key topics in pattern recognition: learning and generalization.

2.1. Learning and Generalization

2.1.1. Goals

The learning process must synthesise a *reliable* pattern recogniser for the PR problem using a set of labelled examples (the training set). Reliable means that the learning machine must ensure a correct response to unseen examples during learning (**generalization**). We say that a system has a good generalization if it is 'near' the **Bayes classifier** (i.e. the classifier with the best classification accuracy for the problem at hand).

Since a pattern recogniser divides the input space into class regions, the designer must only focus on synthesising a system with similar class borders than Bayes classifier. Consequently, PR problems are much easier than regression problems and one can expect to build a learning machine with good generalization in practical applications where **finite resources** are used (finite amount of training samples and computation, and finite complexity of the recogniser).

The consideration of learning machines with '*good generalization*' implies, more precisely speaking, that these devices must fulfil some desirable features as:

1) The learning machine must convergence to the Bayes classifier when infinite resources are used (**consistency**)
2) The **convergence rate** to Bayes classifier must be fast
3) The learner must have **procedures of controlling the generalization ability** (**capacity control**)
4) There must be **practical ways of constructing learning algorithms** for these devices

(See for more details e.g. (Vapnik, 1995b)).

2.1.2. Learning as optimization.

Learning in these devices implies some kind of **parameter tuning**. The learning machine (or learner) induces from examples a statistical model that might reflect the computational structure of the problem. The model has a set of parameters that are estimated during learning.

The learning machine searches in a hypothesis space (i.e. the space that covers all the possible solutions depending on the values of the set of parameters) through time until it finds a feasible solution. The learner uses a **cost (or objective) function** to guide the search. In fact, it associates the desired solution with the global minimum of this function.

The learner typically uses an iterative algorithm (the **learning algorithm**) to minimise the cost function updating the parameters of the model through time. The learning system is consequently an **optimiser**, that is, a particular kind of dynamical system (a **trainable dynamical system**) with the goal of minimising a cost function.

In classification, the obvious cost function to minimise is the number of misclassifications in the training set. Nevertheless, this cost function is usually minimised implicitly since a direct use of the training classification error cause problems to most common optimization methods (e.g. gradient-based methods). Hence, alternative cost function like mean squared or cross-entropy errors are frequently used (see for more details (Bishop, 1995) §6).

2.1.3. Problems in Learning from examples

Learning from examples is not easy. The empirical model induced by the learning machine might serve to obtain general laws about the process which examples are taken. The model might also predict an indeterminate number of new phenomena from the problem (**generalization**). Besides it might show how the input variables of the problem are interrelated, that is, it might indicate the **computational structure of the problem** (e.g. a statistical structure).

Consequently, the learner must find (or select) a valid model of the problem from a pool of candidates belonging to the hypothesis space (**model selection**). However learning machines employ **finite resources**, that is, finite amounts of computation (or training time), finite number of examples and finite approximating power of the candidates (e.g. finite number of parameters in the model).

The effect of finite resources in the induced model limits the number of solutions that are reliable, that is, correct from the point of view of the learning process. In fact, we must conform with a reliable (or generalizable) empirical model instead of an optimal model of the problem. It is then fundamental to understand and quantify the limitations of the learning process due to the use of finite resources. We address this question in the following lines.

The approximation power of a learning system determines the hypothesis space where the learning algorithm searches a solution. As the hypothesis space augments, the learning system has more changes to solve the classification (or regression) problem since the **approximation error** (i.e. the difference between the Bayes classifier and the best classifier that belongs to the hypothesis space) will be typically smaller. However the model induced by the learning algorithm will have presumably worse generalisation capabilities since the solution can excessively tuned to training data (**over-fits data**) and then the **estimation error** will be (probably) bigger. The estimation error is that error produced by computing parameters with a finite training set. It is measured by the difference between the best model that belongs to the hypothesis space and the estimated model. A simple (and intuitive) explanation can justify the increase in the estimation error as the complexity of the model increases. If the complexity of the model is augmented, many more (statistical) parameters must be estimated using the same number of examples. It is known in the field of statistics than the ‘reliability’ of a statistic depends on the number of (random) samples used to compute it. Thus, the estimated parameters are less reliable as their number augments. Moreover, the computation of the parameters of a model of increased complexity (using a fixed number of training samples) can lead to problems of numerical stability since the learning system can be under-determined. E.g. if there is a number of unknown variables (parameters) of the same order than the number of equations (determined by the number of training samples) then the learning equations are not over-determined (a common condition in numerical methods in order to ensure stability).

While the approximation error is more problem-dependent and hence more difficult to handle, the learning machine can always reduce the estimation error to a negligible value reducing the hypothesis space. However a very reliable and simple empirical model can under-fit data so it is not a valid model of the problem (the **under-fitting** phenomena). In order to achieve a reliable and valid model of the problem, the learning system must balance both approximating and estimation errors or, in other words, solve efficiently a **trade-off** between its approximating power and the information about the problem given by the training set. This implies an **optimal capacity control of the learning machine**. (There are complementary ways of formulating this problem: bias-variance trade-off (Geman et. al, 1992), approximation error vs. estimation error (Niyogi & Girosi, 1994), and the relation between the capacity of the learning system and the number of training samples (Vapnik, 1995a). Besides, particular trade-offs for the classification problem have been formulated e.g. (Friedman, 1996a) (Tibshirani, 1996). However all these formulations are qualitatively similar.)

Finally, a third source of error comes from the optimization process that the learning algorithm performs (the **optimization error**). Most of the usual learning algorithms employ simple optimization techniques (like gradient descent) that hardly find a global minimum of the cost function. Besides, users often employ on-line versions of the algorithms where the learning equations are updated each time a pattern is presented. Although the use of on-line algorithms is supported by empirical evidence (e.g. they often find better solutions than batch versions), their convergence cannot be guaranteed theoretically and their exact equilibrium points are usually unknown. Hence, on-line algorithms can stuck at some undesirable points causing an unexpected source of error. Figure 1 shows the third sources of error in learning from examples.

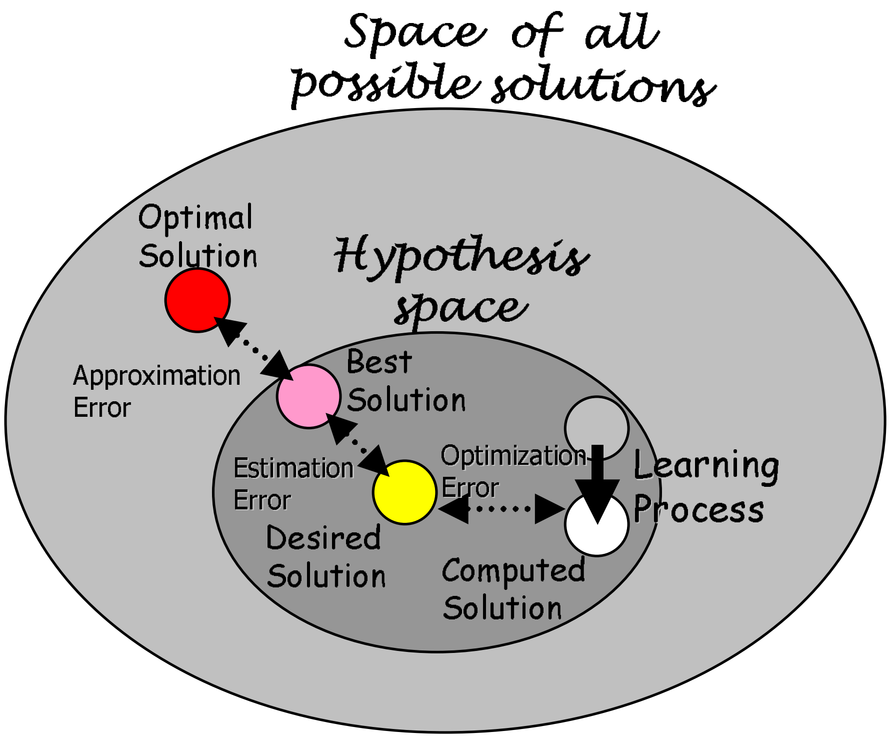


Fig.1. The third sources of errors in learning from examples.

2.1.4. Capacity control

The complexity (or **capacity**) of the learning machine determines how valid the induced model is (its generalization ability). Several measures of the capacity of learning machines have been proposed. The first and most notorious is the **VC dimension** (Vapnik, 1982). The VC-dim (h) is a combinatorial measure for two-class problems that counts the maximum number of training samples that the classifier can shatter. This measure is typically a monotonically increasing function that depends on the *number of parameters* of the learning machine and the *input data dimension*. If the learning machine minimises the number of misclassifications in the training set, the generalization error can be bounded using the VC-dim (see e.g. (Vapnik, 1982) or (Devroye, Györfi & Lugosi, 1996)). Then if n/h>20 (where n is the number of training samples), the estimation error of the learning machine for two-class classification problems is negligible (Vapnik, 1995b). For many learning machines the input space dimension affects the VC-dim (e.g. VC-dim augments as the input space). Consequently, the estimation error predicted by the VC theory agrees with the well-known phenomena in PR known as the **curse of dimensionality**: training points are sparser as the input dimension augments and thus we need much more data to build a reliable empirical model based on them. However learning machines often achieve good generalization with training sets that are much smaller than those predicted by the VC-dim bounds (e.g. neural networks). This happens due to the notion of the VC-dim of a learning machine is too general since it is independent of the learning algorithm, the target rule and the input distribution. Hence, the generalization bounds using VC-dim gives a **worst-case analysis**. Moreover, VC-dim works with the (unrealistic) assumption that the learning algorithm always finds a solution that (globally) minimises the training error. However, practical learning algorithms hardly find a global minimum since they limit their search to a restricted hypothesis space. Thus, the ‘real’ VC-dim of practical learning machines is much smaller than the theoretical VC-dim. Accordingly, we can obtain in practice much better results than those predicted by the VC theory. Besides, capacity of many learning machines (e.g. systems with many local minima that depend on the particular training set) is a dynamic measure that can vary during learning phase (Vapnik, Levin & LeCun, 1994).

Recent research attempts address the problem of deriving a capacity measure to reflect the pattern distribution in input space, the target function and constraints imposed on the classifier *during learning* in order to derive an **average-case analysis**. A first example is the **effective number of parameters** proposed by (Moody, 1992). A second example is the **effective VC dimension** (Vapnik, Levin & LeCun, 1994) (Corina, 1995) that measures the ‘real’ VC-dim adjusting several parameters of a theoretical model with experimental results of the learning machine. Then the effective VC-dim is used instead of the theoretical VC-dim in the equations for obtaining tighter bounds on the generalization error. However, the method is only valid for linear classifiers where the cost function of the learning device has only a single global minimum.

Current research is focussed on **scale-sensitive versions of the VC-dim** that are more tuned to the real behaviour of some learning machines (e.g. neural networks trained with the back-propagation algorithm) like the **fat-shattering dimension** or **covering numbers** (Barlett, 1998) (Smola, 1998). It has been observed than the use of some cost functions like mean squared error minimises the training error but also maximises the **average margin error** (i.e. the classifier assigns training data with high confidence to one of the classes). Moreover, it has been showed that the generalization error of these learning machines with a large margin distribution of input data depends not on their VC-dim, but on the fat-shattering dimension. This new capacity measure is consequently affected by the margin distribution of input data. E.g. the fat-shattering dimension of feed-forward networks also depends on the size of their weights but not on the size of the architecture and the generalization error is worse as the size of the weights augments (Barlett, 1998).

As we pointed out before, capacity of the learning machine must be controlled to balance approximation and estimation errors. From a theoretical point of view, *optimal capacity control is achieved* when

Both terms of the right-hand side of the following equation are simultaneously minimised:

$$P(Err_C) < \hat{F}_C(\mathbf{D}_N) + r(h, N, \varepsilon) \quad \text{with} \quad \text{Probability} \quad 1-\varepsilon \tag{1}$$

where $P(Err_C)$ is the probability of error of the classifier C, $\hat{F}_C(\mathbf{D}_N)$ is the empirical cost function minimised using the training set $\mathbf{D}_N$ in the learning phase (e.g. the average training error or the proportion of examples that are not correctly classified with a margin γ) and the $r(h,N,\varepsilon)$ is the complexity term that depends on a capacity measure (h), the number of training samples (N) and constant ε.

The method of Structural Risk Minimization (SRM) (Vapnik, 1982) proposes the use of a constructive technique for the simultaneous minimization of both right-hand terms in equation 1. SRM computes nested subsets of learning machines of increasing capacity that minimise the empirical cost function $\hat{F}_C(\mathbf{D}_N)$. Then SRM choose that learning machine whose sum of right-hand terms of equation 1 is the absolute minimum of the computed devices. However, the complexity term $r(h,N,\varepsilon)$ is often too loose so the method of SRM cannot be applied in practice.

An alternative way of model selection is choosing that device from a pool of learning machines (of increasing capacity) which has the best classification accuracy measured on a independent set (the validation set). Then, all the learning devices adjust their parameters using the training set and stop learning when they achieve a minimum of the classification error on the validation set. In spite of its simplicity, this technique is theoretically motivated (Devroye, Györfi & Lugosi, 1996) §22 and the theory predicts than the method is more effective as the validation set size augments since the validation error becomes closer to the generalization error. However, a practical problem arises. Real validation error curves always have several (local) minimum points and achieving the deeper local minimum cannot avoid completely over-training since the method can fail due to the use of a finite validation set. Thus we must decide at which local minimum to stop (Prechelt, 1998). The theoretical study of (Barlett, 1998) in neural networks with large margin suggests that an effective and practical method to ensure a good value of the right-hand side of equation 1 is the computation of classifiers with a small error of the cost function and small weights. Since the weights of a neural network are increased during learning (since large weights augment the slope of the transfer functions and hence the margin on the classifications), a very **early stopping** can help to avoid over-training (e.g. stopping at one of the very first local minimum points). Other heuristic techniques like **weight decay** (Krogh & Hertz, 1992) pursue the same goal and can also be strongly recommended.

Alternative practical forms of controlling capacity include reducing the input space dimension through feature extraction techniques or by introducing some a priori knowledge of the problem into the learning machine in form of some kind of invariance. (E.g. techniques like thinning (Lam, 1995) are often employed in automatic handwriting recognition to provide invariance to the thickness of characters.)

2.2. Bibliographical references.

An exhaustive and complete review of the main references in the field of PR is a daunting task. However we will cite some of the most popular books and several articles that point to significant references in the field. (Nilson, 1965) and (Duda & Hart, 1973) are two classic books that analyse the most relevant work on PR in the fifties and sixties. While Nilson (1965) mainly focus on linear classifiers (e.g. Perceptrons), (Duda & Hart, 1973) analyses other forms of classification and also feature extraction. (A very recent update of the latter book can also be recommended (Duda, Hart & Stork, 1996).) Both books are a very useful introduction to recent work. Other significant PR books from the eighties are (Fukunaga, 1980) and (Hand, 1981). Recent books for Pattern recognition with Neural Networks are (Cherkassky, Friedman & Wechsler, 1991) (Hertz, Krogh & Palmer, 1991) (Kung, 1993) (Haykin, 1994) and (Bishop, 1995). (Ripley, 1996) includes other pattern recognition methods like nearest-neighbour classifiers. A current machine learning perspective of PR is given in (Mitchell, 1995) that includes e.g. CARTs (Breiman et al., 1984). A Recent survey of the application of PR to real-world problems is (Michie et al., 1994). General statistical learning theory is introduced in (Vapnik, 1982), (Vapnik, 1995a) and (Vapnik, 1998). (Vapnik, 1998) can also served as an introduction to the emerging topic of support vector learning. Statistical learning theory on Pattern Recognition is addressed in (Devroye, Györfi & Lugosi, 1996). Early surveys on Pattern recognition (sixties) include e.g. (Misky, 1961). A very recent survey on Pattern recognition with 303 references is (Kulkarni, Lugosi & Venkatesh, 1998). (LeCun & Bengio, 1995) and (Ripley, 1994) gives some review of some recent work on Pattern recognition with neural networks. (Dietterich, 1997) reviews some of the current lines of research in PR.

3. Nearest Neighbour Classifiers

Nearest neighbour (NN) methods are still among the most simple and successful for many (real-world) pattern recognition problems. See e.g. the extensive study on practical problems of (Michie et al., 1994) where NN methods are very competitive in comparison with more sophisticated and modern algorithms. Recent work on memory-based systems (Stanfill & Waltz, 1986), lazy methods (Aha, 1997) and local regression (Fan & Gijbels, 1996) (Cleveland & Loader, 1995) (Hastie & Loader, 1993) have revived the interest in these techniques.

NN classifiers are local learning systems (Bottou & Vapnik, 1992) (Atkeson, Moore & Schaal, 1997) since they fit the training data only in a region around the location of an input pattern. 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 se of prototypes (or codebook) **P**={(**m**$_i$, **y**$_i$), i=1...M} where **m**$_\mathbf{i}$ is a prototype (or codevector) that belongs to one of the classes and **y**$_\mathbf{i}$ is a class indicator variable. (The nth coefficient of **y**$_\mathbf{i}$ is equal to 1 and the other coefficients are equal to 0 when **m**$_\mathbf{i}$ belongs to the nth class.)

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

3.1. Generalization properties

K-NN methods **control capacity** through regulating how local the solution is (**locality control**) (Bottou & Vapnik, 1992). A trade-off between capacity and locality must be performed to achieve good generalization. Hence, they must control the effective number of training samples available for training locally the system. K-NN techniques ensure, in some degree, good generalization since they always have enough data to compute the estimations and K effectively controls the locality since, as K augments, neighbours tends to be sparser.

K-NN estimate posterior class probabilities (see e.g. Ripley, 1996; §6.2). Consequently they converge to Bayes classifier as K, M$\rightarrow\infty$ at an appropriate rate (Devroye, Györfi & Lugosi, 1996)§5 for all distributions (**universal consistency**). For K finite and M$\rightarrow\infty$, their probability of misclassification tends to a limit close but larger than Bayes error $Perr_B$: $Perr_{K\text{-}NN} \leq (1+\sqrt{(2/K)})Perr_B$ and $Perr_{1\text{-}NN} \leq 2Perr_B$ (p.62; Devroye, Györfi & Lugosi, 1996).

3.2. Learning algorithms

K-NN classifiers have several design choices that might be adjusted automatically from data (preferably) like: 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(**x**,**y**) is the Euclidean distance d(**x**,**y**)=||**x-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. However 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 $\mathbf{D}_N=\{(\mathbf{x}_j, \mathbf{y}_j), j=0...N-1\}$ (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 classes of learning algorithms 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, Györfi & Lugosi, 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 (see section 4.1.2).

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, Györfi & Lugosi, 1996) §19.3.

However the most favourable strategy of designing prototypes is to minimise the empirical classification error produced on the training set $\mathbf{D}_N$ (p.311, Devroye, Györfi & Lugosi, 1996) since generalization error bounds based on VC theory can be applied.

Other recent approaches in machine learning, like large margin classifiers (Smola et al., 1999), advocate the use of alternative minimization cost functions that allow tighter bounds on the generalization error.

See (Darasay, 1991) (Devroye, Györfi & Lugosi, 1996) §19 and §26 for more information on condensed, edited and automatic K-NN rules.

4. Learning algorithms

This section studies novel ways of constructing practical learning algorithms for k-NN classifiers and some procedures for controlling the capacity of these learning machines to ensure good generalization.

We also present a global learning algorithm for a classifier and a feature extractor based on oriented principal components (OPCA). In the context of k-NN classifiers, OPCA derives in a problem of learning their weighted metric.

4.1. Learning algorithms based on supervised clustering

4.1.1. The very first basic algorithm: Voronoi Data-dependent Partitioning.

One of the simplest algorithms to compute a set of prototypes for 1-nearest-neighbour classifiers is based on the use of a (unsupervised) cluster algorithm as follows:

1. Compute k_N cluster centroids $\{\mathbf{m}_i, i=1...K_N\}$ using a cluster algorithm from a training set $\mathbf{D_N} =\{(\mathbf{x}_i, \mathbf{y}_i)\ i=0...N-1\}$ where $\mathbf{x}_i$ and $\mathbf{y}_i$ are defined as before.
2. Assign each centroid to a class according to a majority vote of labelled training data that falls in each cluster

This simple algorithm computes first a (nearest-neighbour-based) **partition** $\wp_N = \bigcup_{i=1}^{K_N} R_i$ of the input region (i.e. it divides the input space in K_N cells) using the estimated cluster centroids to induce $\wp_N$ in the following way:

$$R_i = \left\{ \mathbf{x} \middle| d_q(\mathbf{x}, \mathbf{m_i}) = \min_{j=1..K_N} d_q(\mathbf{x}, \mathbf{m_j}) \right\} \quad i = 1...K_N \tag{1}$$

where $d_q(\mathbf{x}, \mathbf{m_j}) = \left(\sum_{i=1}^{p} |x_i - m_{ji}|^q \right)^{1/q}$. Then it assigns class labels to the (unlabelled) prototypes based on a majority vote within the regions of the partition. Hence, the training set is used twice: the first time for creating the clusters and the second for assigning labels to clusters centroids. This kind of algorithms belongs to the so-called **data-dependent partitioning methods** (Devroye, Györfi & Lugosi, 1996) §21.

The most popular clustering algorithm for D_2 (the Euclidean distance metric) is the **K-means algorithm** (McQueen, 1967). K-means works in input regions of high probability (i.e. regions with high density of input patterns) and places prototypes to approximate discretely the empirical density of samples observed in the training set.

Suppose two gaussian classes A and B with the same priors and variance $\sigma^2 = 6.25$ centred at points 2 and 7 respectively. Figure 2 shows class densities and random samples taken from their respective class distributions. The Bayes border is located at 4.5 and the minimum probability of error (Bayes error) is 0.1573. If we apply the K-means over a set of random samples taken from these two classes, the algorithm will tend to make a discrete representation of the input density function (fig. 3). Figure 4 shows the application of the K-means with 20 prototypes and the label assignation algorithm for this problem. The training set was formed by 2000 samples (1000 samples/class). We used the batch version of K-means and stopped at a minimum of the training misclassification error. The combined algorithm achieves a probability of misclassification (0.1589) near Bayes' (0.1573). Observe that K-means places prototypes to represent the input class density. Figure 5 display the histogram estimate of the prototypes computed with K-means, which effectively coincides with the input density function.

K-means computes prototypes to minimise locally the average reconstruction error of approximating input samples with the nearest prototypes (in the Euclidean sense):

$$E_{VQ}\left(\mathbf{m}_1,\ldots,\mathbf{m}_{K_N};\mathbf{D}_N\right)=\frac{1}{N}\sum_{i=0}^{N-1}\min_{j=1..K_N}\left\|\mathbf{x}_i-\mathbf{m}_j\right\|^2 \tag{2}$$

In fact, K-means solves a problem of **vector quantization** (VQ) for Euclidean nearest-neighbour vector quantisers (Gray & Gerscho, 1992).

For two-class problems, the use of K-means (with a sub-optimal convergence to local minimum points) and the labelling schema ensure the *convergence of the derived 1-NN classifier to the Bayes classifier* with probability one as $N \rightarrow \infty$ when

$$K_N \to \infty \text{ and } \frac{K_N{}^2 \ln(N)}{N} \to 0 \quad \text{(p. 378-380; Devroye, Györfi \& Lugosi, 1996)} \tag{3}$$

where ln is the natural logarithm (base e).

For a finite N, however, this condition is useless. The choice of K dramatically affects the overall classification performance and consequently must be carefully determined. A pseudo-optimal value of K can be empirically estimated using a validation set. See (Devroye, Györfi & Lugosi, 1996) §22.4 for error bounds on this method.

There are two versions of the K-means algorithm: **the on-line and batch versions**. While the batch version pass through the whole training data before the update of prototypes, the on-line counterpart modifies prototype each time a training pattern is presented. Batch version is based on Newton optimization (Bottou & Bengio, 1995) and on-line version performs the pattern version of gradient descent over equation (2).

The goal of both learning systems is to reach a local minimum point of (2).These points can be computed solving $\nabla_{\mathbf{m}_i} E_{VQ}(\mathbf{m}_1,...,\mathbf{m}_K;\mathbf{D}_N)=0$ i=1,...,K:

$$\mathbf{m}_i = \frac{1}{N_i}\sum_{j=0}^{N-1} 1(\mathbf{x}_j \in R_i)\,\mathbf{x}_j \quad i=1,...,K \tag{4}$$

where N_i is the number of training samples that fall in Voronoi region R_i and **1**(u) is the indicator function that is 1 when u is true and 0 otherwise.

While **batch version** converges very fast due to Newton's effect, it is more sensitive to initial conditions since it can violently diverge when it is not near local minima. The update equation of this algorithm is:

$$\mathbf{m}_i[n+1] = \frac{1}{N_i[n]}\sum_{j=0}^{N-1} 1(\mathbf{x}_j \in R_i[n])\,\mathbf{x}_j \quad i=1,...,K, \quad n \ge 0 \tag{5}$$

Note that equation (5) achieves an equilibrium point when $\mathbf{m}_i[n+1]=\mathbf{m}_i[n]$ that coincides with the minimum points of (2) (equation (4)).

On the other hand, **on-line K-means** is less sensitive to initial conditions and is preferred to batch gradient descent since it can make use of redundancy of training sets. Besides the on-line version has a (presumably) noisier dynamics and consequently could reach a global minimum point with greater probability than the batch gradient version could do. (See (Bishop, 1995) §7.5 for a general discussion about the benefits of on-line versions.) The update equation of online K-means is:

$$\mathbf{m}_i[n+1] = \mathbf{m}_i[n] + \alpha[n]1(\mathbf{x}[n+1] \in R_i[n])(\mathbf{x}[n+1] - \mathbf{m}_i[n]) \quad i=1...K, \quad n \ge 0 \tag{6}$$

where $\alpha[n]$ is the step size function and **x**[n+1] is an input sample that belongs to the training set. The algorithm can randomly select **x**[n+1] from the training set (random sampling) or pass through the training set in a cyclic fashion (cyclic sampling). Equation (6) has a very simple geometric behaviour: the nearest prototype of the current input pattern is updated to come near it.

Asymptotic convergence (i.e. convergence when training time tends to infinite) of on-line K-means is guaranteed when the number of training samples $N \rightarrow \infty$ (e.g. p.228-232: Kosko, 1992).

However, these large-sample results provides little guide about the real convergence of on-line K-means when N is finite. Before we derive our analysis of the **real attractors** (i.e. the equilibrium points) of on-line K-means, we will introduce a simple clustering problem that illustrates the behaviour of on-line K-means.

Suppose that we have three training samples (2,6 and 10). We want to compute two prototypes (w_1,w_2) that minimise the average reconstruction error. It is easy to see that this VQ problem has two optimal solutions (4,10) and (2, 8). Figure 6 shows the contour levels of the cost function (equation 2) of our problem. The cost function is symmetrical around axis $w_1=w_2$ and $w_1+w_2=8$. Besides, it has four global minimum points located at (2,8), (4,10), (8,2) and (10,4). Figure 6 also shows several lines that divide the cost function in regions. In each of these regions, the assignation of training samples to the scalars of the vector quantiser is the same. This is an important property of the cost function and helps us to analyse later the real convergence of on-line K-means.

Any optimization algorithm based on gradient descent over equation 2 must (approximately) follow a trajectory perpendicular to the contour levels of figure. Figure 7 shows several trajectories of on-line K-means with $\alpha=0.001$ and cyclic sampling for different initial points. These trajectories have been computed for each epoch (i.e. when the algorithm have passed through the training data once). As one can observe, on-line K-means follows the gradient of the cost function so here it has the same behaviour than a batch version based on gradient descent. However, what happens if we increase the step size? Figure 8 shows on-line K-means trajectories for $\alpha=0.5$ and cyclic sampling. Now, the equilibrium points do not coincide with the minimum points of the cost function. However they remain near them. This result suggests that the attractors (softly) diverge from the minimum points of the cost function as the step size augments. Figure 9 shows another problem that arises from using the on-line version. If we use a different ordering of training samples in memory, the attractors of on-line K-means vary. Hence, the *attractors depend on the step size and the ordering of training samples in memory*.

Our analysis of the **real convergence** of on-line K-means starts with the **study of attractors** for constant step size and cyclic sampling. As we pointed out, the cost function is divided into regions where training samples are always assigned to the same prototypes. (In fact, the number of these regions is the number of the combinations of N samples assigned to K prototypes.) Near attraction basins, the algorithm is inside one of these regions so each prototype always 'sees' the same training samples. Hence the K coupled linear systems of equation 6 can be decomposed in K uncoupled linear equations. If $\alpha \in (0,2)$, each linear system is (BIBO) stable (e.g. impulse response is absolutely summable) and then on-line K-means converges to the following equilibrium point:

$$\mathbf{m}_i[\infty] = \frac{\alpha}{1-(1-\alpha)^{N_i}} \sum_{j=0}^{N_i-1} \mathbf{x}_i[N_i - j](1-\alpha)^j \quad i = 1,\ldots,K \tag{7}$$

where the sequence $\{\mathbf{x}_i[j],\ j=0\ldots N_i-1\}$ are the training samples assigned to $\mathbf{m}_i$ in the order in which they arrive (i.e. the order that are stored in memory) and N_i is the number of training samples assigned to $\mathbf{m}_i$. Besides, each linear system converges to the solution with an exponential rate of $(1-\alpha)^{N_i}$.

Equation (7) shows that attractors depend on the step size and the order in which training data are stored in memory. K-means only converges near equation (4) when $(N_i - 1)\alpha < 2$ since $(1-\alpha)^{N_i} \approx 1 - N_i\,\alpha$. Then, K-means also follows the gradient path of a modified cost function: equation 2 is affected by a term that depends on the order of training samples in memory.

Convergence of the algorithm from any starting point can be studied when we consider the conditions that ensure that on-line K-means was a line search method (Dennis & Schnabel, 1989)§4.2 that minimises a (globally defined) cost function using a gradient descent approach.

The idea is then to compare the behaviour of on-line K-means at the end of each epoch and a (batch) gradient version. If $(N_i - 1)\alpha < 2$, the algorithm effectively performs (some kind of) batch gradient descent inside regions of constant assignation of samples to prototypes. Then, we only might ensure that the algorithm makes a correct transition between these regions. This implies the fulfilment of several technical conditions (Dennis & Schnabel, 1989)§4.2. However, if the algorithm performs gradient descent on the regions, good transitions are observed in practice.

Our study also addresses general case study of convergence: the step size $\alpha[n]$ is variable and data sampling is undetermined (e.g. random). Convergence then is also possible when the step size is a non-increasing function. Here the asymptotic value of prototypes is much less clear than in the case of the constant step size. Anyway, attractors are a weighted averaging of training samples assigned to them that depends on the shape of the step size $\alpha[n]$ and the way of sampling the training data. However, we have observed empirically that random sampling causes a slower convergence and an unstable behavior of the algorithm near attraction basins so cyclic version seems then the right choice.

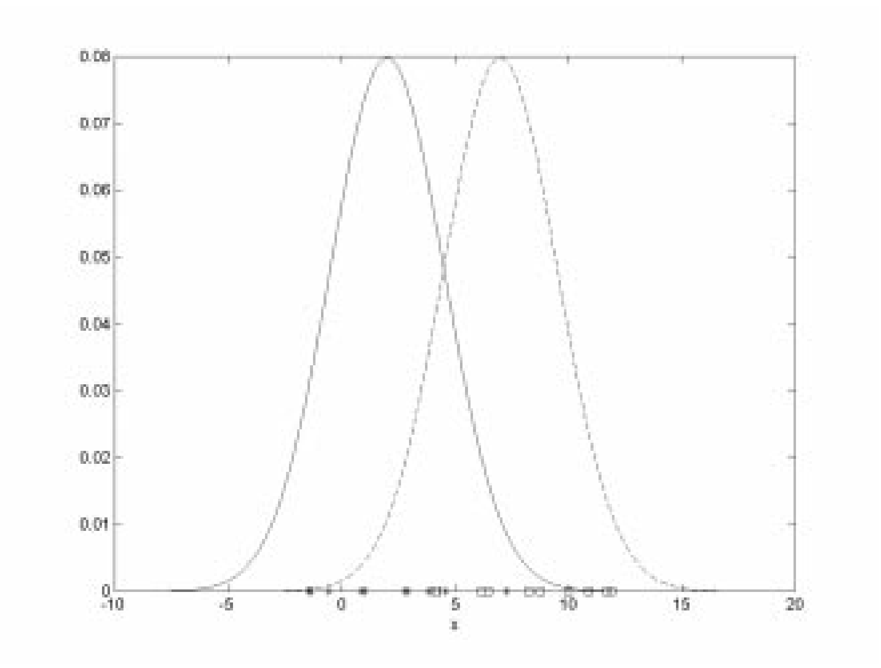


Fig.2. A two-class pattern recognition problem. We show the two normal class densities (centred at 2 and 7 with variance of 2.5^2) and 10 random samples of each class (squares and x's) taken from their density functions. Note that the optimal decision point is near 5 (4.5) where both class densities achieve the same value.

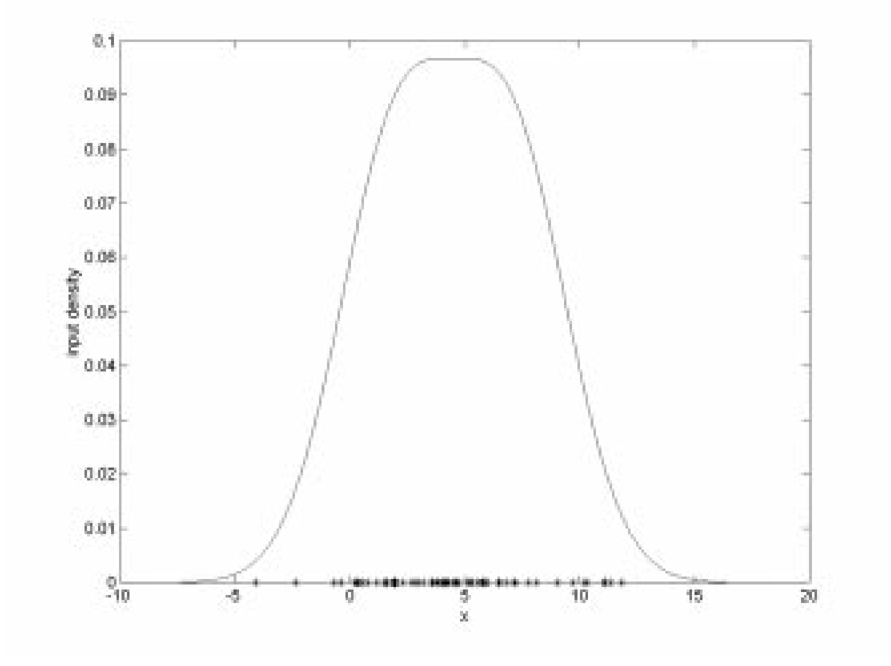


Fig.3. Input probability density function for the two-class pattern recognition problem of figure 2. We also show the random samples (or training points) taken from this problem presented to the K-means algorithm. Since the K-means does not take into account class labels, all points are represented with the same label. Notice that K-means 'sees' this distribution of samples during learning and consequently will place prototypes to approximate it.

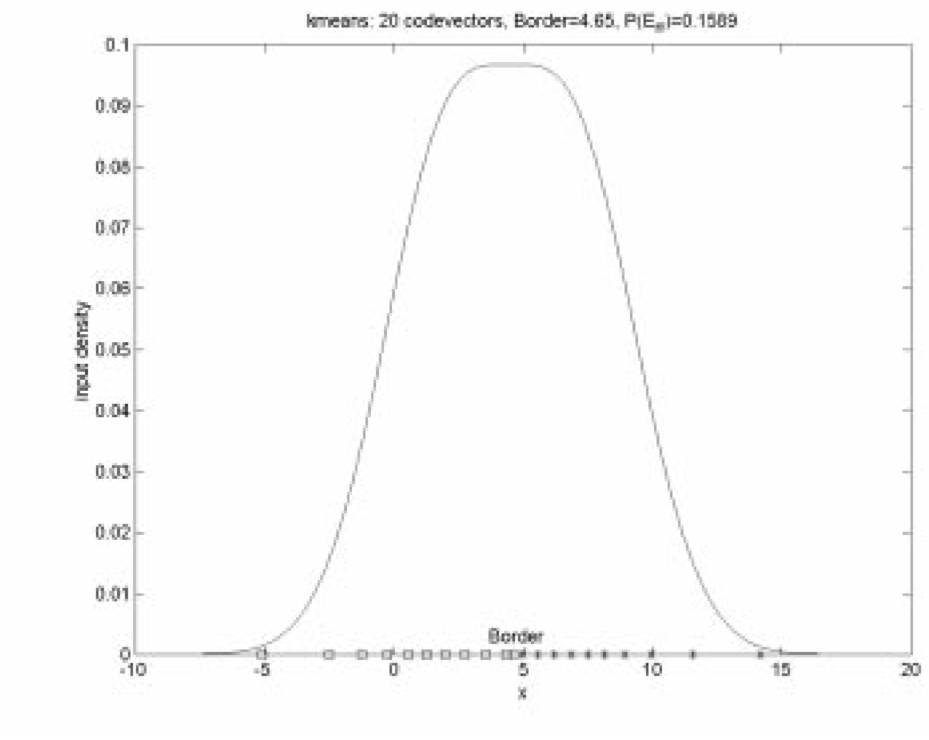


Fig.4. K-means (k=20) + labelling schema for the two-class gaussian problem. Border point (4.65) is marked with a circle while the labelled prototypes of the 1-nearest-neighbour classifier are denoted with squares and x’s. The combined algorithm achieves a probability of error of 0.1589 (very near Bayes error=0.1573). Note that K-means places prototypes to approximate input distribution (e.g. the density of prototypes is similar to the input density).

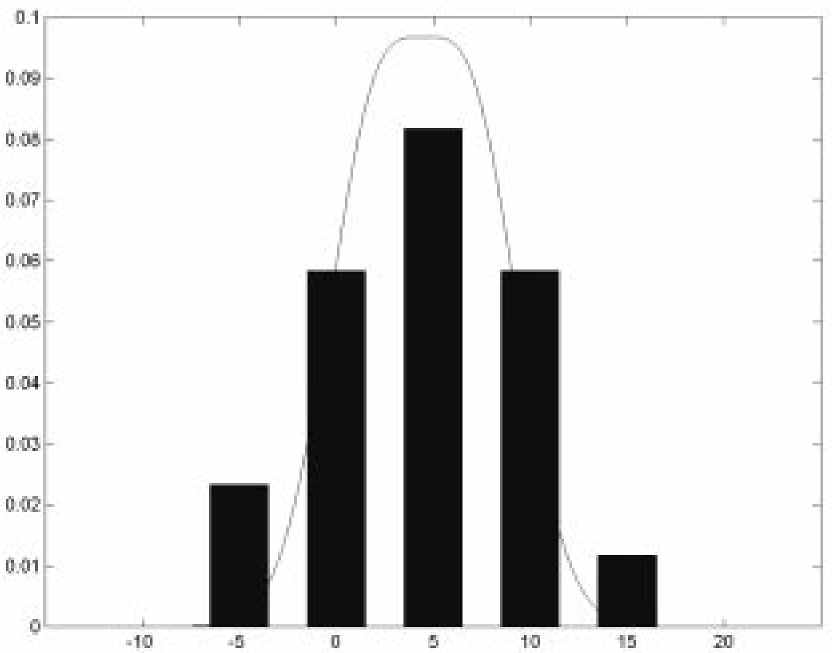


Fig.5. Input probability density and the histogram estimated from K-means’ prototypes.

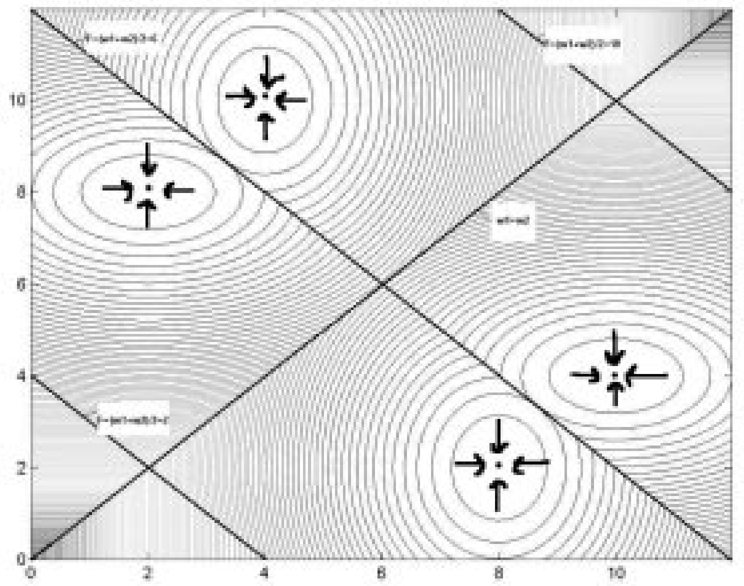


Fig.6. Contour levels of the cost function for the following 1-D VQ problem: represent points (2,6,10) with 2 scalars that minimises the average reconstruction error (equation 2). Thick lines denote the different parts that compose the cost function. Each of these parts forms a different assignation of training samples to the scalars of the vector quantiser. Arrows around minimum points denote the trajectories around minimum points of a gradient-descent algorithm that minimises the cost function. Consequently, K-means might be attracted (or converge) to them.

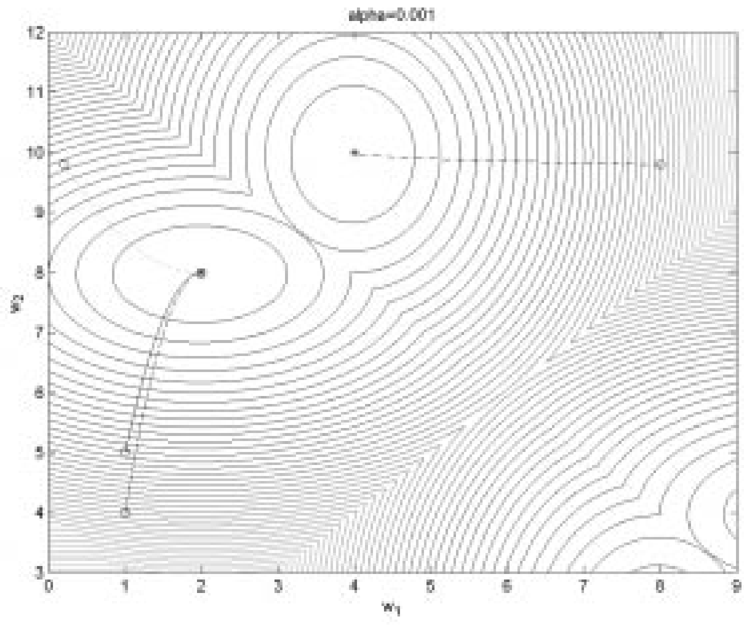


Fig.7. Trajectories several trajectories of on-line K-means with α=0.001, K=2 and *cyclic sampling* for different initial points. These trajectories have been computed for each epoch. Circles mark the initial departure of K-means and x's denote the equilibrium point of the algorithm.

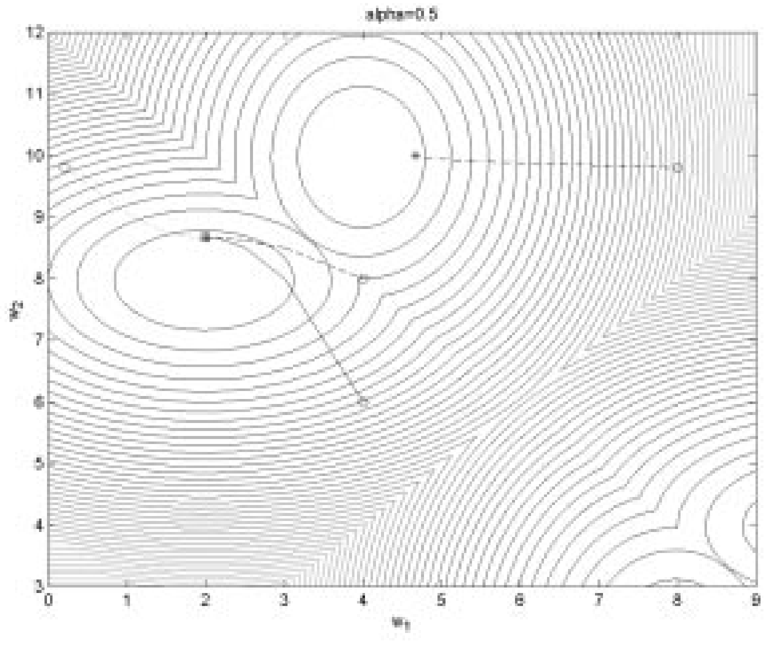


Fig.8. On-line K-means trajectories for α=0.5, K=2 and *cyclic sampling*. Note that the equilibrium points do not coincide with the minimum points of the cost function.

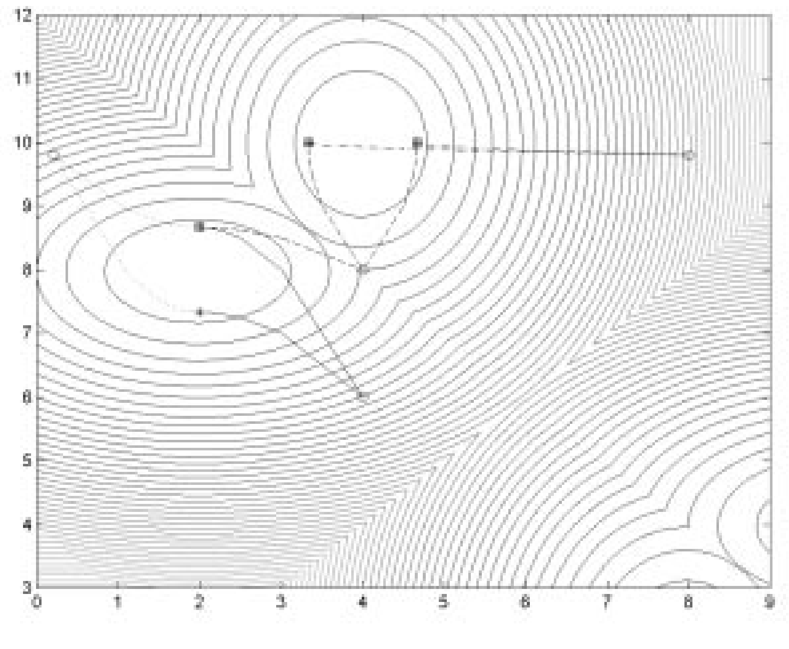


Fig.9. On-line K-means trajectories for α=0.5, K=2 and *cyclic sampling* for different ordering of training samples in memory. Observe that the equilibrium points of the algorithm depend on how the training samples are stored in memory.

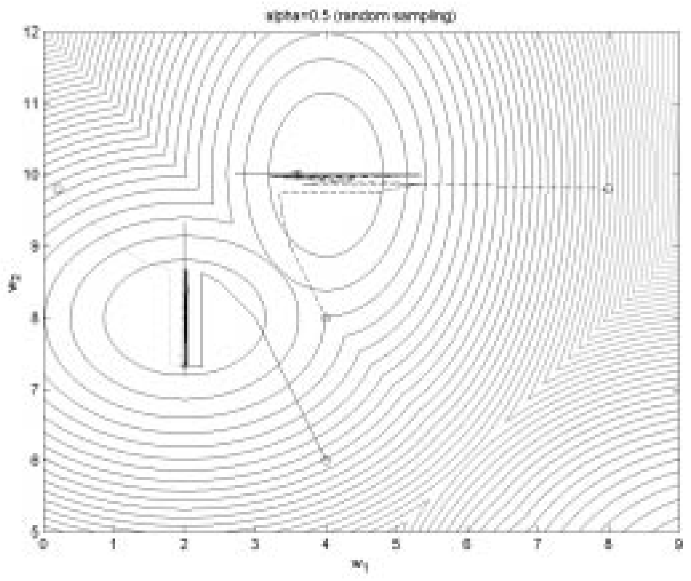


Fig.10. On-line K-means trajectories for α=0.5, K=2 and *random sampling*. Note that the random version begins to oscillate near the attractor points. Hence, the random version is less stable than cyclic version and has a slower convergence rate.

4.1.2. Supervised Clustering: The LVQ1 algorithm.

The above learning algorithms perform first an unsupervised clustering process and then assign labels to prototypes. A simple refinement of the above idea is to integrate the label assignation and the clustering process in a single step, that is, to perform **supervised clustering**. Supervised clustering will place labelled prototypes according to the distribution of classes and might ensure that estimated class borders agree with Bayes borders. Kohonen's LVQ1 (Kohonen, 1996) is a very simple algorithm based on supervised clustering that approximates Bayes borders for two-class classification problems in an elegant way.

LVQ1 is an on-line algorithm (i.e. prototypes are adapted each time an input pattern is presented) that has the following update equation:

$$\mathbf{m}_i[n+1] = \begin{cases} \mathbf{m}_i[n] + \alpha[n](\mathbf{x}[n+1] - \mathbf{m}_i[n]) & \text{if } \mathbf{x}[n+1] \in R_i[n] \text{ and } \mathbf{x}[n+1], \mathbf{m}_i \in \text{same class} \\ \mathbf{m}_i[n] - \alpha[n](\mathbf{x}[n+1] - \mathbf{m}_i[n]) & \text{if } \mathbf{x}[n+1] \in R_i[n] \text{ and } \mathbf{x}[n+1], \mathbf{m}_i \notin \text{same class} \end{cases} \quad i = 1,\dots,K \tag{8}$$

where α[n] is the step size function that belongs to the interval (0,1). The geometric interpretation of equation 8 can clarify the utility of LVQ1 for classification. Prototypes are moved away from samples of other classes but they come close to samples belonging to the same class. Note that

LVQ1 is a modified version of on-line K-means where now class labels affect the way that the clustering process is performed.

In fact, LVQ1 is an on-line gradient descent algorithm since it applies to compute the set of prototypes **P** the *pattern-based version* of gradient descent over the following cost function:

$$
\begin{aligned}
E_{LVQ1}(\mathbf{P}) = \frac{1}{2}\sum_{i=0}^{N-1}\sum_{j=1}^{K} 1(\mathbf{x}_i \in R_j) 1(cl(\mathbf{x}_i) = cl(\mathbf{m}_j)) \left\|\mathbf{x}_i - \mathbf{m}_j\right\|^2 \\
-\frac{1}{2}\sum_{i=0}^{N-1}\sum_{j=1}^{K} 1(\mathbf{x}_i \in R_j)\, 1(cl(\mathbf{x}_i) \neq cl(\mathbf{m}_j)) \left\|\mathbf{x}_i - \mathbf{m}_j\right\|^2
\end{aligned} \qquad (9)
$$

where the cl(**x**) function returns the class label of **x** and **1**(condition) is the indicator function which is 1 if condition is true and 0 otherwise. The minimization of equation 9 ensures a training error of the resulting nearest neighbour classifier smaller than 50%. This is possible since the minimum points of E_{LVQ1} fulfil that $2N_{mi} > N_i$ for all i=1,...,K where N_i is the number of training that fall in Voronoi region R_i and N_{mi} are those training samples that fall in R_i belonging to the same class that $\mathbf{m}_i$ and consequently $\sum_{i=1}^{K} 2N_{mi} > \sum_{i=1}^{K} N_i = N$. In spite of the minimum points of E_{LVQ1} do not guarantee than training error is minimised, LVQ1 achieves in practice very good classification results (much better than a training error of 50%). The study of the asymptotic convergence of LVQ1 for the infinite sample case provides *additional insights* about how works LVQ1 so well.

Study of the large-sample convergence of LVQ1 can be addressed using tools of the stochastic approximation theory (e.g. (Benveniste et. al, 1990)). The idea is to study the convergence of an ODE (ordinary differential equation) that has the same asymptotic convergence than equation (8). Then we apply the condition of stochastic equilibrium over the ODE, which gives:

$$
\begin{aligned}
E_X\left[(\mathbf{x}-\mathbf{m}_i) 1(\mathbf{x}\in R_j)(1(cl(\mathbf{x}) = cl(\mathbf{m}_i)) - 1(cl(\mathbf{x}) \neq cl(\mathbf{m}_i)))\right] = \\
\int_{R_i} (\mathbf{x}-\mathbf{m}_i)\left(p(\mathbf{x}|C_{cl(\mathbf{m}_i)}) P(C_{cl(\mathbf{m}_i)}) - \sum_{\substack{j=1 \\ j\neq cl(\mathbf{m}_i)}}^{C} p(\mathbf{x}|C_j) P(C_j) \right) = 0 \quad i = 1,\ldots, K
\end{aligned} \qquad (10)
$$

where C is the number of classes, $\{P(C_j)\}$ are the (prior) class probabilities, $\{p(\mathbf{x}|C_j)\}$ are the class density functions. (We have omitted the technical details that the learning system must fulfil in order to derive equation 10. The reader can find them in e.g. (LaVigna, 1990).)

Equation 10 is the stochastic equilibrium condition of a vector quantizer (see e.g. p.228-232: Kosko, 1992) that places prototypes according to the following density function:

$$p(\mathbf{y}) = p\left(\mathbf{x}\middle|C_{cl(\mathbf{m}_i)}\right)P\left(C_{cl(\mathbf{m}_i)}\right) - \sum_{\substack{j=1 \\ j\neq cl(\mathbf{m}_i)}}^{C} p\left(\mathbf{x}\middle|C_j\right)P\left(C_j\right) \quad \forall\mathbf{x}\in R_i \quad i = 1,\ldots,K \tag{11}$$

For a two-class problem and K→∞, equation (11) is a probability density function that is zero at Bayes borders. Hence, LVQ1 places prototypes around Bayes borders and consequently the resulting nearest-neighbour classifier estimates Bayes classifier. Figure 11 shows LVQ1's input density and the prototypes computed with LVQ1 for the problem of figure 2. Observe that the density function is zero at bayes point (4.5) and prototypes are placed according the LVQ1's density and hence an estimation of the Bayes classifier is finally performed by the nearest-neighbour classifier.

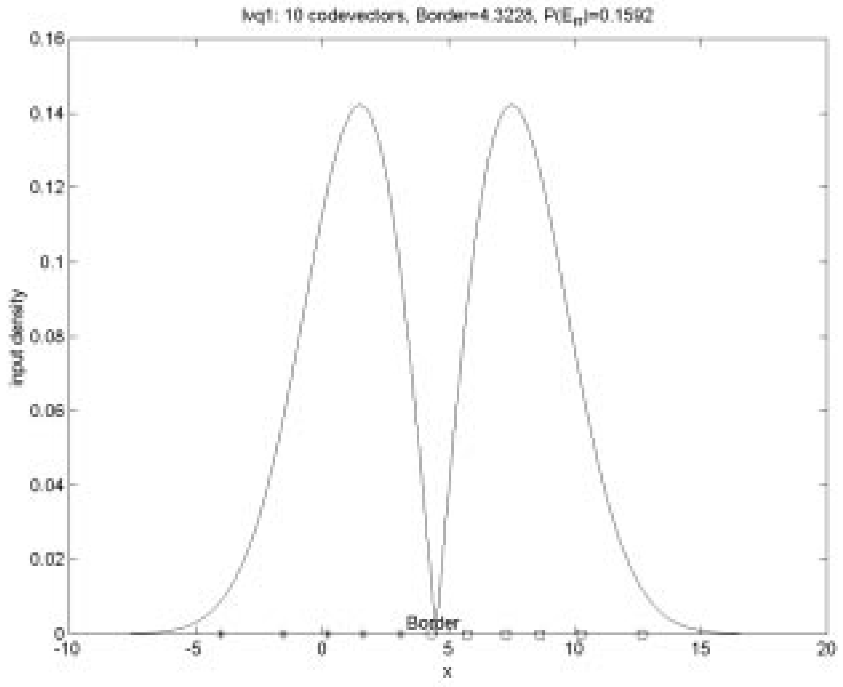


Fig.11. LVQ1' s input density function (equation 11) for the problem of figure 2 and the LVQ1's prototypes computed with a training set. These prototypes are denoted with squares and x's. The border point of the resulting nearest-neighbour classifier (4.32) is marked with a circle. Note that prototypes are placed according to the displayed density function.

One problem of LVQ1 is that can easily over-fit training data. Figure 12 shows the nearest-neighbour border (dashed line) using 32 prototypes computed with LVQ1 and Bayes border in a synthetic problem (Ripley, 1994). Clearly, the estimated border is more complex than Bayes border since prototypes are over-tuned to training samples.

However, LVQ1 can give better performance than nearest neighbour classifiers formed with the whole training set. Besides, it asymptotically achieves a similar error rate of these classifiers as its number of prototypes tends to the training set size (N). We can understand why this phenomenon is produced since in the limit case (when the number of prototypes is N), LVQ1's prototypes converge to the training points. Figure 13 shows the average error rate (over 100 runs) estimated with a test set of size 1000 of nearest-neighbour classifiers trained with LVQ1 for different sizes of the set of prototypes. (The training set is formed by 250 samples and we stop training when the training error rate achieves a minimum.) As we can see, the error rate achieves a minimum point for 4 prototypes and then the LVQ1-based classifier asymptotically converges to the error rate of nearest-neighbour classifier that use the whole training set. Figure 14 displays 1-NN's border with the whole set of prototypes. Compare figures 14 and 12 and observe that LVQ1's solution resembles 1-NN's in some regions: both classifiers converge as the number of prototypes augments.

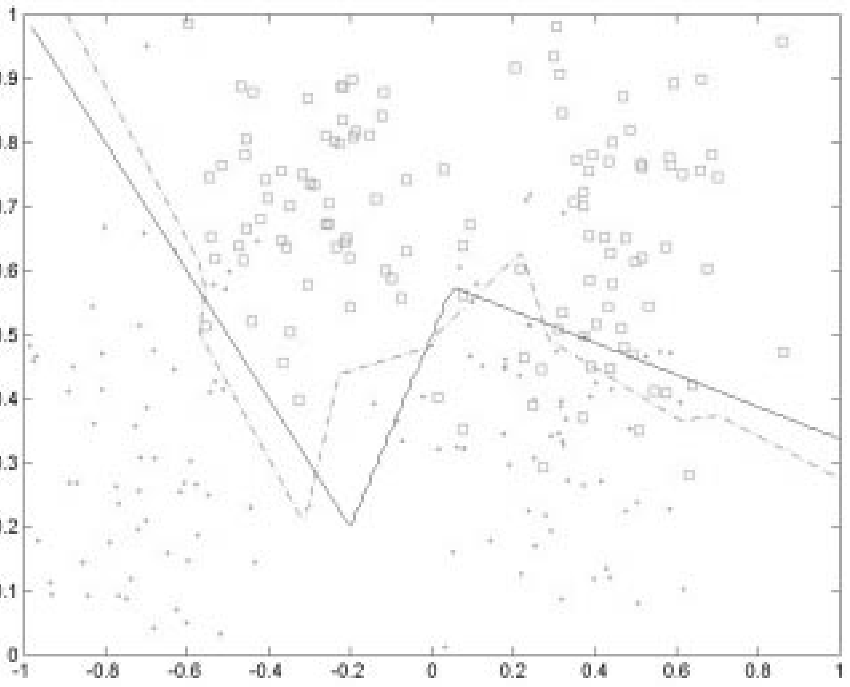


Fig.12. Ripley's synthetic training set (250 samples) with the Bayes border (solid line) and the class borders computed with LVQ1 for 32 prototypes (dashed line). The test error for this classifier was 9.4 %. Note that the classifier tends to over-fit training data.

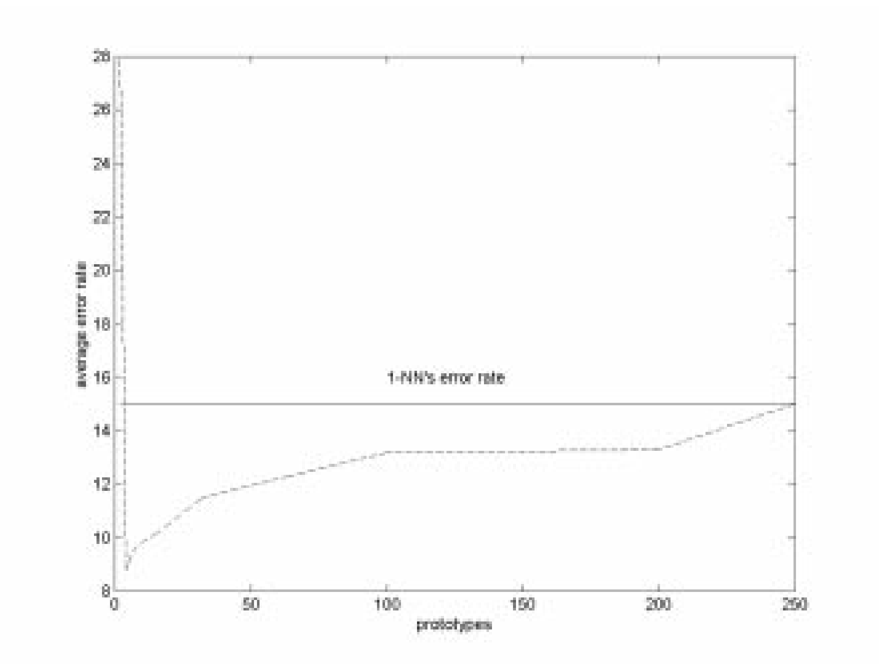


Fig.13. Error rate estimated with a test set of size 1000 for Ripley's problem. Dashed line shows average error rate over 100 runs of LVQ1 for a different number of prototypes. Solid line shows the error rate of the 1-nearest-neighbour classifier formed with the whole set of prototypes. While LVQ1 achieves a maximum relative improvement of 70% over 1-NN, it converges asymptotically to 1-NN's error rate as the number of prototypes increases.

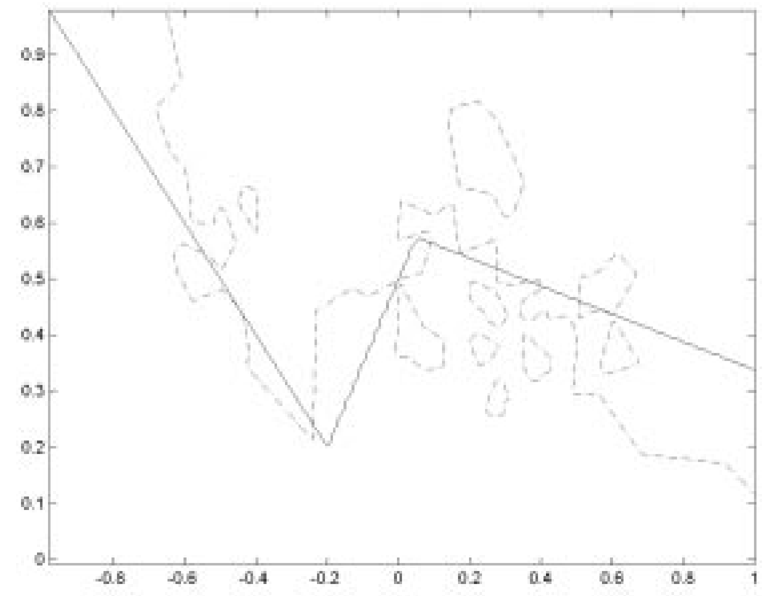


Fig.14. Ripley's Bayes border (solid line) and the 1-nearest-neighbour border induced with the whole training set (250 samples) as the set of prototypes. The test error for this classifier was 15.0 %. Note that some regions of the 1-NN's border resemble LVQ1's solution with 32 prototypes.

Another problem arises. If LVQ1 is an on-line algorithm, does it minimise practically E_{LVQ1}? As we just seen above on-line K-means converges to the right equilibrium points under certain constraints. Dynamics of LVQ1 and the cost function E_{LVQ1} has many similarities with on-line K-means and the average reconstruction error (e.g. search space is divided in regions where training points are assigned to the same prototypes). Consequently our analysis of real attractors of LVQ1 and the conditions for ensuring convergence follows a similar reasoning. We avoid repeating again a similar derivation than the above section and we will present some results.

First, we have found the necessary conditions for ensuring that LVQ1 will be globally convergent for constant step sizes (α) and cyclic sampling. Global convergence (i.e. convergence to a minimum of the cost function from any starting point) is ensured when LVQ1 is a line search algorithm (see (Dennis & Schnabel, 1989)§4.2). This is achieved when

$$\alpha < \min_{i=1,\dots,K} U^{conv}(i) / \kappa \text{ for all the epochs} \tag{12}$$

where $U^{conv}(i) = 2 / |S_i - N_i / S_i|$ when $S_i^2 \neq N_i$, $S_i = 2N_{mi} - N_i$, N_i is the number of training that fall in Voronoi region R_i and N_{mi} are those training samples that fall in R_i belonging to the same class that $\mathbf{m}_i$ and κ is a constant (e.g. 10 or 100). Furthermore, LVQ1 must follow the gradient path. This second condition can be fulfilled when

$$\alpha < \min_{i=1,\dots,K} 2/S_i \text{ for all the epochs} \tag{13}$$

(This is in fact the typical condition of gradient algorithms that limits the step size with the largest eigenvalue of the Hessian matrix of the cost function; see (Bishop, 1995) §7.5.1 for further details.) These two simple bounds on the step size can help to ensure a good global convergence of LVQ1. However in practice LVQ1 can be locally convergent (that is it can converge near an attraction basin) for bigger values of the step size. Figure 15 shows convergence of LVQ1 with 2 prototypes near an attraction basins for the problem of figure 2. Only a very small alpha ensures that LVQ1 follows the gradient path. However the algorithm also converges (and very fast) for the other values of the step size.

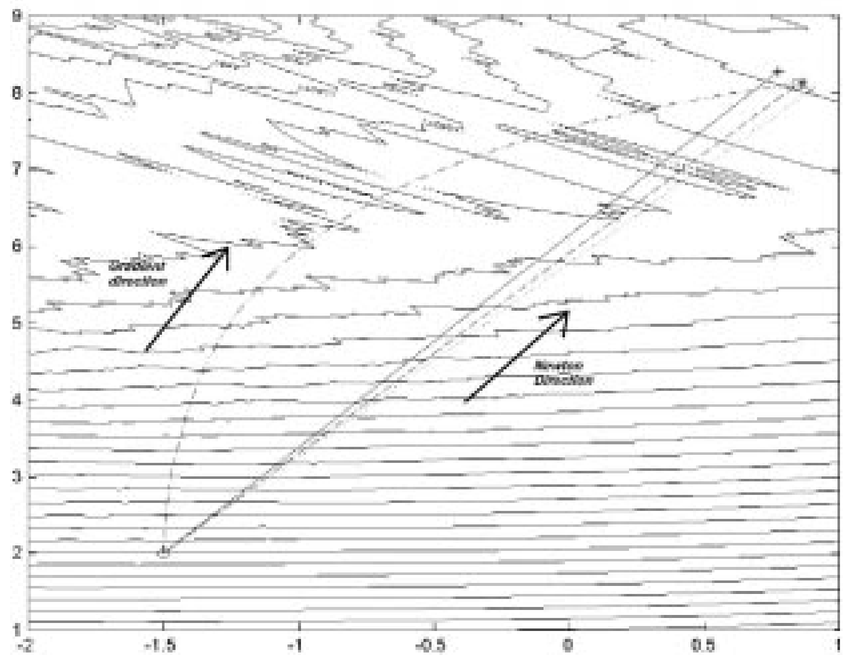


Fig.15. Dynamics of LVQ1 with 2 prototypes, constant step size and cyclic sampling for the problem of figure 2. The initial point is near an attraction and is marked with a circle while the equilibrium points for α=0.5,0.01,0.001 and 0.00001 are marked with X,+, a square and * respectively. Note that LVQ1 only follows the gradient path for α=0.00001. The other values of alpha are much greater than bounds that ensure global convergence. However, LVQ1 also converges for these values and follows a trajectory than resembles a method based on Newton's optimization.

Furthermore, attractors depend on the order in which training data are stored in memory and the value of the step size (again). The optimization error (i.e. the differences between the minimum points of E_{LVQ1} and the equilibrium points of LVQ1) of LVQ1 will be small when

$$\alpha < \min_{i=1,\dots,K} 2/\kappa N_{mi} \text{ for all the epochs} \tag{14}$$

Then the equilibrium points of LVQ1 are

$$\mathbf{m}_i[\infty] = \frac{1}{2N_{mi} - N_i}\sum_{j=1}^{N_i} \left(1(\mathrm{cl}(\mathbf{x}_i[j]) = \mathrm{cl}(\mathbf{m}_i)) - 1(\mathrm{cl}(\mathbf{x}_i[j]) \neq \mathrm{cl}(\mathbf{m}_i))\right)\left(1 - \alpha(N_{mi}[j] - N_i + j)\right)\mathbf{x}_i[j] \quad i = 1,\dots,K \tag{15}$$

where N_{mi} is the number of training points that fall in R_i belonging to the same class than $\mathbf{m}_i$ and $N_{mi}[j]$ is the number of the samples in $\{\mathbf{x}_i[u]\}$ (the sequence of training data assigned to $\mathbf{m}_i$) after $\mathbf{x}_i[j]$

that belong to the same class than **m**$_i$. The attractor of LVQ1 (equation 15) is near a minimum point of E_{LVQ1}. Thus, LVQ1 has an optimization error that depends on the position of data in memory (see figure 16). This error only will be arbitrary small when alpha is decreased to zero. However, we find in practice that it is small enough for κ=10.

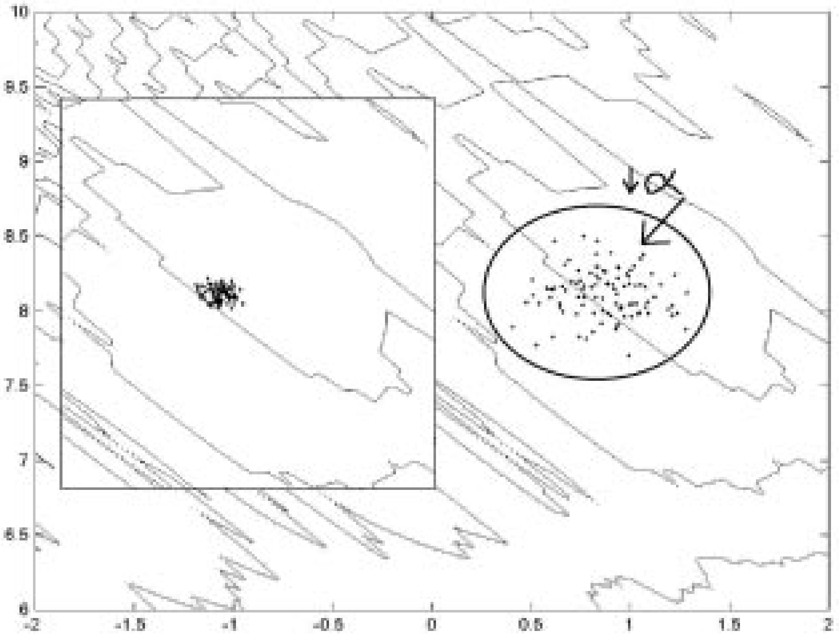


Fig.16. Distribution of attractors of LVQ1 for problem of figure 2 when alpha=0.01 and 0.001 (detail) for different ordering of training data in memory. Note that the dispersion of attractors (and the hence the dependence on the order) decreases as alpha does.

Finally, we notice that random sampling make worse the dependence of attractors on the order of supplied data and also the dynamics. Again, random version violently oscillates near a minimum point (see figure 17). Besides the algorithm can converge to a maximum point when prototypes are initialised near it.

4.1.3. Improving the LVQ1 algorithm: the generalised LVQ1.

LVQ1 performs a clustering process over a probability density function that is zero at Bayes borders for a two-class problem. Consequently, the resulting nearest-neighbour classifier estimates Bayes classifier. However, what happens for problems with a greater number of classes? Inspecting equation 11 gives insights about what is going on then.

If the density (equation 11) has a negative value in some regions of the input space, LVQ1 does not achieve a good solution. This happens when, in some regions, the density of points of the majority class is smaller than the sum of the other densities. In the prototype's space (i.e. the space where the learner searches a solution), these regions cause a maximum of the cost function E_{LVQ1}

since there $2N_{mi} < N_i$ for all i=1,...,K. Hence, LVQ1 gets away from these (repelling) points. However, Bayes borders are in these regions of 'negative' density and consequently a correct placement of prototypes could be impossible.

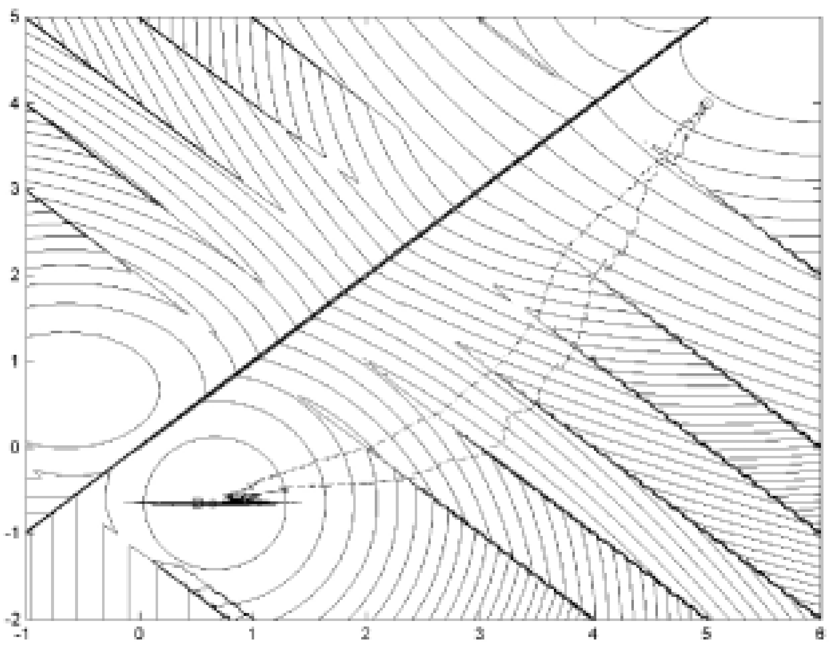


Fig.17. Convergence of random LVQ1 for a two-class problem with 2 prototypes for several values of alpha. Note that the algorithm makes a 'noisy' gradient descent until it reaches a minimum. Then, it starts to oscillate and finally converges.

Figure 18 display a 3-class problem in which there are three radial gaussians centred at (2,4), (3,3) and (4,4) with variance 0.5. In figure 19 we can see equation (11) (LVQ1's density) for this 3-class problem. Observe that the density has a deep valley (regions with a negative value) around Bayes border so LVQ1 cannot place any prototype in the valley. Figure 20 shows 270 prototypes computed with a training set of random samples extracted from the problem (30000 samples). Note that prototypes are repelled from Bayes border since LVQ1's density is negative there.

A simple solution to the problem of negative density would be introducing a scaling factor in the second term of equation 11. Then, we could control in some degree negatives valleys. This simple extension leads to the so-called **generalised LVQ1 (GLVQ1) algorithm** that is presented in on-line and batch versions. In the on-line version, the scaling factor is simply introduced in the amount of change of the winning prototype for the case in which its label does not agree with the input pattern's label:

$$\mathbf{m}_i[n+1] = \begin{cases} \mathbf{m}_i[n] + \alpha[n](\mathbf{x}[n+1] - \mathbf{m}_i[n]) & \text{if } \mathbf{x}[n+1] \in R_i[n] \text{ and } \mathbf{x}[n+1], \mathbf{m}_i \in \text{same class} \\ \mathbf{m}_i[n] - \lambda\alpha[n](\mathbf{x}[n+1] - \mathbf{m}_i[n]) & \text{if } \mathbf{x}[n+1] \in R_i[n] \text{ and } \mathbf{x}[n+1], \mathbf{m}_i \notin \text{same class} \end{cases} \quad i = 1,\dots,K \tag{16}$$

where λ>0. The batch version has the following update equation:

$$\mathbf{m}_i[n+1] = \frac{1}{(1+\lambda)N_{mi}[n] - \lambda N_i[n]} \sum_{j=0}^{N-1} 1(\mathbf{x}_j \in R_i[n])\big(1(\text{cl}(\mathbf{x}_j) = \text{cl}(\mathbf{m}_i)) - \lambda 1(\text{cl}(\mathbf{x}_j) \neq \text{cl}(\mathbf{m}_i))\big)\mathbf{x}_j \quad i = 1,\dots,K \tag{17}$$

where N_{mi} and N_i are defined as before. Note that GLVQ1 gives the K-means algorithm performed separately for each class when λ=0 and gives LVQ1 and batch LVQ1 (BLVQ1) when λ=1. In fact, GLVQ1 minimises the following cost function:

$$\begin{aligned} E_{GLVQ1}(\mathbf{P},\lambda) = {} & \frac{1}{2}\sum_{i=0}^{N-1}\sum_{j=1}^{K} 1(\mathbf{x}_i \in R_j)1(\text{cl}(\mathbf{x}_i) = \text{cl}(\mathbf{m}_j)) \left\|\mathbf{x}_i - \mathbf{m}_j\right\|^2 \\ & - \frac{\lambda}{2}\sum_{i=0}^{N-1}\sum_{j=1}^{K} 1(\mathbf{x}_i \in R_j)\, 1(\text{cl}(\mathbf{x}_i) \neq \text{cl}(\mathbf{m}_j))\left\|\mathbf{x}_i - \mathbf{m}_j\right\|^2 \end{aligned} \tag{18}$$

where the cl(**x**) function returns the class label of **x** and **1**(condition) is the indicator function. On-line GLVQ1 performs the *pattern-based version* of gradient descent over equation 18 and batch GLVQ1 employs Newton optimization. While on-line version is globally convergent (for λ belonging to a certain interval), batch GLVQ1 is locally convergent and must be avoided when the algorithm is not near a minimum point. Besides, the algorithm must be restarted when the dividing term of equation 17 is zero or takes a negative value.

Figures 21 and 22 show the application of GLVQ1 in the 3-class problem (figure 18). Observe that GLVQ1 can control the existence of the negative valley. Hence the supervised clustering process near Bayes borders can be performed.

For a large set of prototypes, GLVQ1 effectively performs a clustering process over a density function that could be zero at Bayes borders so the resulting nearest-neighbour classifier could estimate Bayes. However, an effective clustering process can be performed when training sets are very large and this is not the common situation in real-world problems.

Hence, for small data sets, the equilibrium points of GLVQ1 can serve as an indication of how classification accuracy could be improved in comparison with LVQ1. The minimization of E_{GLVQ1} (equation 18) ensures a training error ≤100/(1+λ)% since the minimum points accomplish that $N_{mi} > \lambda(N_i - N_{mi})$ for i=1,...,K and hence the number of correct classifications $\sum_{i=1}^{K} N_{mi} > (\lambda/1+\lambda)N$.

In this way, we could push the learning system towards a minimum of the training classification error for very large values of λ (λ>>1). However the cost function E_{GLVQ1} has more chances of being ill conditioned as λ augments since the number of feasible solutions (minimum points E_{GLVQ1}) is then reduced. Moreover, numerical instability of the learning algorithm may appear for large values of λ. Let us illustrate the effect of the regularising parameter lambda in the minimum points of the learning systems using the following example.

Suppose that we have two classes (A and B), 2 prototypes w_A and w_B (one for each class) and the following data: -.6(B), -.6(B), -.6(B), -.5(A), 1(A), 1.5(A), 2(A), 2.5(A), 3(B), 4(A), 6(B). Figure 17 shows the contour levels of E_{LVQ1}. The minimum point $\mathbf{w}^*$=(0.6, -0.65) with a frontier point F=-0.025 achieves a classification error of 3/11 while the minimum error is 2/11. The minimum error would be fulfilled for any F between –0.6 and –0.5. If the cost function moves $\mathbf{w}^*$ to the left side, the classifier would achieve the minimum error. Figure 24 shows the cost function of GLVQ1 for several values of λ. For λ=1.25, the minimum points are moved and $\mathbf{w}^*$ is placed near the optimal solution. Note that the minimum points of GLVQ1 disappear as λ augments. Hence, λ determines the number of feasible solutions and can be an effective parameter for improving the classification error.

Since GLVQ1 has an additional parameter (λ) for controlling better the classification error, it can applied to any classification problem. Figure 25 displays the application of GLVQ1 in Ripley's synthetic problem, a two-class problem (Ripley, 1994).

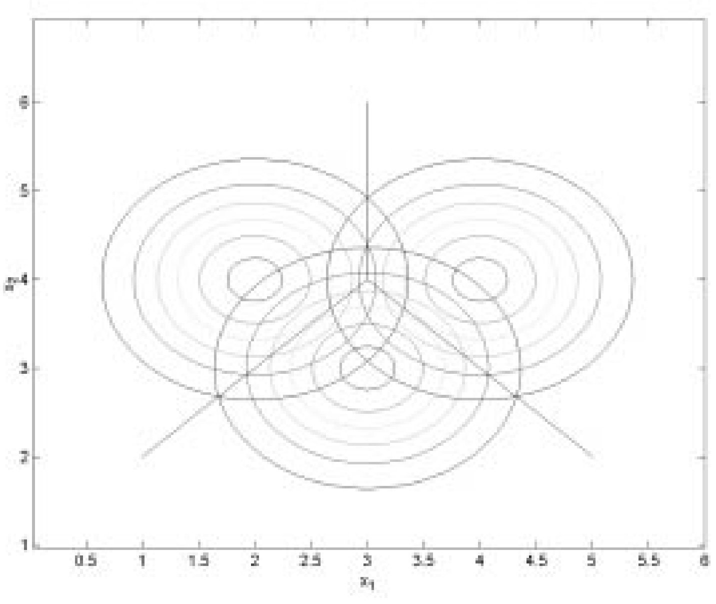


Fig.18. A 2-D three-class pattern recognition problem. There are three radial normal class densities (centred at (2,4), (3,3) and (4,4) with variance 0.5) and all classes has the same priors (1/3). Bayes borders are the lines displayed.

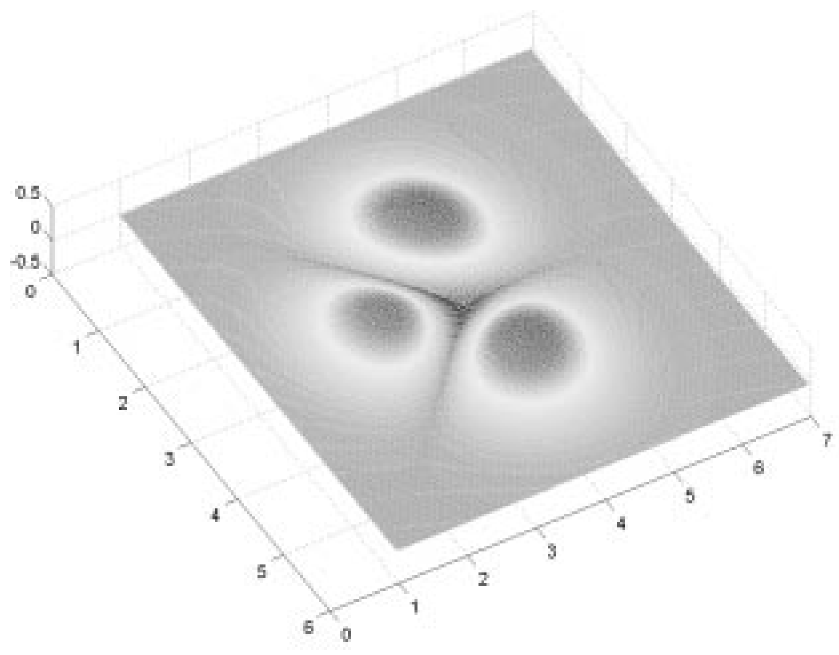


Fig.19. LVQ1's density distribution for the 3-class problem (figure 18). Note that the density has a deep (negative) valley around Bayes border.

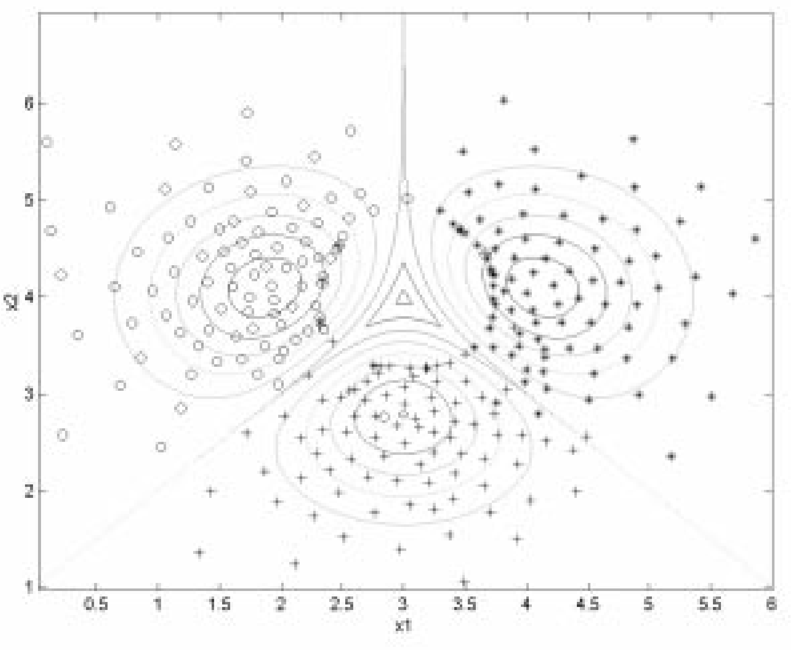


Fig.20. Prototypes computed with LVQ1 and LVQ1's density function for the 3-class problem (figure 18). Observe that prototypes are placed according to the density displayed. E.g. prototypes are repelled from valley where there are a negative density.

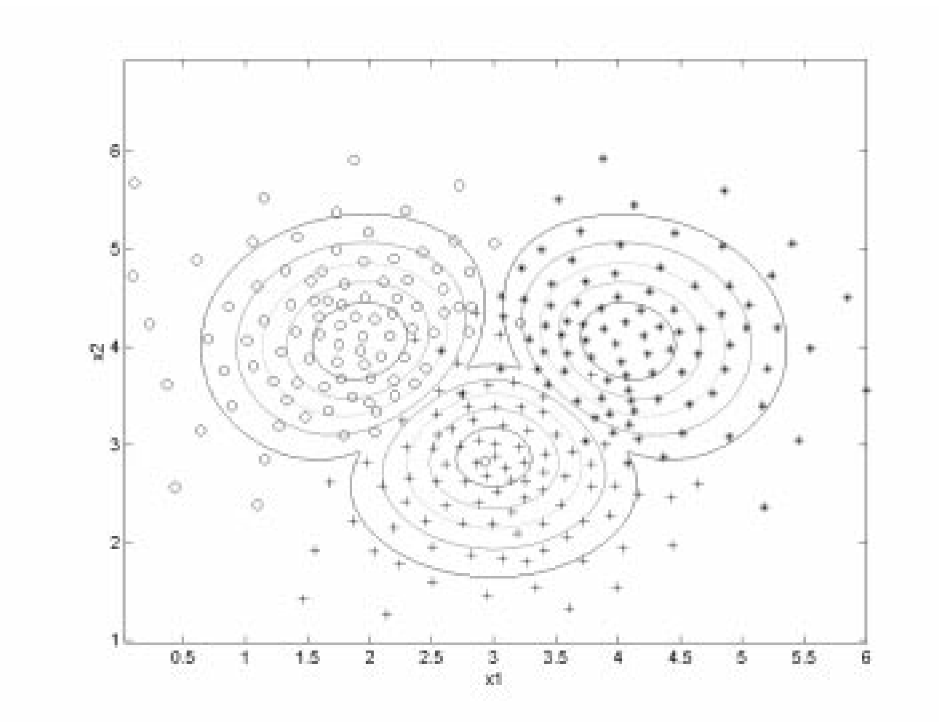


Fig.21. Prototypes computed with GLVQ1 (λ=0.5) and GLVQ1's density function for the 3-class problem (figure 18). Note that now the negative valley in the density has disappeared and GLVQ1 places prototypes near Bayes borders.

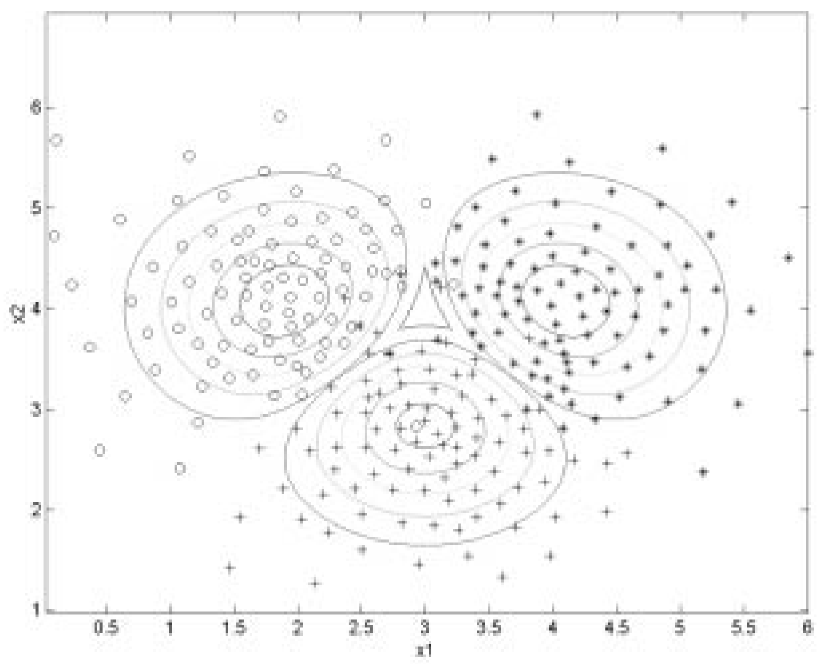


Fig.22. Prototypes computed with GLVQ1 (λ=0.7) and GLVQ1's density function for the 3-class problem (figure 18). Note that the negative valley in the density has appeared again. However the negative region is smaller than in the LVQ1 case. GLVQ1 still places prototypes near Bayes borders.

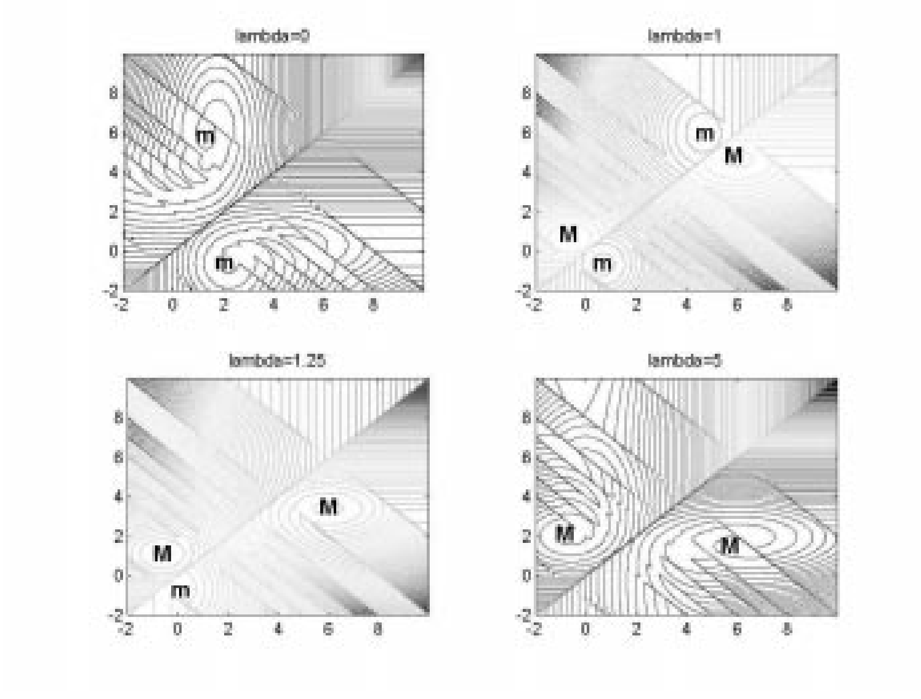


Fig.23. Contour levels of E_{GLVQ1} for several values of λ with 2 prototypes in a two-class problem. M and m denotes maximum and minimum points respectively. Note that the cost function losses its minimum points as λ augments.

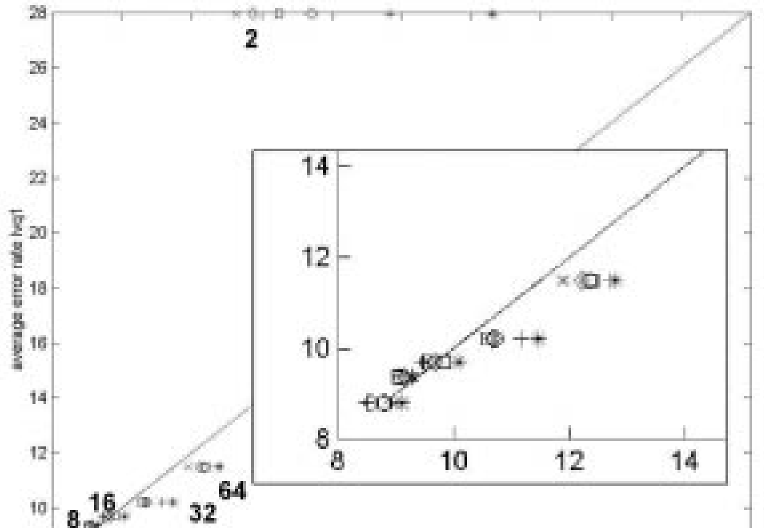


Fig.24. Average test error of LVQ1 and GLVQ1 in Rypley's synthetic problem for different number of prototypes (2,4,8,16,64). GLVQ1 use several values of λ. marked as *(1.5), +(2.0), circle(2.5), square (2.75), diamond (3) and x (3.5). Note that GLVQ1 achieves the best global result (8 prototypes) and is useful for 2,4,8 and 16 prototypes.

4.1.4. Generalization of Kohonen's LVQ algorithms: Voronoi Data-dependent Partitioning revisited.

Large-samples results of LVQ1 indicate that it performs a vector quantization process over a probability density function that is zero at Bayes borders for two-class problems (equation 10). However, LVQ1 can have some problems for problems with more classes (e.g. negative valleys near Bayes borders). Heuristic modifications of LVQ1 has been proposed to improve the supervised clustering process (e.g. Kohonen's LVQ2 and LVQ3). However, it is possible to derive a principled learning algorithm (called BLVQ) based on the idea of performing clustering over a density with zero's at Bayes borders. Figure 25 displays the application of BLVQ in the synthetic 3-class problem (figure 18). The algorithm effectively places prototypes according to the following probability density function that is zero at Bayes borders:

$$p(\mathbf{y}|\mathbf{x}\in R_i) = p(\mathbf{x}|C_r)P(C_r) - \max_{\substack{j=1,\dots,C\\ j\neq r}} p(\mathbf{x}|C_j)P(C_j) \quad \forall \mathbf{x}\in R_i \quad i=1,\dots,K \tag{19}$$

where $p(\mathbf{x}|C_r)P(C_r) = \max_{j=1,\dots,C} p(\mathbf{x}|C_j)P(C_j) \quad \forall \mathbf{x}\in R_i$.

BLVQ is based on Voronoi data-dependent partitioning (§4.1.1). First, it performs a VQ process over the modified density and then assigns labels to prototypes (figure 26). BLVQ is a very fast batch learning algorithm for Euclidean NN classifier based on Newton optimization and uses less arithmetic operations than Kohonen's LVQ algorithms for the same number of epochs. The algorithm can be also derived for a more general class of NN classiifer based on the d_q metric.

The clustering process of BLVQ is based on the idea of estimating the two majority classes in each Voronoi cell using the density of training samples that fall in them. Then BLVQ perfors the same correct process executed by LVQ1 for 2-class problems. Accordingly, the BLVQ cost function is defined as:

$$E_{BLVQ}(\mathbf{P}) = \sum_{j=0}^{N-1}\sum_{i=1}^{K} 1(\mathbf{x}_j \in R_i)\left(1(\mathbf{x}_j \in \hat{C}_{ri}) - 1(\mathbf{x_i} \in \hat{C}_{si})\right)\left\|\mathbf{x}_j - \mathbf{m}_i\right\|^2 \tag{20}$$

where $\hat{C}_{ri}$ is the first majority class of data samples that fall in R_i, $\hat{C}_{si}$ is the second majority class of data samples that fall in R_i. These two classes are estimated using the distribution of training samples in Voronoi regions. The update equation is then:

$$\mathbf{m}_i[n+1] = \left. \frac{\sum_{j=1}^{N_{R_i,\hat{C}_{ri}}} \mathbf{x}_j \middle| R_i, \hat{C}_{ri} - \sum_{u=1}^{N_{R_i,\hat{C}_{si}}} \mathbf{x}_\mathbf{u} \middle| R_i, \hat{C}_{si}}{N_{R_i,\hat{C}_{ri}} - N_{R_i,\hat{C}_{si}}} \right|_{\text{estimated at time n}} \qquad i = 1,\dots,K \tag{21}$$

where $N_{Ri,\hat{C}ri}$ is the number of data samples that fall in R_i and belong to class $\hat{C}_{ri}$, $N_{Ri,\hat{C}si}$ is the number of data samples that fall in R_i and belong to $\hat{C}_{si}$, $\mathbf{x}_j | R_i, \hat{C}_{ri}$ are the samples that fall in R_i and belongs to $\hat{C}_{ri}$ and $\mathbf{x}_\mathbf{u} | R_i, \hat{C}_{si}$ are the samples that fall in R_i and belongs to $\hat{C}_{si}$.

Since VQ processes are consistent for density estimation (Lugosi & Nobel, 1996), BLVQ can estimate accurately a density function that is zero at Bayes borders. Consequently large-sample results for the BLVQ-based NN classifier could be arbitrary near Bayer classifier.

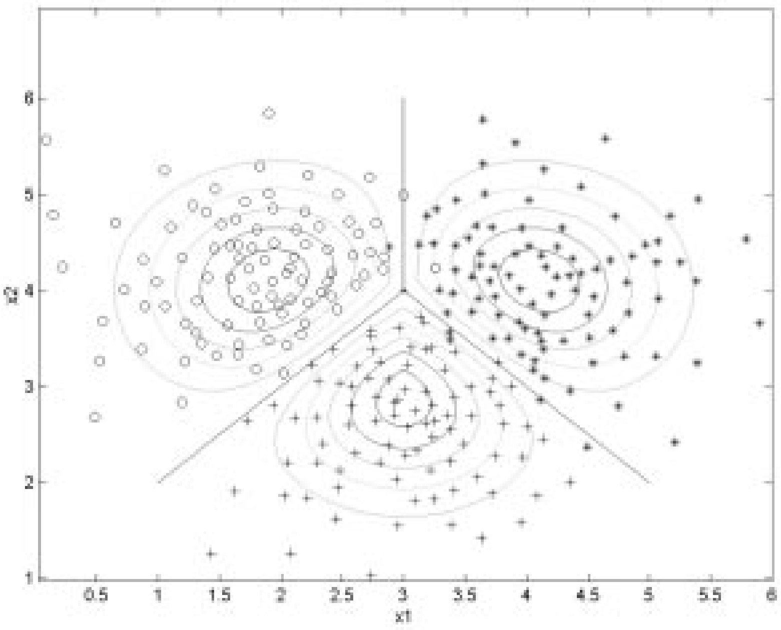


Fig.25. Prototypes computed with BLVQ and density function of equation 19 for the 3-class problem (figure 18). Observe that prototypes are placed according to the density displayed that is zero at Bayes borders.

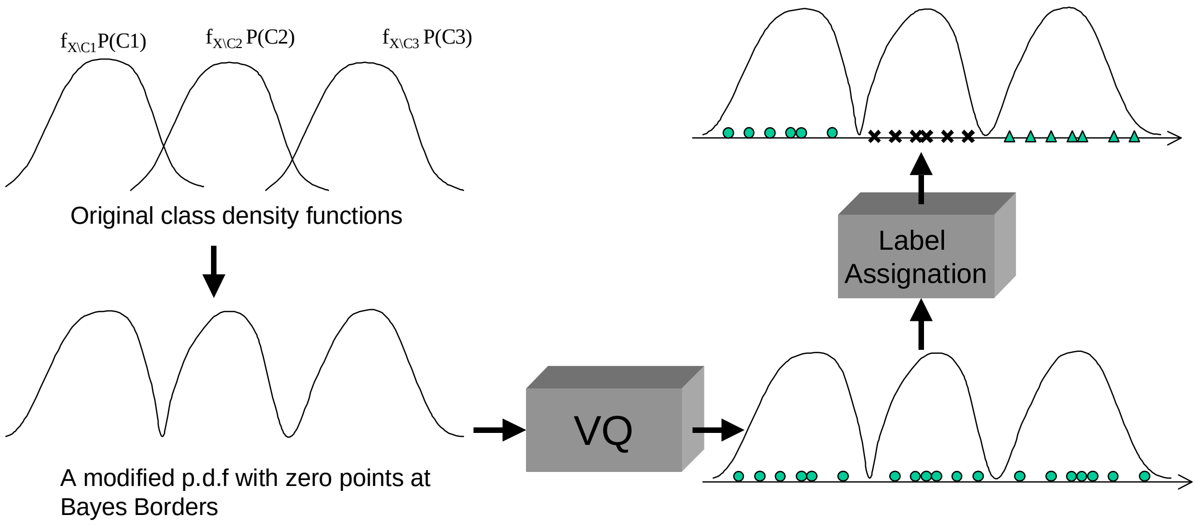

$f_{X\backslash C1}P(C1)$
$f_{X\backslash C2}P(C2)$
$f_{X\backslash C3}P(C3)$
Original class density functions
A modified p.d.f with zero points at Bayes Borders
VQ
Label Assignation

Fig.26. Steps of the BLVQ algorithm: 1) Perform VQ over a modified density function that is zero at Bayes borders; 2) Assign labels to prototypes.

4.1.6. Dynamic allocation and deletion of prototypes.

Constructive (or growing or incremental) and Pruning algorithms are concerned with the problem of choosing the optimal balance between the approximation power and the estimation error of the learning system given a training set of fixed size. In multi-layer feed-forward neural networks, this problem is choosing the right architecture- e.g. the number of hidden layers and hidden units per layer. In nearest neighbour classification, it is simply the selection of the number of prototypes for each class. The first group of algorithms starts with a small model and grows it until a satisfactory solution is found. On the other hand, pruning algorithms removes those parameters of a big (and previously trained) model that are not effectively used. Much work on dynamic learning procedures (that is learning algorithms that deal with adaptive architectures) has been devoted to feed-forward neural networks. (See (Bishop, 1995) §9.5 for an introductory reference for growing algorithm and pruning algorithms using feed-forward architectures.)

Kohonen's LVQ algorithms design codebooks (or set of prototypes) for 1-NN classifiers that exhibit good generalization. Kohonen has shown that given certain conditions these local learning algorithms can estimate the Bayesian borders with arbitrary good accuracy, depending on the number of codebook vectors used (p. 206; (Kohonen, 1996)). However, in practice, we do not know the optimal number of prototypes for each class that achieves a good balance between the approximating power and the estimation error of the NN classifier. Typically, the user of the learning algorithm assigns the number of prototypes before the training process begins. Then several training sessions with different assignations are performed and finally the user choose the better session according to a pre-established criterion (e.g. choose that classifier that gives the best classification accuracy on the validation set). Clearly, a better strategy would be a dynamic assignation of the prototypes according to the errors produced in the classifier. In this way, we could perform several training sessions linked with a growing algorithm that adds new prototypes in those local regions where the misclassification error is greater. This constructive process would be repeated until the classification accuracy measured with a validation set stops decreasing and finally a pruning algorithm could be executed to remove all those prototypes that do not form class borders. According to the above considerations, we propose the following dynamic learning algorithm:

1. Train using a static LVQ algorithm until the classification error of a validation set stops decreasing
2. *Constructive part*: Add a maximum of MAV new prototypes in those Voronoi regions in which the classification error is greater. (Since the complexity of Bayes borders is locally defined, we can reduce bias (the difference between the Bayes classifier and our classifier) if we adjust *locally* each border according to the evolution of the classification error during training phase. This means give more approximating resources, say codevectors, to regions that are more difficult to classify properly in a dynamic fashion. Besides, new prototypes are added using an estimation of the attractors of the LVQ algorithms.
3. Repeat 1 and 2 until the validation error stops decreasing
4. *Pruning part*: Remove all those prototypes that do not form class borders.

4.1.7. Local stabilisation of nearest-neighbour ensembles: local averaging and local extreme.

Many learning systems are **unstable** in the sense that the models (or predictors) computed by them strongly depend on the particular training set, initial conditions and parameters of the algorithm. Besides, the use of capacity control techniques like early stopping (Prechelt, 1998) introduce some variability between predictors computed in different training sessions since the leaning algorithm can stop at different points of the cost error function.

Figure 27 shows the class borders of 20 nearest-neighbour classifiers computed with the LVQ1 algorithm for a synthetic 2-class problem (based on a mixture of non-radial gaussians). The prototypes have been computed using *a single training set* and the initial values of the prototypes and the order of training samples in memory have been modified in each training session. Besides, the early stopping technique has been employed to avoid over-fitting. Observe that the series of class borders are 'variations on a theme'. However, the question that arises now is which classifier must we choose from the pool of classifiers?

Ensemble learning emerges as a solution to this problem. The idea of ensemble methods learning is to combine a collection of learning systems (or predictors) that have been all trained in the same task. The goal of these methods is to obtain a stabilised solution from a set of unstable solutions. Two simple methods for stabilisation are voting (in classification) and averaging (in regression) (Breiman, 1997) (Breiman, 1998). However, many other ensemble methods exist. See (Sharkey, 1999) for a review of general ensemble methods.

Generally speaking, ensembles methods control capacity stabilising the set of solutions through the reduction the dependence of the combined solution on the training set and the optimization algorithms used by the members of the ensemble.

Ensembles are well-established methods for obtaining a better-combined classifier than single members of the ensemble. However, many open questions arise (Dietterich, 1997). Ensembles usually perform better than separate members do but the emerging decisions made by them are typically difficult to interpret. Hence, more emphasis must be done in **understanding how ensembles methods make decisions**. On the other hand, ensembles require large amounts of memory and demands large execution times so this limits their practical application. Consequently, there is the need of finding ways to **convert ensembles into less redundant representations** (e.g. **single classifiers** that perform as well as the ensemble).

In the context of nearest-neighbour classifiers, stabilisation can be useful when learning algorithms like LVQ1 are used. A direct approach is to apply any general ensemble method to the NN classifier that fuse at the output level (e.g. (Alpaydin, 1997)(Skalak, 1997)). However, there are ways of performing stabilization inside the classifier.

Let us return to the above classification problem to see how stabilise the solutions of the members of the ensemble. If many class borders emerge around a 'mean' border (figure 27) is because prototypes of the ensemble are grouped in clusters (see figure 28). Why are formed these clusters? There are several reasons. As we have shown before, (finite-sample) attractors of (on-line) LVQ1 depend on the order of training samples in memory. For a given order, LVQ1 gives more relevance to some training samples. (It would be equivalent than generating a bootstrap replicate (Efron & Tibshirani, 1994) from the training set and then applying the batch version of LVQ1 over the replicate.) Figure 16 shows the attractors for different orderings in memory: they are all around a centre (the minimum of the cost function). Besides, for a given ordering, early stopping can provoke to stop at different points. Finally, LVQ1 stops at a local minimum and many local minima can exist for a classification problem. Distributions of prototypes arranged in clusters have also been observed in other on-line learning algorithms (e.g. LVQ2 and LVQ3).

Hence, if prototypes are arranged in natural groups, we could compute cluster centroid and use them to form a single classifier of the same complexity of the members of the ensemble. E.g. if we compute 100 NN classifier with 8 prototypes, we can expect to find a distribution of 8*100 prototypes grouped in 8 clusters of 100 points. Then, we can obtain a NN classifier with 8 prototypes computing cluster centroid in each cluster (see figure 28). This method is called **local averaging** and has the following steps:

1. Execute Q times a Kohonen's LVQ algorithm that is started with different initial conditions of a codebook (of size K) and different position of the training data in the memory array.
2. Compute K cluster centroids over the totality of the computed codevectors (QxK) in the training sessions using the batch K-means with the restriction that the algorithm only will average codevectors of the same class. The execution of the K-means is stopped when the validation error of a NN classifier that uses the resulting averaged codebook (the K current centroids) increases.

Local averaging is, in fact, the local application of the averaging technique used in regression. The most remarkable difference between our method and typical averaging (or any other combination technique) is that the combined NN classifier does not employ the ensemble of NN classifiers anymore. Instead, it has a single codebook (of the same size than the members of the ensemble) that is formed with the K centroids. Besides, the way in which local averaging works is easy to interpret geometrically as we have just seen.

Local averaging can improve in some cases the classification accuracy since it reduces the variance of prototypes as the result of averaging an ensemble of particular 'bootstrap' replicates. Hence, local averaging pretend to stabilise implicitly the class borders through an explicit stabilisation of the prototypes.

Figure 29 shows the application of local averaging in the above synthetic problem. In figure 30, we compare local averaging and voting (i.e. an input pattern is assigned according to a majority vote among the members of the ensemble) in three problems. While both methods achieve a similar accuracy (local averaging outperforms voting in one problem), voting uses 10, 30 and 100 times more prototypes than local averaging. Hence, *local averaging can compute a* ***single classifier*** *that achieves a similar (or even better) accuracy than ensembles.*

Another simple method of local stabilization can also be possible. Suppose that we get the extreme prototypes of the ensemble and form a single classifier with them. Figure 28 and 29 shows the possible set of extreme prototypes for the 2D problem and the NN classifier computed with this technique (**local extreme**). The algorithm to detect these extreme prototypes can be summarised as follows:

1. Execute Q times a Kohonen's LVQ algorithm that is started with different initial conditions of a codebook (of size K) and different position of the training data in the memory array.

2. Compute the fussed codebook $\mathbf{P}_F = \bigcup_{i=1}^{Q} \mathbf{P}_i$ with the Q codebooks {$\mathbf{P}_i$, i=1...Q} of the training sessions. Then, remove those codevectors in which no training data are assigned to them (pass 1). Finally, delete those codevectors that classification accuracy of the validation set stands or improves if they are not used in the NN classifier, (pass 2). This second step is repeated cyclically through the codebook until there is no 'redundant' codevector left.

Local extreme tries to stabilise class borders explicitly using these extreme prototypes. In spite of the method achieves some degree of stabilization, it is less robust than local averaging since extreme points can vary more in different training sessions. However if some very different solutions co-exist in the ensemble, local extreme can be useful since it allows combining codebooks of different sizes in an easy way.

4.1.8. Extensions for Learning based on Supervised Clustering.

A classifier based on data-dependent partitioning divides the input space in cells according to rule. The above learning algorithms are devoted to nearest-neighbour cells, one of the simplest forms of partition. Advanced VQ based on modular and tree structures could be employed to derive new forms of classification. Besides, more complex clustering learning algorithms like the EM (expectation-maximation) algorithm or fuzzy clustering algorithms (e.g. soft K-means) could be employed in nearest-neighbour partitioning and also in the advanced VQ schemes. These more advanced clustering algorithms could be also used for local averaging.

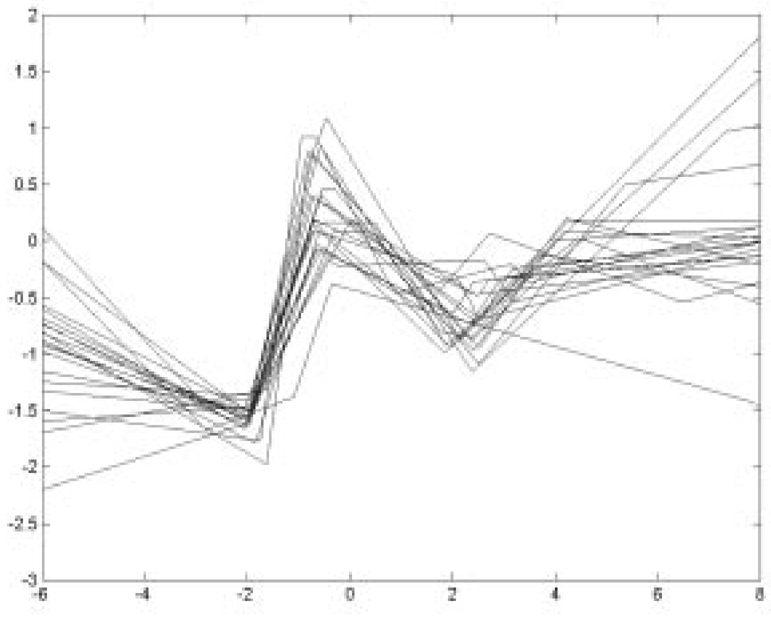


Fig.27. Class borders of 20 nearest-neighbour classifiers computed with the LVQ1 algorithm for a synthetic 2-class problem based on a mixture of non-radial gaussians. The prototypes have been

computed using a single training set and varying the initial values of the prototypes and the order of training samples in memory. The early stopping technique has been employed to avoid over-fitting.

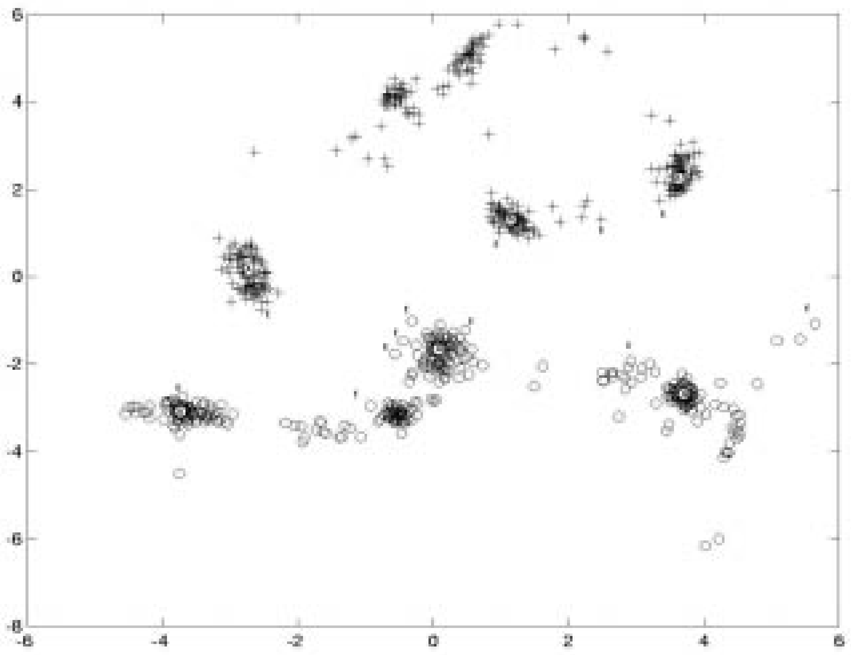


Fig.28. Prototypes computed using LVQ1 in 100 training sessions for 8 prototypes. Centres computed with local averaging that form class borders at denoted by A. E marks those prototypes that local extreme could select.

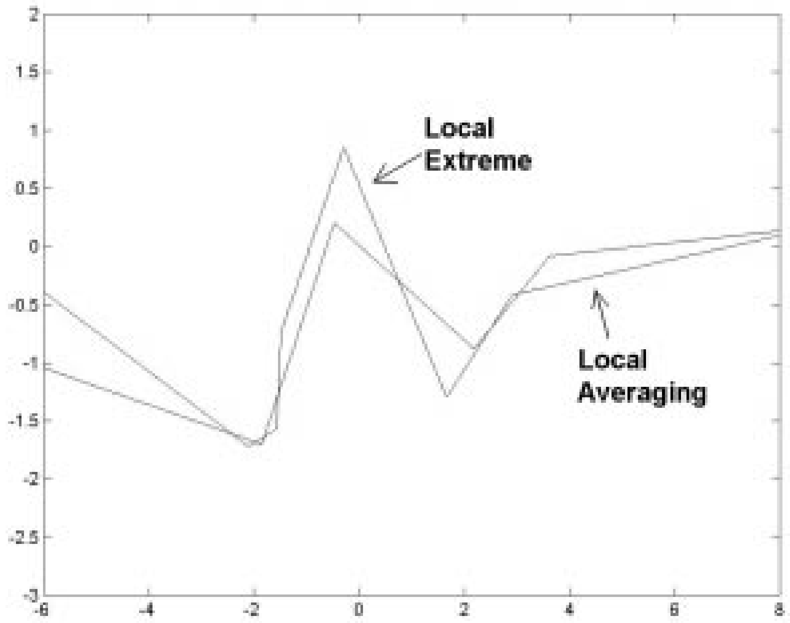


Fig.29. Stabilised class borders from an ensemble of NN classifiers (figure 27) using the proposed local stabilisation techniques.

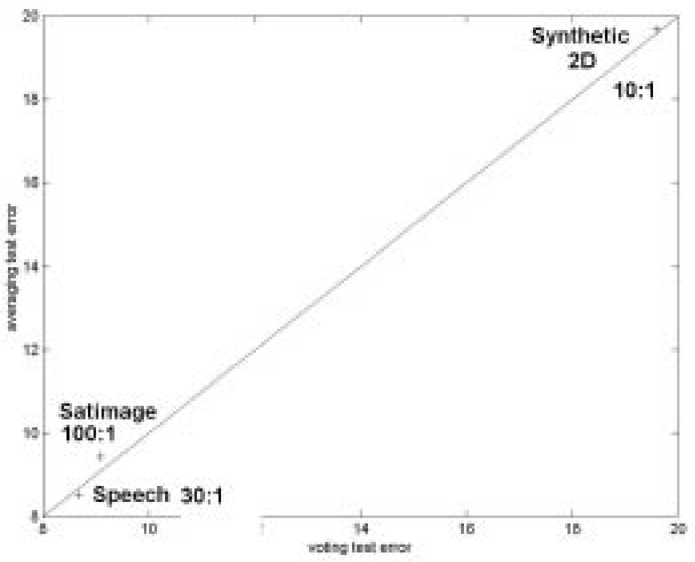


Fig. 30. Test error of local averaging and voting for Speech, Satimage and synthetic 2D problems. Note that both methods achieve a similar accuracy and averaging outperforms voting for the Speech problem. We show the ratio between the number of prototypes of the 'voting' ensemble respect to 'local averaged' NN classifier.

4.2. Learning algorithms for large margin nearest-neighbor classifiers

In classification problems, the learning machine must assign input patterns to one of the pre-defined categories. Typically, these systems are designed to *minimise the number of misclassifications in the training set*. However, recently it has been showed that, in order to ensure a small generalization error, we should *also take account the confidence of the classifications*. Then classifiers must also be designed to have a *large margin distribution* of the training samples; that is, the training samples must be assigned to the correct class with high confidence. A large margin distribution helps to stabilise better the solution and hence capacity can be controlled.

4.2.1. 1-nearest-neighbour classifiers with large margin: the Learn1NN algorithm.

Let us start our discussion on large margin classifier with an example. Suppose we have the 2-class problem of figure 31. It is linearly separable. However many linear classifiers solve the problem. Figure 31 also shows ten solutions computed with the LVQ1 algorithm. Suppose now that we impose a slope of 45º to the linear classifier. Again, many solutions coexist. Nevertheless, it has been observed (e.g. (Vapnik,1998b)) that the linear classifier with a slope of 45º that achieves a maximal separation (or margin) between the extreme data points (or support vectors) of each class controls effectively capacity and consequently is highly generalizable even if the input space has a high dimensionality (see figure 32). Note that the optimal margin (OH) hyperplane is more robust with respect training patterns and parameters: a slightly variation on a test pattern or on the value of the line will presumably not affect the classification accuracy (Smola et al., 1999). Consequently, OH is more “reliable” since it has the largest margin and then can achieve better generalization performance. (Observe that local stabilization can also make possible place borders with greater margin so capacity would be controlled.)

Figures 33 and 34 show OH for two examples of a two-class pattern recognition problem, one separable and one not. The application of the **Learn1NN algorithm** is also shown in these problems and converges to OH. Learn1NN is based on performing gradient-descent over a function that minimises the number of misclassifications while maximises the average margin of training samples (e.g. many training samples are classified with high confidence) (see figure 37). The learning algorithm has the remarkable property of using only few training samples to form class borders (e.g. support vectors). Besides, the under-computing of the gradient learning algorithm prevents the algorithm to over-fit training data (see figure 35 and 36). Over-parametrised Learn1NN-based classifiers achieve a similar rate than simpler solutions since the learning algorithm is biased toward smooth solutions (e.g. the effective number of prototypes is held constant). Let us see how Learn1NN is derived.

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 {$d_i(\mathbf{x})$, i=1...C} (where C is the number of classes) use the nearest models of a mixture for each class. The discriminant functions are derived as the following simplification of Parzen windows:

$$d_i(\mathbf{x}) \approx \frac{G(\mathbf{x}-\mathbf{m}_W^i;\gamma)}{\sum_{i'=1}^{C} G(\mathbf{x}-\mathbf{m}_W^{i'};\gamma)} \qquad i=1,\dots,C \tag{22}$$

where $\mathbf{m}_W^i$ is the nearest centre of the mixture model to **x** that belong to class i using the distance metric d (e.g. $d=\left\|\mathbf{x}-\mathbf{m}_W^i\right\|^2$) and $G(\mathbf{x}-\mathbf{m}_W^i;\gamma)=G(d(\mathbf{x},\mathbf{m}_W^i);\gamma)$.

The learning process is based on the minimization of a modification of the cross-entropy error function for multiple classes:

$$L=-\frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=1}^{C} y_{ji}\, d_j(\mathbf{x}_i) \tag{23}$$

The absolute minimum with respect to the {$d_j(\mathbf{x}_i)$} occurs when all the training points are **correctly classified with the maximum confidence**, that is when $d_j(\mathbf{x}_i)=y_{ji}$ for all j and i. An interesting property of equation (23) is that the training set's outliers cause a lower impact than in the cross-entropy error ((Bishop, 1995)§6.9) case. Besides, equation (23) is equivalent than **averaged margin function for two-class problems**.

Let $G(d(\mathbf{x},\mathbf{m});\gamma)=\exp\left(-\|\mathbf{x}-\mathbf{m}\|^2/2\sigma\right)$ and $L=-\sigma/N\sum_{i=0}^{N-1}\sum_{j=1}^{C} y_{ji}\, d_j(\mathbf{x}_i)$. (This modification of the loss function is made for analytical convenience since the added constant does not affect the desired points of convergence.) In this way, the pattern version of gradient descent over L gives

$$\begin{aligned}
&\text{if } \mathbf{x}[k]\in \text{class j}, \mathbf{m}_i^j[k-1]=\mathbf{m}_W^j[k-1]\\
&\qquad \mathbf{m}_i^j[k]=\mathbf{m}_i^j[k-1]+\alpha d_j[k-1](\mathbf{x}[k])(1-d_j[k-1](\mathbf{x}[k]))(\mathbf{x}[k]-\mathbf{m}_i^j[k-1])\\
&\text{if } \mathbf{x}[k]\in \text{class l}, \mathbf{m}_i^j[k-1]=\mathbf{m}_W^j[k-1]\\
&\qquad \mathbf{m}_i^j[k]=\mathbf{m}_i^j[k-1]-\alpha d_j[k-1](\mathbf{x}[k])d_l[k-1](\mathbf{x}[k])(\mathbf{x}[k]-\mathbf{m}_i^j[k-1]) \qquad (24)\\
&\text{otherwise}\\
&\qquad \mathbf{m}_i^j[k]=\mathbf{m}_i^j[k-1]
\end{aligned}$$

where $\mathbf{x}[k]=\mathbf{x}_{(k-1)\bmod N}$ $k\geq 0$ if we pick up cyclically, $\mathbf{m}_i^j$ is the i-th prototype of class j and $\mathbf{m}_W^j$ is the nearest prototype of class j to x in the Euclidean sense, that is $\mathbf{m}_W^j=\arg\min_{\mathbf{w}_i^j}\|\mathbf{x}-\mathbf{m}_i^j\|^2$.

Note that we can also derive a simplified learning rule for the case $\sigma\rightarrow\infty$. Then all the discriminant functions tend to 1/C (where C is the number of classes) since $\exp(-\bullet/2\sigma)\to 1$. The algorithm have two parameters: the step size α that controls the precision of computation of the minimum points (like LVQ1) and the locality parameter σ that controls the number of minimum points of the cost function (see figure 38). Note that the algorithm moves prototypes only when discriminant functions are around 0.5. Consequently, this learning algorithm places hyperplanes that are completely determined by the patterns closest to them.

In the recall phase, (hard) 1-NN classification rule is then employed since the discriminant functions induces the same classification borders and are more expensive computationally. However, soft labels also can be derived using the discriminant functions when a post-processor was used.

The generalization error of learn1NN-based NN classifiers depends not on their VC-dim, but on a scale-sensitive version of the VC-dim called **fat-shattering dimension** since learn1NN maximises the margin distribution of training data. Feed-forward networks trained with backpropagation also depend on the fat-shattering dimension (Barlett, 1998). In these systems, the fat-shattering dimension depends on the size of their weights but not on the size of the architecture and the generalization error is worse as the size of the weights augments. Nearest-neigbour classifiers with the form of equation 22 with 2 prototypes and gaussian kernels are equivalent to a perceptron with a sigmoid as an activation function:

$$\begin{aligned} d(\mathbf{x}) &= \frac{\exp\left(-\|\mathbf{x}-\mathbf{m}_1\|^2 / 2\sigma\right)}{\exp\left(-\|\mathbf{x}-\mathbf{m}_1\|^2 / 2\sigma\right)+\exp\left(-\|\mathbf{x}-\mathbf{m}_2\|^2 / 2\sigma\right)} = \\ &= \frac{1}{1+\exp\left(-\frac{1}{\sigma}\left(\mathbf{x}^{\mathrm{T}}(\mathbf{m}_2-\mathbf{m}_1)+\frac{\|\mathbf{m}_2\|^2-\|\mathbf{m}_1\|^2}{2}\right)\right)} = \frac{1}{1+\exp\left(-\left(\mathbf{x}^{\mathrm{T}}\mathbf{w}'+c\right)\right)} \end{aligned} \qquad (25)$$

where T denotes transpose.

As equation 25 indicates, the weight of a perceptron is equivalent to the difference between prototypes that form the hyperplane of NN classifier. Consequently, a small norm of the difference between prototypes could ensure good generalization as the theoretical study of (Barlett, 1998) in neural networks with large margin suggests. Since the norm of the difference augments during learning, since large prototypes augment the slope of the discriminant function and hence the margin on the classifications (see e.g. figure 38), a very **early stopping** can help to avoid over-training. Other heuristic techniques like **weight decay** (Krogh & Hertz, 1992) adapted to our problem, e.g.

$$L_{\lambda} = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=1}^{2} y_{ji}\, d_j(\mathbf{x}_i) + \frac{\lambda}{2}\|\mathbf{w}_1-\mathbf{w}_2\|^2, \qquad (26)$$

pursue the same goal and can also strongly recommended. In fact, early stopping and weight decay can achieve similar results (see figure 39).

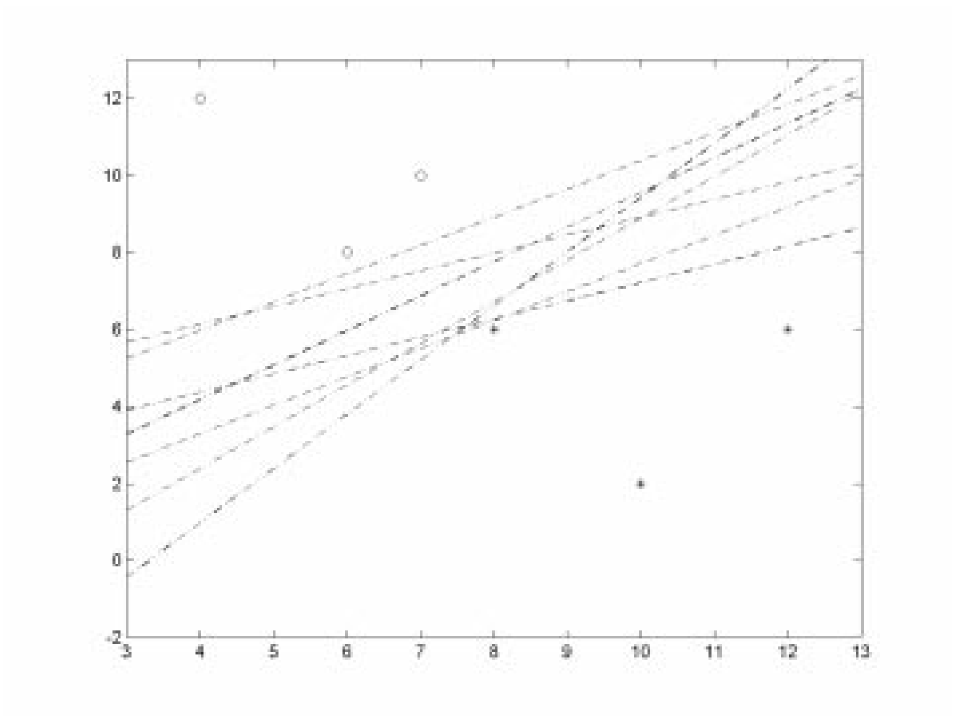


Fig.31. A toy two-class problem that is linear separable. We show the class border of ten linear classifiers (i.e. a 1-nearest-neighbour classifier with 2 prototypes) computed with the LVQ1 algorithm.

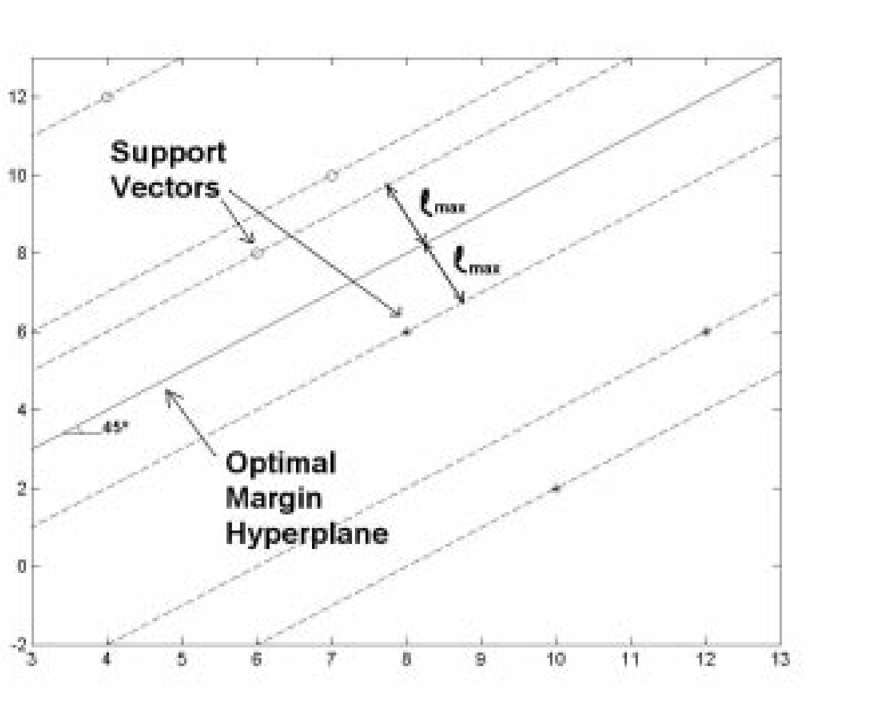


Fig.32. Optimal margin hyperplane for the toy problem (figure 31). This line has a slope of 45º and achieves a maximal separation between the extreme data points of each class. These extreme points are also known as support vectors.

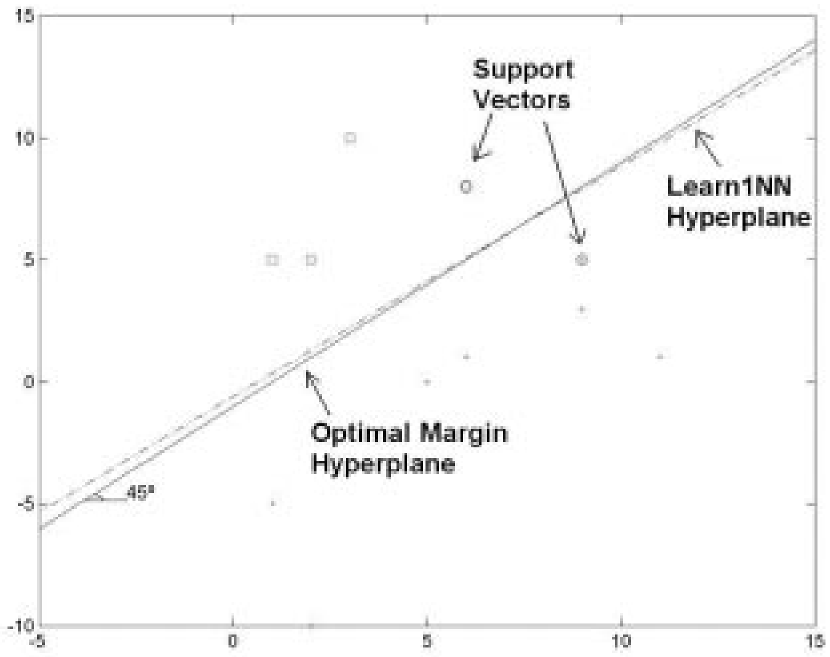


Fig.33. Optimal Margin Hyperplane (OH) for a second toy problem that is linearly separable. We also show the class border of the NN classifier computed with learn1NN. Observe that it converges to OH.

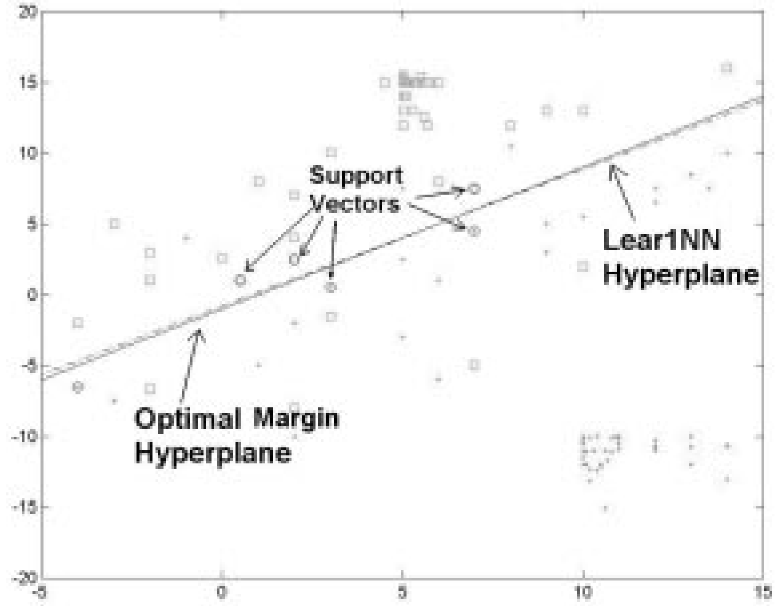


Fig.34. Optimal Margin (OH) and learn1NN hyperplanes for another toy problem that is not linearly separable. Again both solutions converge.

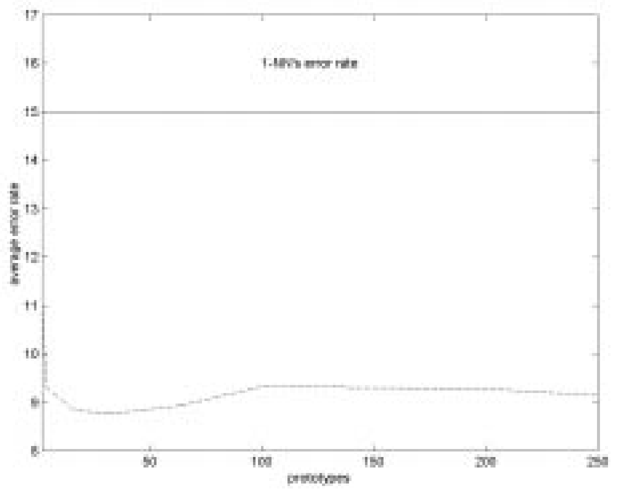


Fig.35. Error rate estimated with a test set of size 1000 for Ripley's problem. Dashed line shows average error rate over 100 runs of learn1NN for a different number of prototypes. Training set size is 250. Solid line shows the error rate of the 1-nearest-neighbour classifier formed with the whole set of prototypes. The best Learn1NN result is near Bayes error (8%). Besides, learn1NN does over-fit training data, like e.g. LVQ1 (see figure 13), as the number of prototypes augments. The over-parameterised NN classifier with 250 prototypes (the same number than training points) is slightly worse than the best solution and even better than solutions between 100 and 200 prototypes.

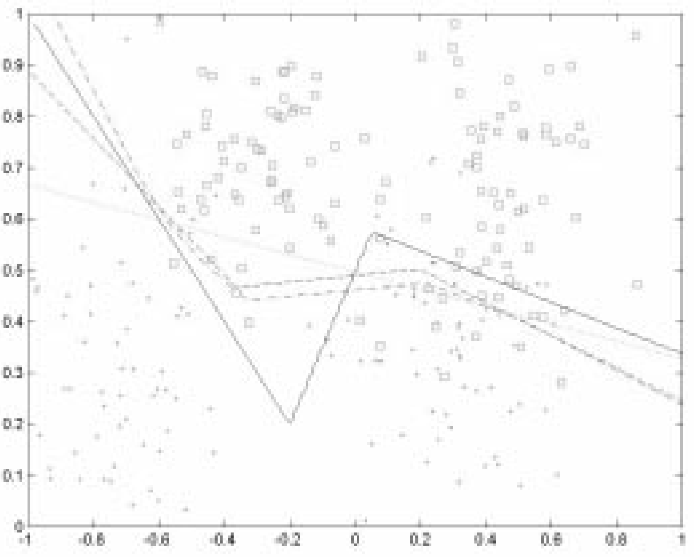


Fig.36. Ripley's synthetic training set with the Bayes border (solid line) and the class borders computed with lear1NN for 2 (dotted line), 6 (dashed line) and 32 (dashdot line) prototypes. The test error for these classifiers was 10.7%, 9.4% and 8.4 % respectively. Note that the NN classifier with 32 prototypes do not over-fit training data.

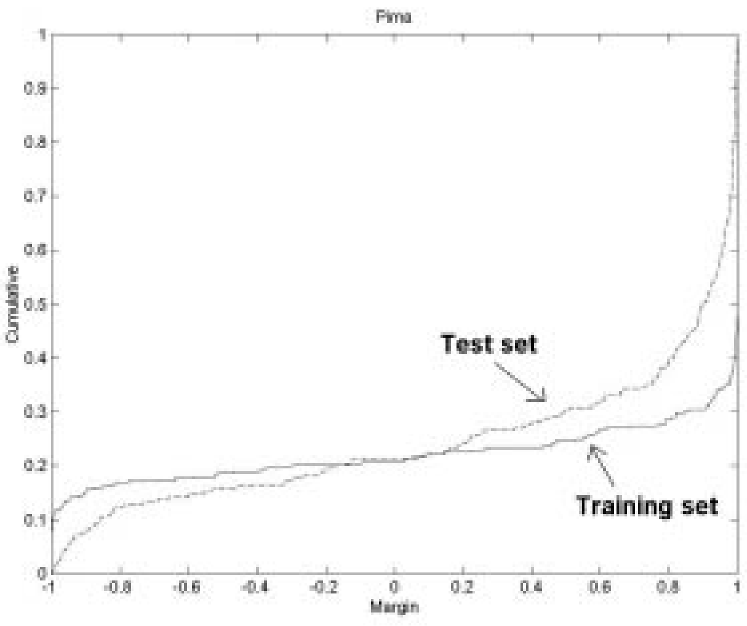
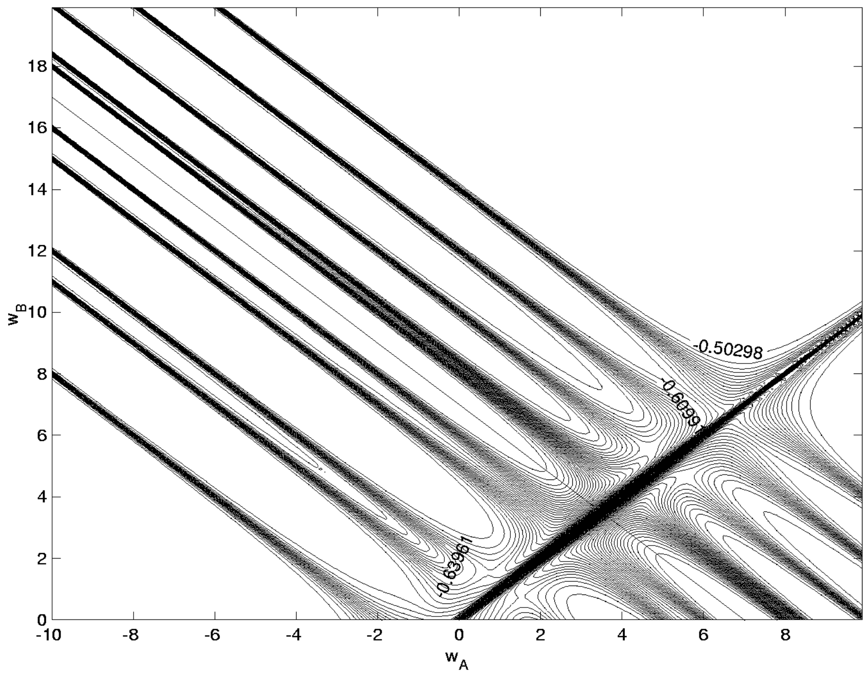
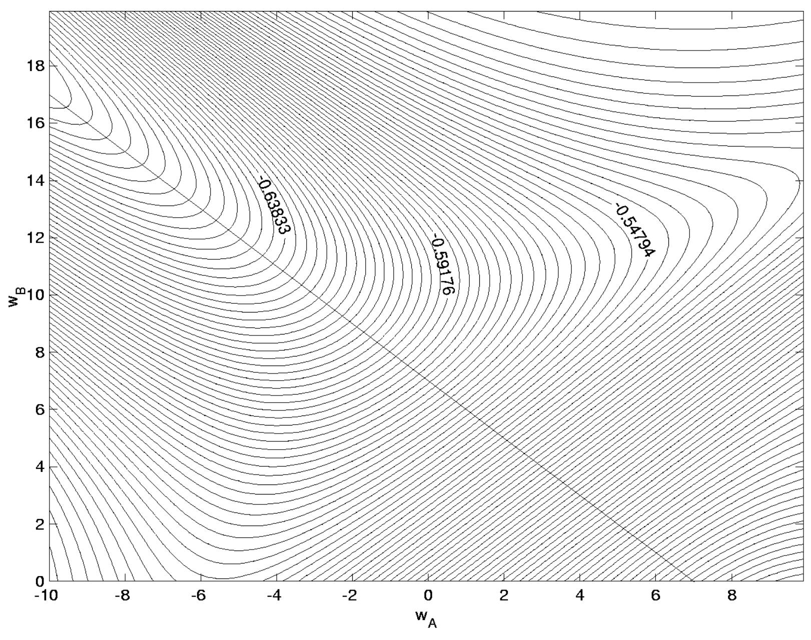


Fig.37. Cumulative distributions for Pima database after we complete a training session with learn1NN. Observe that only few samples have a margin near 0.5. Samples around a margin of 0.5 are used to compute prototypes and may correspond to support vectors.



a)



b)

Fig.38.Contour levels of cost function of Learn1NN for an example with a solution in line $(w_A+w_B)/2=3.5$. This solution minimises the number of misclassification with maximal margin between support vectors: a) $\sigma=0.325$ and b) $\sigma=25$. Sigma controls the number of local minimum points of the cost function.

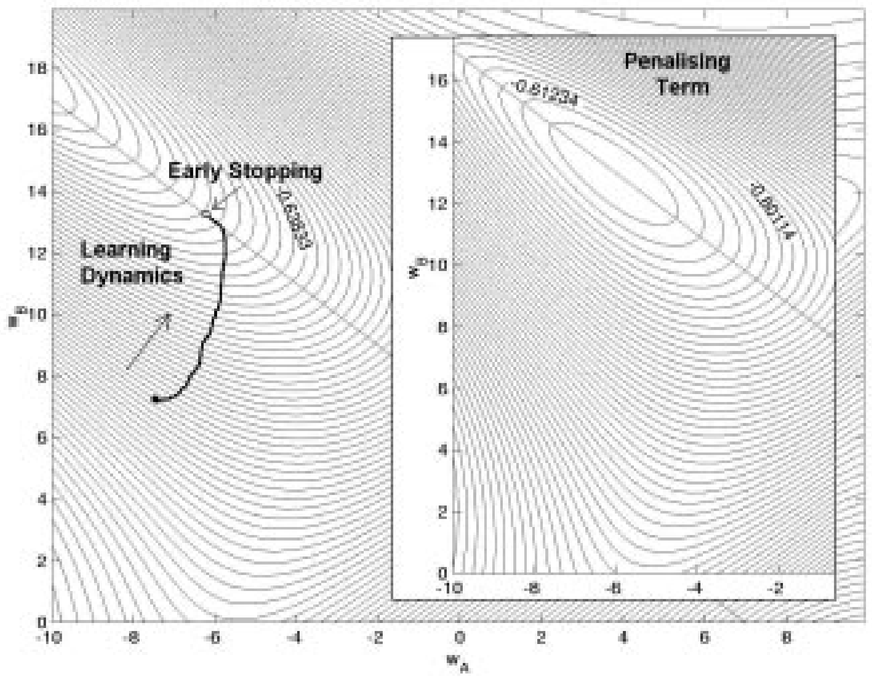


Fig.39. Practical equivalence of early stopping and the addition of a penalising term in the cost function. While early stopping presumably will stop when the learning algorithm gets to F, the penalising term modifies the minimum point.

4.2.2. Soft K-NN classifiers with large margin

K-NN techniques are very successful examples of local learning systems. For every testing pattern, they estimate posterior class probabilities with a fixed number of training points. In this way, good generalization is guaranteed since there is always enough data to compute the estimations. By contrast, these systems have poor approximation capabilities due to estimate using a ratio of integers. However, the quality of the K-NN approximation can be simply improved if we use instead of the usual ratio, a Parzen window estimate computed with the K-nearest-neighbor training samples. Hence, the resulting *soft* estimation could benefit from the virtues of K-NN and Parzen estimates, since it improves the approximation quality of crisp K-NN estimators by means of a smooth interpolation and it retains their control of the training points that contribute to the estimations. The soft K-NN rule is as follows:

$$d_j(\mathbf{x};\gamma)=\sum_{i=1}^{K_j} G\left(\mathbf{x}_i{}^{j}-\mathbf{x};\gamma\right) \Bigg/ \sum_{j'=1}^{C}\sum_{u=1}^{K_{j'}} G\left(\mathbf{x}_u{}^{j'}-\mathbf{x};\gamma\right) \quad j=1...C \tag{27}$$

where G(u) is a peaked function around u=0, $\mathbf{P}_{\text{K-NN of class } j}=\{\mathbf{x}_u^j,\ u=1,...,K_j\}$ are those training points of class j among the k nearest prototypes to **x** and $K=\sum_{j=1}^{C} K_j$ .We remark that the (crisp) K-NN algorithm in equation (27) is recovered in the limit $\sigma\rightarrow\infty$. In the particular case k=2, the resulting classifier is equivalent to the 1-nearest-neighbour classification rule since only two prototypes are involved in the decision.

The above equation, so-called **Soft K-NN rule**, uses a locally defined Parzen window using the K-nearest prototypes to the input pattern. Soft K-NN improves the approximating capabilities of the posterior class probabilities estimates of its crisp counterpart through soft interpolation (figure 40) while it can gives a soft output for post-processing purposes.

However all of the training data must be retained to compute the estimations. We also present a more sophisticated version of the algorithm to allow fewer data points to be used. This includes a learning algorithm to compute a soft K-NN classifier with large margin (see figures 41,42 and 43). The algorithm is based on gradient descent over the cost function of equation (23). If $G(d(\mathbf{x},\mathbf{w});\gamma)=\exp\left(-\|\mathbf{x}-\mathbf{m}\|^2/2\sigma\right)$ and $L=-\sigma/N\sum_{i=0}^{N-1}\sum_{j=1}^{C} y_{ji}\, d_j(\mathbf{x}_i)$, the pattern version of gradient descent over equation (23) gives

$$\begin{aligned}
&\text{if } \mathbf{m}_i^j[k-1] \in \mathbf{P}_{k-NN}(\mathbf{x}[k]) \text{ and } j = \text{class index}(\mathbf{x}[k]) \\
&\quad \mathbf{m}_i^j[k] = \mathbf{m}_i^j[k-1] + \eta\, d_W(\mathbf{x}[k])(1 - d_j(\mathbf{x}[k]))(\mathbf{x}[k] - \mathbf{m}_i^j[k-1]) \\
&\text{if } \mathbf{w}_i^j[k-1] \in \mathbf{P}_{k-NN}(\mathbf{x}[k]) \text{ and } j \neq \text{class index}(\mathbf{x}[k]) \\
&\quad \mathbf{m}_i^j[k] = \mathbf{m}_i^j[k-1] - \eta\, d_W(\mathbf{x}[k])\, d_j(\mathbf{x}[k])(\mathbf{x}[k] - \mathbf{m}_i^j[k-1]) \\
&\text{otherwise} \\
&\quad \mathbf{w}_i^j[k] = \mathbf{w}_i^j[k-1]
\end{aligned} \tag{28}$$

where $\mathbf{x}[k] = \mathbf{x}_{(k-1)\bmod N}$ $\quad k \geq 0$ if we pick up cyclically, $\mathbf{m}_i^j$ is the i-th prototype of class j among the K-nearest-prototypes to **x** and

$$d_W(\mathbf{x}) = \frac{\exp\left(-\left\|\mathbf{x} - \mathbf{m}_i{}^j\right\|^2 / \sigma\right)}{\sum_{m=1}^{C} \sum_{u=1}^{K_m} \exp\left(-\left\|\mathbf{x} - \mathbf{m}_u{}^m\right\|^2 / \sigma\right)} \tag{29}$$

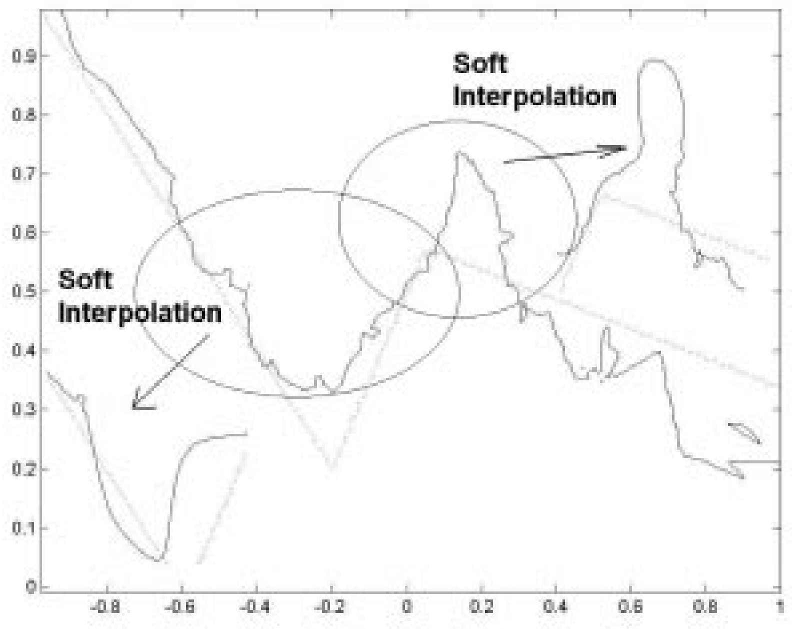


Fig.40. Soft K-NN vs. hard K-NN. The soft algorithm produces a smooth interpolation of hard K-NN's class borders.

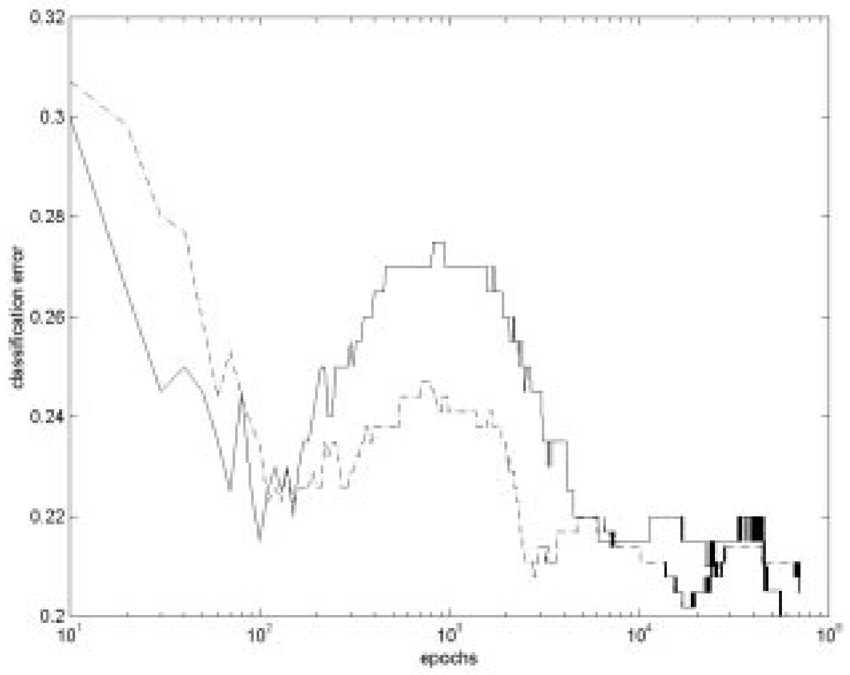


Fig.41. Evolution of training (-) and test (--) errors for Pima Indians database using the soft K-NN classifier as the number of epochs augments.

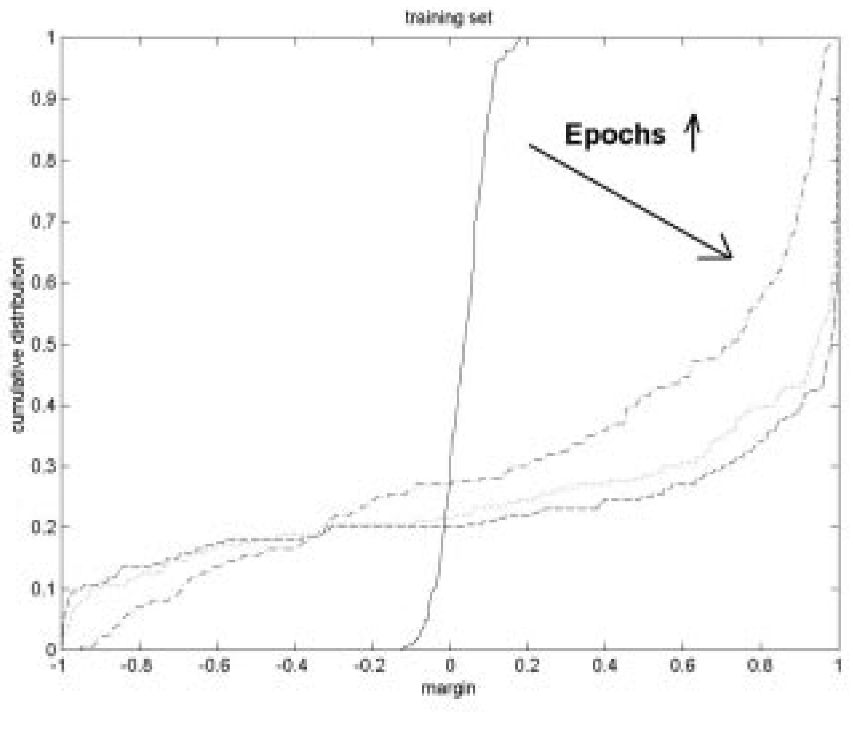


Fig.42. Evolution of the cumulative distributions for the Pima training set as the number of epochs augments. Note that the algorithm increases the confidence margin of classifications during the learning phase.

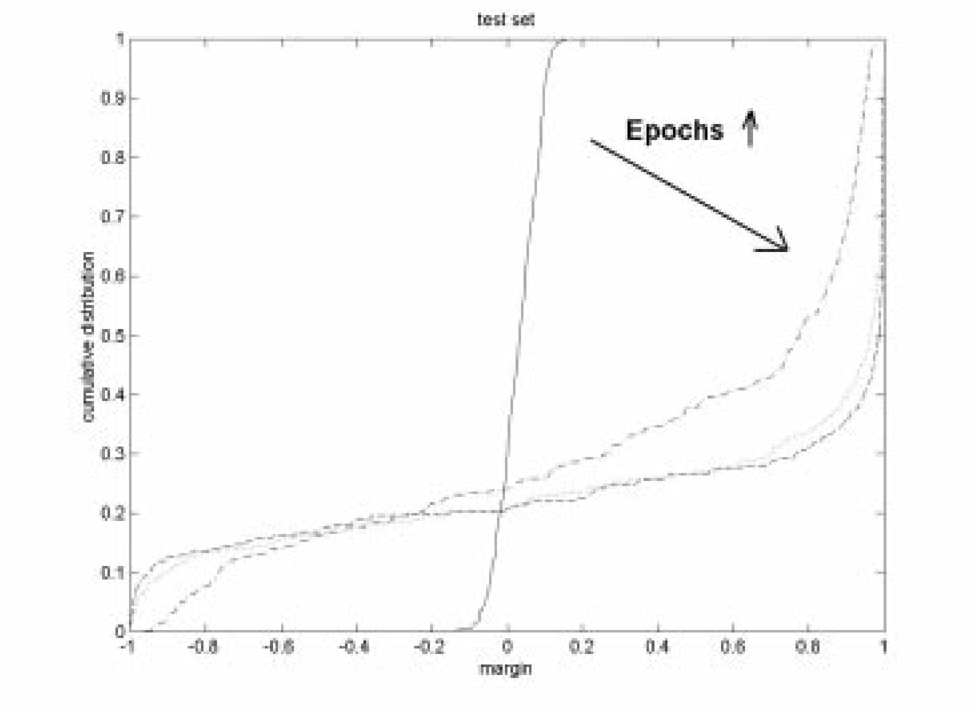


Fig.43. Evolution of the cumulative distributions for the Pima test set as the number of epochs augments. Note that the training and test distributions have a very similar form. The NN classifier then generalises well.

4.2.3. Extensions for K-NN classifiers with large margin

The large margin NN classifiers derived above minimise the number of training errors as the result of maximising the margin of correct classifications. Consequently, capacity of these classifiers does not depend on the VC dimension but a scale sensitive version called fat-shattering. Generalization error bounds could then derived if the fat-shattering was computed.

As we have shown, large margin classifiers and support vector (SVM) machines are related. SVM transform the original input space into a high-dimensional space where it places a single optimal margin hyperplane (OH). However, by the use of *kernels*, all necessary computations are only performed in the input space. On the other hand, Learn1NN-based NN classifiers places a series of OH’s in the input space. It is possible that large margin NN classifiers also make use of *kernels*. Since, a high-dimensional transformation increase the probability that the problem was linearly separable, kernel-based NN classifiers could employ then a smaller number of OH’s to solve the classification problem and then generalization could be enhaced.

4.3. Oriented Principal Component Analysis

In classification, observations belong to different classes. Based on some prior knowledge about the problem and on the training set, a pattern recogniser is constructed for assigning future observations to one of the existing classes. The typical method of building this system consists in dividing it into two main blocks that are usually trained separately: the feature extractor and the classifier.

The feature extractor was often handcrafted since it is rather specific to the problem. However, the current tendency is rely more on learning devices that *automatically extract features* and less on manual feature extraction of discriminatory information. Unsupervised learning algorithms (e.g. principal component analysis, PCA) are commonly used to build feature extractors from training data. However the application of these algorithms can lead to lose important discriminatory information since they take no account of the class label information.

An alternative and powerful implementation is to *integrate the feature extractor into the classifier* and to perform a global training of both systems to alleviate the problem of separate and uncoupled training. The thesis proposes a novel method (called *oriented principal component analysis*, OPCA) to perform a *global gradient-based training* of a feature extractor that uses several lineal combinations of input variables and any classifier that allow a back-propagation of an error signal through its architecture (e.g. feed-forward networks).

OPCA allows that the pattern recogniser project input data to a feature space of lower dimension that maximises the separation between classes. Figures 43 and 44 shows two synthetic problems in which OPCA finds the linear projection that achieves maximal separation between classes. Note that learning the linear projection that achieves the maximal classification information is a problem of learning a metric that takes into account class information in the context of nearest-neighbour classifiers (see figure 45).

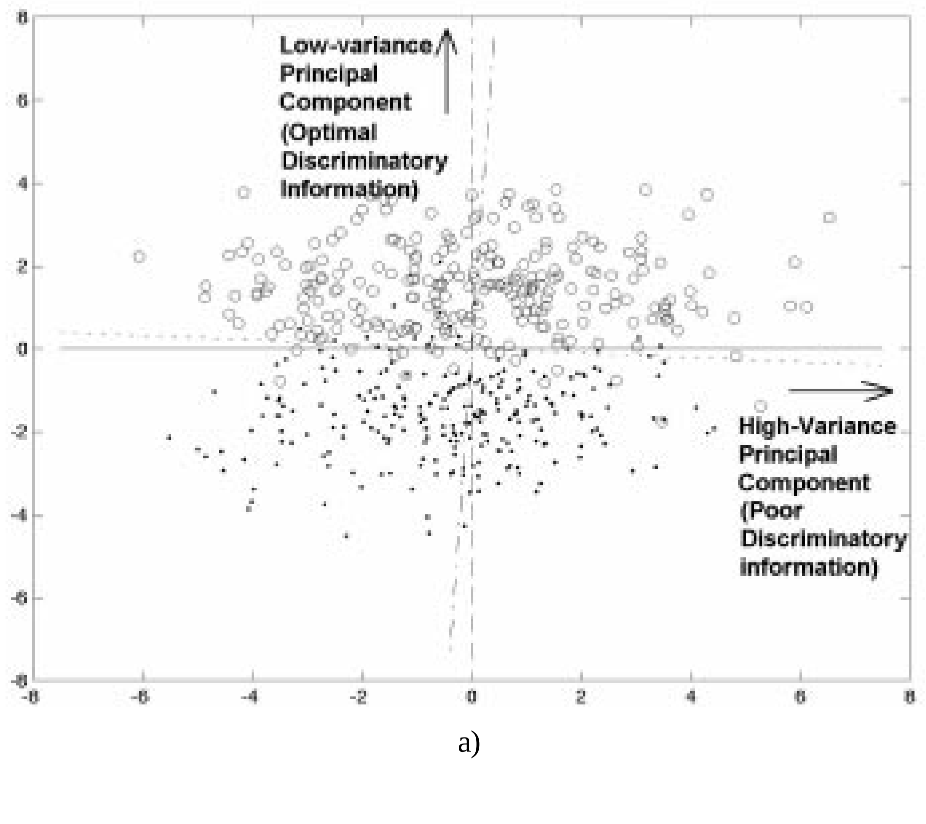
OPCA is based on a global and iterative training of itself and the classifier that discriminates in the OPCA's feature space, instead of the usual separate training of the feature extractor and the classifier. In this framework, the classifier regularises the principal component extractor in order to achieve a better separation of classes in the feature space. Hence, the supervised and unsupervised learning systems co-operate in order to find a global solution to the classification problem. The classifier repeatedly learns to form class boundaries in a feature space that is progressively transformed according to the information given by the misclassification error of the classifier in order to better separate classes.

As we pointed out before, our goal here is to obtain a linear transformation **U** of the input space based on principal component analysis but useful for classification. The simplest way of modifying the original PCA to incorporate class information is by adding a penalty term $\mathrm{L}_{\text{classifier}}(\Psi, \mathbf{U})$ on the original cost function in the following way:

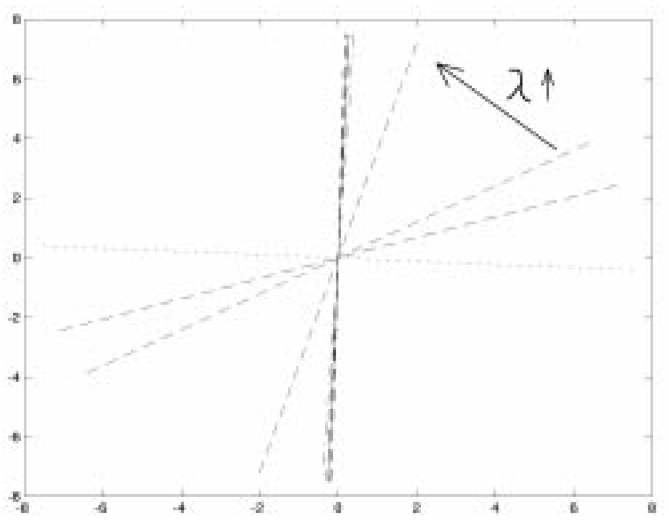
$$\mathrm{L}_{\text{OPCA}}(\mathbf{U}) = \mathrm{L}_{\text{PCA}}(\mathbf{U}) + \lambda\, \mathrm{L}_{\text{classifier}}(\Psi, \mathbf{U}) \tag{30}$$

Here $L_{PCA}(\mathbf{U})$ is the cost function defined of PCA, $L_{classifier}(\Psi,\mathbf{U})$ denotes any function of the misclassification error of a classifier Ψ that works in the feature space $\mathbf{Z}=\mathbf{U}^T\mathbf{X}$ and the parameter λ controls the degree to which the penalty term influences the solution. The resulting projections from minimising $L_{OPCA}(\mathbf{U})$ are a compromise between achieving PCs and those projections useful for discriminating in the feature space. If λ is small, the projections will be close to PCs. As λ grows, they will be oriented from PCs to those projections that cause that the classifier that discriminates in the feature space makes a small number of misclassifications.

In order to provide correct information about how the projections are useful to classification, we must re-train the classifier Ψ that works in the feature space, each time we modify the linear transformation **U**. In this way, the regularisation process to obtain the oriented PCs (OPCs) might form part of a global training process in which the feature extractor and the classifier are simultaneously trained in a co-operative manner. See chapter 10 for more details.

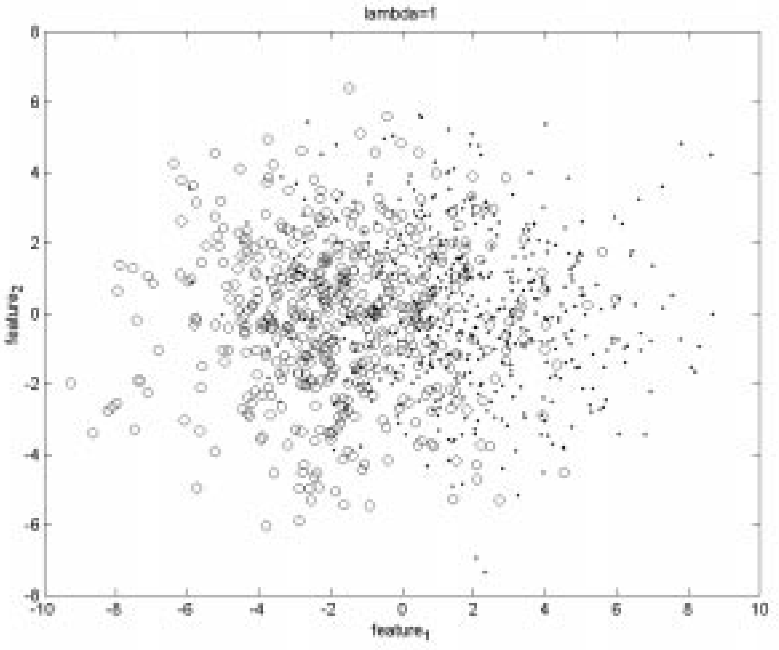


a)

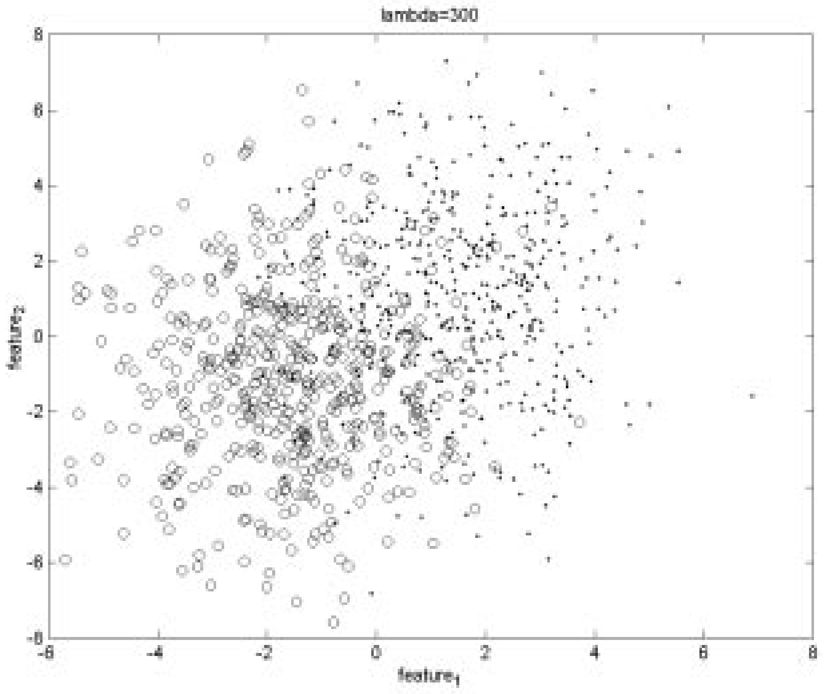


b)

Fig.43. The 2-D problem: a) data samples from the 2-D Problem and its principal components (PCs). We show data samples from class 1 (labelled ‘O’) and class 2 (‘.’), the optimal PCs- the dashed (--) and solid (-) lines- and the sample PCs- the dotted (.) and dashdot (-.) lines- that are estimated from training data. Observe that due to finite sample size the estimated PCs are slightly rotated versions of PCs. Typical use of PCA involves the extraction of high-variance PC. In this case, its use implies an important loss of discriminatory information; b) the Oriented PCs and sample PCs for the 2-D problem. We show the sample PCs (estimated from the sample covariance matrix computed with a training set)- the dotted (.) and dashdot (-.) lines-, the oriented PCs -the dashed (--) lines- for values of λ=1, ..., 90 (the estimated optimal value). As λ increases, the Oriented PC goes from the estimated PC of highest variance to the estimated PC of lowest variance.

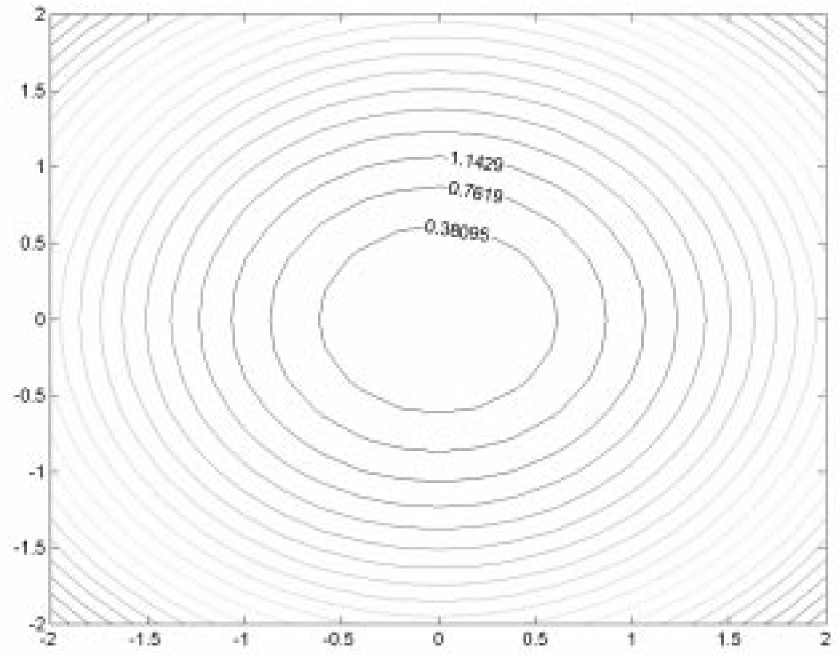


a)

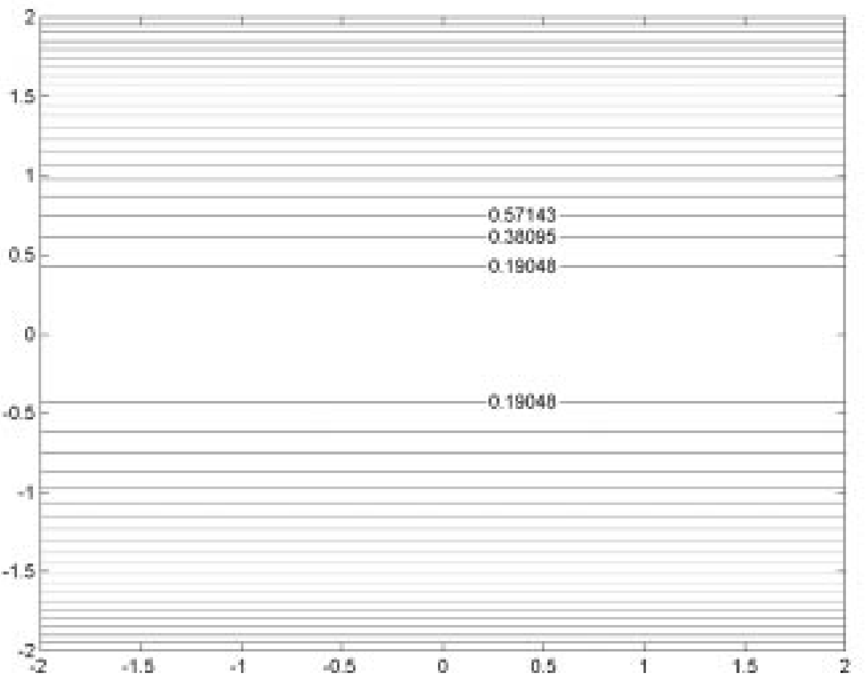


b)

Fig.44. 2-D Feature Space for the 3-D problem using OPCA for lambda=1.0 (a) and 300.0 (b). Observe that the second solution (OPCA) achieve a better separation between classes in feature space than the first solution (PCA).



a)



b)

Fig.45. Euclidean (a) and OPCA-based (b) metrics of a nearest-neighbour classifier for the synthetic 2-D problem (figure 43). Note that OPCA-based metric only takes into account discriminatory information to measure distances.

References

Aha, D. W. (Ed.) (1997). Special Issue on Lazy learning, Artificial Intelligence Review, 1-5.

Alpaydin, E. (1997). Voting over Multiple Condensed Nearest Neighbors, Artificial Intelligence Review, 1-5, p.115-132.

Atkeson, C. G., Moore, A. W. & Schaal, S. (1997). Locally Weighted Learning, Artificial Intelligence Review, 1-5, p.11-73.

Arbib, M.A. (Ed.) (1995). Handbook of Brain Theory and Neural Networks, Boston, MA:MIT Press.

Barlett, P. L. (1998). The Sample Complexity of Pattern Classification with Neural Networks: The Size of the Weights is More Important than the Size of the Network, IEEE Transaction on Information Theory, 44, 525-536.

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

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

Bishop, C. M. (Ed.) (1998). Neural Networks and Machine Learning, NATO ASI Series, Series F: Computer and Systems Sciences, Vol. 168, Berlin: Springer-Verlag.

Bottou, L.& Vapnik, V. (1992). Local Learning Algorithms, Neural Computation, 4, 888-890.

Bottou, L. & Bengio, Y. (1995). Convergence Properties of K-means, Advances in Neural Processing Systems 8. Boston, MA: MIT Press.

Breiman, L., Friedman, J.H., Olshen, R. A. & Stone, C. J. (1984). Classification and Regression Trees, New York: Chapman & Hall.

Breiman, L. (1997). The Heuristics of Instability in Model Selection, Annals of Statistics, 24, 2350-2381.

Breiman, L. (1998). Bias-Variance, Regularization, Instability and Stabilization, in (Bishop, 1998).

Cleveland, W. S. & Loader, C. (1995). Smoothing by Local Regression: Principles and Methods, Technical Report (40 pages), Murray Hill, NJ: AT&T Bell Laboratories.

Cherkassky,V., Friedman, J.H.,& Wechsler, H. (Eds.) (1994). From Statistics to Neural Networks, Theory and Pattern Recognition Applications, Berlin: Springer-Verlag.

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.

Dennis Jr., J.E. & Schnabel, R.B. (1989). A View of Unconstrained Optimization in Nemhauser, G.L., Rinnooy Kan, A.H.G. & Todd, M.J. (Eds.) Optimization, Amsterdam: North-Holland.

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

Dietterich, T. (1997). Machine Learning Research: Four Current Directions. AI Magazine, 18, 97-136.

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

Duda, R.O., Hart, P.E., Stork, D. G. (1996). Pattern Classification and Scene Analysis, 2nd Edition, New York:Wiley-Interscience.

Efron, B. & Tibshirani, R.J. (1994). An Introduction to the Bootstrap. London: Chapman & Hall.

Fan, J. & Gijbels, I. (1996). Local Polynomial Modelling and Its Applications, London: Chapman & Hall.

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

Friedman, J. H. (1996). On Bias, Variance, 0/1- loss, and the Curse-of-Dimensionality, Technical Report, Stanford, CA: Stanford University, Department of Statistics and Stanford Linear Accelerator Center.

Fukunaga, K. (1980). Introduction to Statistical Pattern Recognition, Boston, MA: Academic Press.

Geman, S., Bienenstock, E. & Doursat, R. (1992). Neural Networks and the Bias/Variance Dilemma, Neural Computation, 4, 1-58.

Gersho, A. & Gray, R.M. (1992). Vector Quantization and Signal Compression, Boston, MA: Kluwer Academic Publishers.

Hand, D. J. (1981). Discrimination and Classification, Chichester: Wiley.

Hastie, T. & Loader, C. (1993). Local regression: Automatic kernel carpentry, Statistical Science, 8, 120-143.

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.

Hertz, J., Krogh, J.A. & Palmer, R.G. (1991). Introduction to the Theory of Neural Computation, Reading, MA: Addison-Wesley.

Jolliffe, I.T. (1986). Principal component analysis. New York: Springer-Verlag.

Krogh, A. & Hertz, J. A. (1992). A simple weight decay can improve generalization, in Moody, J.E., Hanson, S. J. & Lippmann, R. P. (Eds.) Advances in Neural Information Processing Systems 4, San Mateo, CA: Morgan Kauffman Publishers.

Kosko, B. (1992). Neural Networks and Fuzzy Systems, Englewood Cliffs, NJ: Prentice-Hall.

Kulkarni, S., Lugosi, G. & Venkatesh, S. (1998). Learning Pattern Classification-A Survey. 1948-1998 Special Commemorative Issue of IEEE Transactions on Information Theory, 44, 2178-2206.

Kung, S. Y. (1993). Digital Neural Networks, Englewood Cliffs, NJ: Prentice-Hall.

Lam, L. & Suen, C. Y. (1995). An Evaluation of Parallel Thinning Algorithms for Character Recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence, 17.

LaVigna, A. (1990). Nonparametric classification using learning vector quantization. Ph. D. Dissertation, University of Maryland.

LeCun, Y. & Bengio, Y. (1995). Pattern Recognition and Neural Networks, in (Arbib, 1995).

Lugosi, G. & Nobel, A. (1996). Consistency of data-driven histogram methods for density entimation and classification. Annals of Statistics, 24, 687-706.

McQueen, J. (1967). Some Methods for Classification and Analysis of Multivariate Observations, in: Proc. of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability, 1, 281-296.

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

Minsky, M. (1961). Steps Toward Artificial Intelligence, Proceedings of the IRE.

Mitchell, T. (1995). Machine Learning, Boston, MA: Addison-Wesley.

Moody, J.E. (1992). The effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Systems, in Moody, J.E., Hanson, S.J. & Lippmann, R. P. (Eds.) Advances in Neural Processing Systems. Boston, MA: MIT Press.

Nilsson, Nils J. (1965, 1990). The Mathematical Foundations of Learning Machines, Boston, MA: Morgan Kaufmann.

Niyogi, P. & Girosi, F. (1994). On the Relationship Between Generalization Error, Hypothesis Complexity, and Sample Complexity for Radial Basis Functions, A.I. Memo No. 1467, Boston, MA: Massachusetts.

Perrone, Michael P. (1995). Averaging/Modular Techniques for Neural Networks" in (Arbib, 1995)

Prechelt, P. (1998). Early Stopping- but when? In Neural Networks: Tricks of the trade, p.55-69, Lecture Notes in Computer Science 1524, Heidelberg: Springer Verlag.

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.

Sharkey, A. (Ed.) (1999). Combining Artificial Neural Nets, Berlin: Springer-Verlag.

Skalak, D. B. (1997). Prototype Selection for Composite Nearest Neighbor Classifiers, Department of Computer Science, Amherst, MA: University of Massachusetts.

Smola, A. (1998). Learning with Kernels, Ph.D. Dissertation, Berlin: GMD.

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.

Stanfill, C. & Waltz, D. (1986). Toward memory-based reasoning. Communications of the ACM, 29, 1213-1228.

Tibshirani, R. (1996). Bias, variance and prediction error for classification rules, Department of Statistics, University of Toronto.

Vapnik, V. (1982). Estimation of Dependences Based on Empirical Data, Berlin: Springer-Verlag.

Vapnik, V., Levin, E. & LeCun, Y. (1994). Measuring the VC-dimension of a Learning Machine, To appear in Neural Computation, 1994.

Vapnik, V. (1995a). The Nature of Statistical Learning Theory, Berlin: Springer-Verlag.

Vapnik, V. (1995b). Learning and Generalization: Theoretical Bounds in (Arbib, 1995).

Vapnik, V. (1998). Statistical Learning Theory, New York: Wiley-Interscience.

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

Wolpert, D. H. (Ed.) (1995). The Mathemathics of Generalization, Reading, MA: Addison Wesley.