9 Adaptive Soft K-Nearest-Neighbour Classifiers with Large Margin

Abstract- A novel classifier is introduced to overcome the limitations of the K-NN classification systems. It estimates the posterior class probabilities using a local Parzen window estimation with the K-nearest-neighbour prototypes (in the Euclidean sense) to the pattern to classify. A learning algorithm to reduce the number of prototypes (that maximises the confidence or margin on the correct classification of training patterns) is also presented. Experimental results in two hand-written classification problems demonstrate the potential of the proposed classification system.

Index Terms- Soft Nearest Neighbour Classifiers, Online gradient descent, Hand-written Character Recognition.

1. Introduction

Nearest neighbor (NN) techniques are simple but powerful non-parametric classification systems. Since their introduction in the fifties, many sophistication of the basic schema appeared, concerned with different topics as condensing, editing, learning or extensions to include rejection. (See (Dasarathy, 1991) to review much of the existing literature.) In recent years, interest in these methods has flourished again in several fields (including statistics, machine learning and pattern recognition) since they remain the best choice in many classification problems.

This chapter presents a novel classification learning architecture in the context of condensed K-NN classifiers with the goal of enhancing its generalization properties. In learning systems, generalization performance is affected by a trade-off between the number of training examples and the capacity (e.g. the number of parameters) of the learning machine. This global trade-off can be reinterpreted in local learning systems (like NN methods) as a trade-off between capacity and locality (how local the solution is) (Bottou & Vapnik, 1992). One of the implications of this is that the learning system must control the effective number of training samples available for training locally the system. K-NN techniques are good examples of such 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. 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. However, an important disadvantage of the method, inherited from their ancestors, would be that all of the training data must be retained to compute the estimations. So 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 condensed set of prototypes from training data.

The organization of this chapter is as follows. Section 2 derives the soft K-NN classifier in the context of local learning algorithms (Bottou & Vapnik, 1992). In Section 3 we introduce an adaptive version of the previous classification rule. The proposed learning algorithm designs a condensed set of prototypes that minimizes the empirical average number of misclassifications.

Section 4 shows experimental results that compares successfully the adaptive soft K-NN classifier with other algorithms using two NIST handwritten character databases (Garris, 1997). Finally, in Sections 5 and 6 some discussion and conclusion are given.

2. The soft K-NN classifier

Let $c:X\rightarrow\{1, ..., C\}$ be a maximum classifier that assigns a input vector $\mathbf{x}$ to one of the C existing classes. This kind of classifiers is defined in terms of a set of discriminant functions $d_l(\mathbf{x})$, l=1, ..., C. The classifier c then assigns $\mathbf{x}$ to class j if $d_j(\mathbf{x})>d_l(\mathbf{x})$ for all $l\neq j$. In the case of equally misclassification costs, c achieves a minimum expected number of misclassifications when d_l is the posterior probability of belonging to class l, $P(C_l\backslash\mathbf{x})$. Suppose we have N observations pairs $T=\{(\mathbf{x_i}, \mathbf{y_i})\}$ i=0, ..., N-1 where $\mathbf{x_i}$ is a random sample taken from input space X that belongs to one of the classes and $\mathbf{y_i}$ is an indicator variable. If $\mathbf{x_i}$ belongs to the nth class, the nth coefficient of $\mathbf{y_i}$ is equal to 1 and the other coefficients are equal to 0. To ensure c is a good classification procedure each discriminant functions $d_l(\mathbf{x})$ must estimate the binary random variable y_l (that is equal to 1 if $\mathbf{x}$ belongs to the lth class and 0 otherwise). In local algorithms this estimation is defined for each neighborhood around the testing pattern $\mathbf{x}$ as those which minimizes a weighted empirical average loss function of the following form (Bottou & Vapnik, 1992) p.892:

$$\begin{aligned} I_{\text{local emp}}[d_l] &= \frac{1}{M}\sum_{i=1}^{M} G(\mathbf{x}_i - \mathbf{x};\gamma) J(y_{li}, d_l(\mathbf{x}_i;\gamma)) \\ d_l^*(\mathbf{x};\gamma) &= \arg\min I_{\text{local emp}}[d_l] \end{aligned} \tag{1}$$

where kernel G is a bounded and even function on X that is peaked about $\mathbf{0}$, γ denotes the locality control parameters associated to G and $\{d_l^*(\mathbf{x};\gamma), l=1,..,C\}$ are the optimal discriminant functions. If we use a constant approximation d_l (with respect to $\mathbf{x}$) of y_l and a quadratic loss $J(y_l,d_l)=(y_l-d_l)^2$, solving the equation $\partial I_{\text{local emp}}[d_l]/\partial d_l = 0$ l=1,...,C yields

$$d_l^*(\mathbf{x};\gamma) = \sum_{i=1}^{M} G(\mathbf{x}_i - \mathbf{x};\gamma) y_{li} \Big/ \sum_{i=1}^{M} G(\mathbf{x}_i - \mathbf{x};\gamma) \tag{2}$$

As we see from inspecting (2), this optimal local discriminants functions computes kernel-based estimates of the posterior probability of the classes (Bishop, 1995) §2.5. The shape of the kernel

G controlled by γ then determines different well-known solutions to the same original problem. If $G=G_H$ where G_H is a square kernel whose width is adjusted to contain exactly k samples then equation (2) is the *K-NN* algorithm. On the other hand if $G=G_S$ where G_S is a smooth kernel with a width modulated by a locality parameter σ then (2) is the *Parzen window* algorithm.

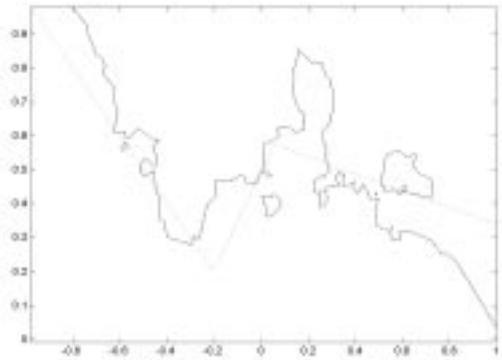
The K-NN technique considers a hypershere centered at the testing pattern **x** that exactly contains K training samples. This feature prevents the estimator from not having enough training data to ensure good generalization. Although in practice, their estimates poorly approximate posterior probabilities, since the region around **x** is not small enough to keep the estimations, a ratio of integers (K_l/K), valid. By contrast, the Parzen window algorithm can estimate better since it is based on a smooth interpolation between training points, although it does not effectively control the tradeoff between the capacity of the learning device and the number of training samples. This is due to using a fixed width parameter σ for all the training points. If the distribution of the training patterns in the input space is uneven, it will be impossible to ensure that there are enough training data that significantly contribute to the estimations for all testing pattern **x**.

We propose an approach that benefits from the virtues of these two techniques and can be understood as a compromise between them: for each testing pattern **x**, a Parzen window estimate is computed using the K nearest (in the Euclidean sense) training points. In fact, this algorithm results of using a composed kernel $G_{soft}=G_H G_S$ in equation (2). As we show in the Appendix, the resulting equation also estimates the posterior class probabilities. Equation (2) is then called the *soft K-NN* algorithm and can be written as follows

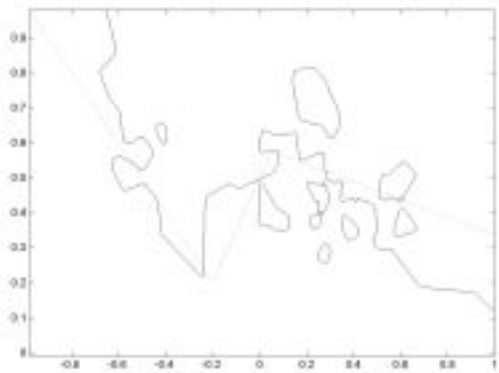
$$d_l^*(\mathbf{x};\gamma)=\sum_{i=1}^{K_l} G\left(\mathbf{x}_i^{\,l}-\mathbf{x};\gamma\right)\Big/\sum_{j=1}^{C}\sum_{u=1}^{K_j} G\left(\mathbf{x}_u^{\,j}-\mathbf{x};\gamma\right) \tag{3}$$

where $\mathbf{C}_{\text{K-NN of class } j=}\left\{\mathbf{x}_u^j,\ u=1,\ldots,K_j\right\}$ are those prototypes 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 (3) is recovered in the limit $\sigma\rightarrow\infty$. In the particular case k=2, the resulting classifier is equivalent to the 1-nearest-neighbor classification rule since only two prototypes are involved in the decision.

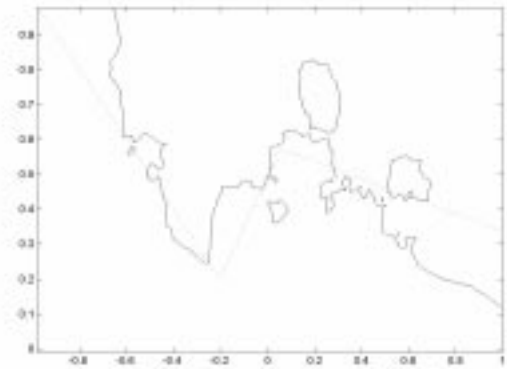
Figure 1 shows the classification borders for the Ripley's synthetic problem (Ripley, 1994) in the case of the soft K-NN and crisp K-NN ($\sigma\rightarrow\infty$) rules. Note that the soft K-NN algorithm can smooth the crisp classification border where σ controls the degree of smoothness.



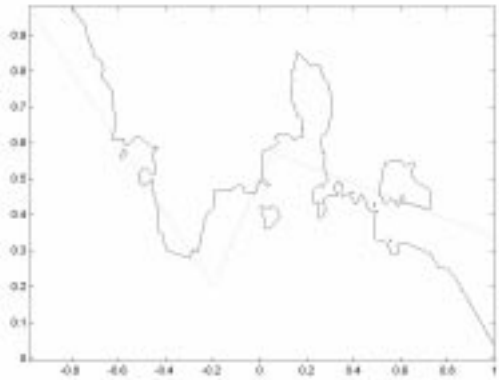
a) K=3, σ→∞



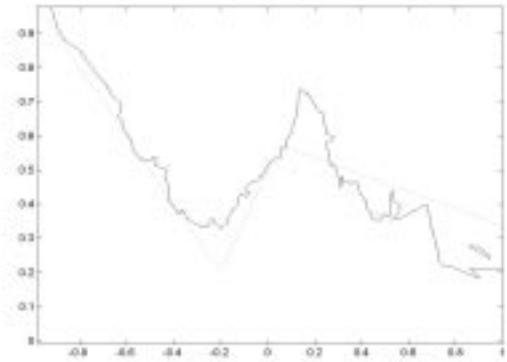
b) K=3, σ=0.001



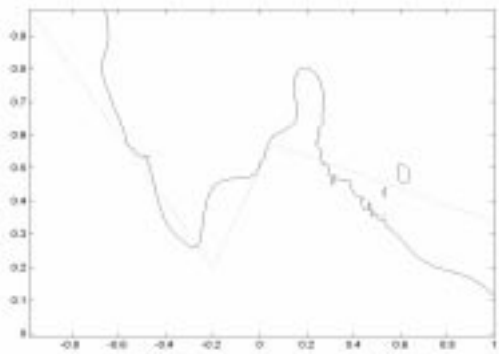
c) K=3, σ=0.1



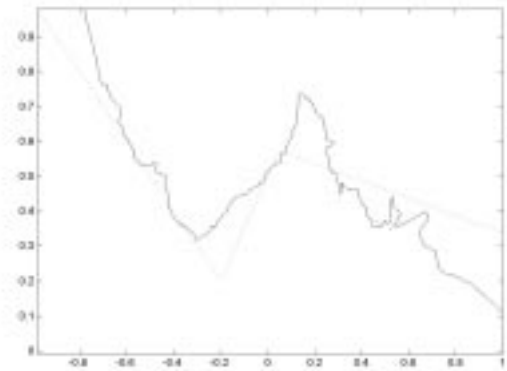
d) K=3, σ=0.05



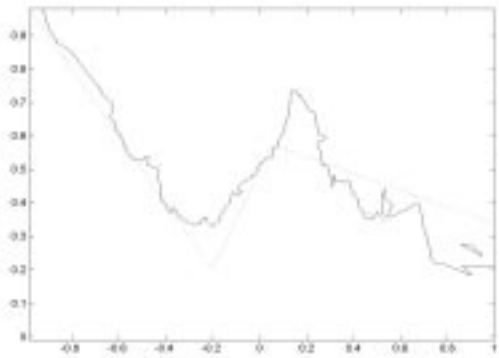
e) K=11, σ→∞



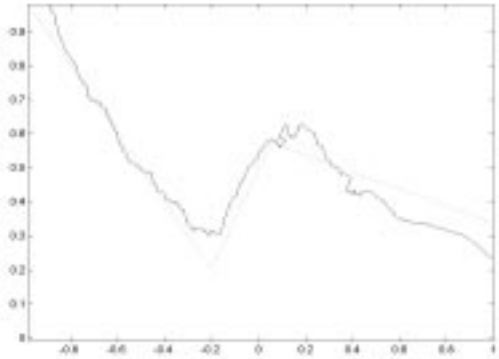
f) K=11, σ=0.01



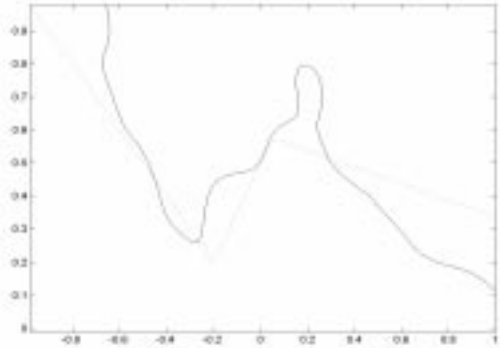
g) K=11, σ=0.05



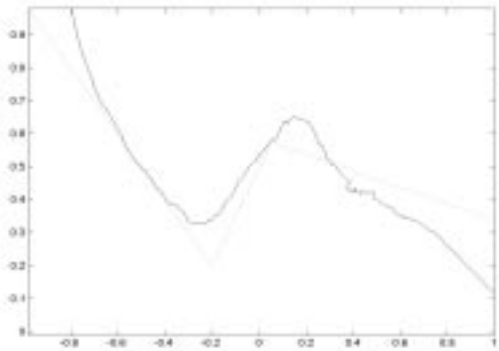
h) K=11, σ=5.0



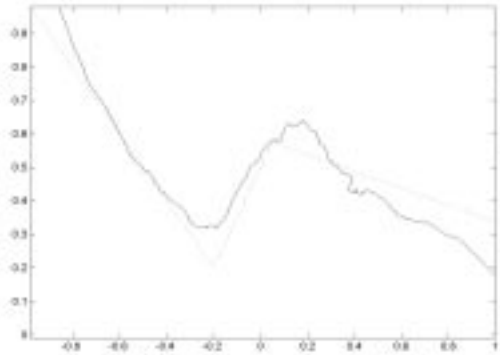
h) K=33, $\sigma \rightarrow \infty$



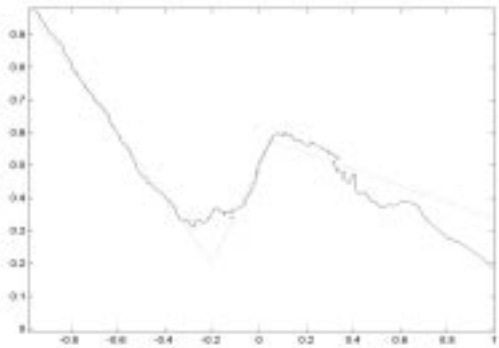
i) K=33, σ=0.01



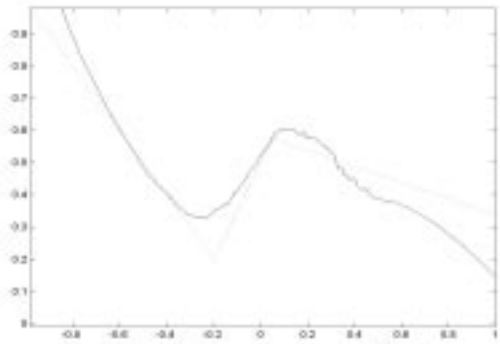
j) K=33, σ=0.05



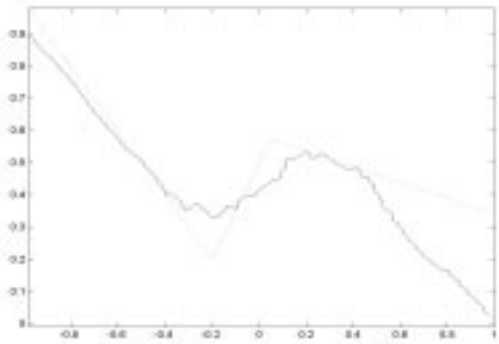
k) K=33, σ=0.1



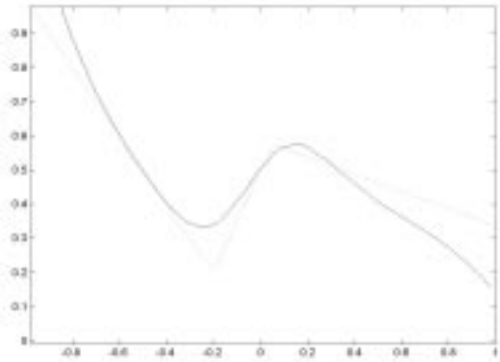
l) K=55, $\sigma \rightarrow \infty$



m) K=55, σ=0.075



n) K=99, $\sigma \rightarrow \infty$



o) K=99, σ=0.075

Fig.1. Soft & crisp K-nearest-neighbor classification borders in the Ripley's synthetic problem for several values of sigma and K. (The soft version uses gaussian kernels)

3. The adaptive soft K-NN classifier

Given a training set **T**, we can consider to apply (2) using G_{soft}, classifying a new input to those class whose discriminant function has the highest value. However when training set size is large, storage and processing requirements make this simple algorithm unusable in engineering applications. In the case of (crisp) K-NN classifiers, many heuristic sophistication of the basic algorithm allow to use fewer data points (e.g. Condensed NN (Hart, 1968)). By contrast, our approach can simply be extended by replacing the *local kernel estimation* with a *simplified adaptive mixture model* (McLachlan & Basford, 1988). Suppose we define for each class density function a mixture as follows

$$f_{X|C_l}(\mathbf{x})P(C_l) = \frac{1}{M}\sum_{j=1}^{M} f_{X|C_l,j}(\mathbf{x}) \tag{4}$$

where $\{f_{X\backslash Cl}, l=1,...,C\}$ are the class density probabilities, $\{P(C_l), l=1,...,C\}$ are the priors and $\{f_{X\backslash Cl,i}\ i=1...M,\ l=1,...,C\}$ are known parametric models restricted to the form $G(\mathbf{x}-\mathbf{w}_i^l;\gamma)$ with $G(\mathbf{u})$ a decreasing function on U and peaked about $\mathbf{u}=\mathbf{0}$. Applying Bayes' theorem, we arrive at the following expression for the posterior probabilities

$$P(C_l|\mathbf{x}) = \frac{\sum_{j=1}^{M} f_{X|C_l,j}(\mathbf{x})}{\sum_{i=1}^{C}\sum_{u=1}^{M} f_{X|C_i,u}(\mathbf{x})} \tag{5}$$

Since each model of the mixture is locally defined, we can approximate equation (5) using only the K-nearest-models to the testing pattern **x**. If we use this simplification to construct the classifier, then the discriminant functions give

$$P(C_l|\mathbf{x}) \approx \frac{\sum_{j=1}^{K_l} f_{X|C_l,j}{}^{K}(\mathbf{x})}{\sum_{i=1}^{C}\sum_{u=1}^{K_i} f_{X|C_i,u}{}^{K}(\mathbf{x})} \tag{6}$$

where $\left\{f_{X|C_i,u}{}^{K}(\mathbf{x}), i=1,...,C, u=1,...,K_i\right\}$ are the K-nearest-models to **x**. Like the mixture models are locally defined around its centers $\mathbf{w}_i^j$, $P(C_l\backslash\mathbf{x})$ then can be approximated by

$$P(C_l|\mathbf{x}) \approx \frac{\sum_{i=1}^{K_l} G\left(\mathbf{w}_i^{\,l} - \mathbf{x};\gamma\right)}{\sum_{j=1}^{C}\sum_{u=1}^{K_j} G\left(\mathbf{w}_u^{\,j} - \mathbf{x};\gamma\right)} \tag{7}$$

where $\{\mathbf{w}_u^j, u = 1,...,K_j, j = 1,...,C\}$ are the K_j-nearest-mixture-centers to $\mathbf{x}$ and $K = \sum_{j=1}^{C} K_j$.

The total number of mixture models (M*C) will be typically much smaller than the number of training points (N), and the mixture models centers ($\mathbf{w}_i^j$) are no longer constrained to coincide with the training points. The parameters of the mixture models could be estimated using unsupervised procedures as the EM algorithm (Dempster, 1977), although the followed approach here make use of a supervised learning procedure based on minimizing the number of misclassifications.

3.1. The Loss function.

Consider a training set of random pairs T={($\mathbf{x}_i$, $\mathbf{y}_i$)} i=0, ..., N-1. If the pattern $\mathbf{x}_i$ belongs to the nth class, the nth coefficient of $\mathbf{y}_i$ is equal to 1 and the other coefficients are equal to 0. As a loss function we propose a modification* of the cross-entropy error function for multiple classes (Bishop, 1995) §6.9:

$$L = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{l=1}^{C} y_{li}\, d_l(\mathbf{x}_i) \tag{8}$$

The absolute minimum with respect to the {$d_l(\mathbf{x}_i)$} occurs when all the training points are **correctly classified with the highest confidence** (or maximum margin) , that is when $d_l(\mathbf{x}_i)=y_{li}$ for all l and i. In the limit in which the size N of the training set goes to infinity, we can substitute the sum over patters with the following integral

$$\langle L\rangle = \lim_{N\to\infty} L = -\int_{\Re^d}\sum_{l=1}^{C} y_l(\mathbf{x}) d_l(\mathbf{x}) f_X(\mathbf{x})\, d\mathbf{x} \tag{9}$$

where f_X is the probability density of the input space. Since y_l=**1**($\mathbf{x}\in$class l) where **1** is the indicator function

* We will see in the next subsection, when we derive the learning equations, why equation (8) can be of more practical interest than cross-entropy error.

$$1(condition) = \begin{cases} 1 & if\ \ condition\ is\ true \\ 0 & otherwise \end{cases} \tag{10}$$

and $f_X(\mathbf{x}) = \sum_{i=1}^{C} f_{X|C_i}(\mathbf{x}) P(C_i)$, equation (9) can be easily rewritten as

$$\langle L \rangle = -\sum_{l=1}^{C} \left(\int_{R_l} d_l(\mathbf{x}) f_{X|C_l}(\mathbf{x}) P(C_l)\, d\mathbf{x} + \int_{R_l} \sum_{\substack{i=1 \\ i \neq l}}^{C} d_i(\mathbf{x}) f_{X|C_i}(\mathbf{x}) P(C_i)\, d\mathbf{x} \right) \tag{11}$$

where R_l are the decision region of class l, that is the input region in which d_l has the highest value among discriminant functions. <L> would be minimized when d_l=1 for all **x** in R_l and $f_{X\backslash Cl}P(C_l)$=max $f_{X\backslash Ci}P(C_i)$ for all R_l. But as we know the best possible choice for d_l is $P(C_l\backslash\mathbf{x})$. If we use an optimization method to minimize iteratively <L>, since the algorithm will try to minimize the number of misclassifications it will be possible a convergence of d_l to $P(C_l\backslash\mathbf{x})$ (at least near Bayes Borders). In this case if class overlapping is moderate, the maximum expected number of correct classifications could approximate the absolute value of <Lmin>,

$$\left| \langle L_{min} \rangle \right| \approx \sum_{l=1}^{C} \int_{R_l} \hat{P}(C_l|\mathbf{x}) f_{X|C_l}(\mathbf{x}) P(C_l)\, d\mathbf{x} \tag{12}$$

This latter result gives us, when N is large enough, a possible tendency of the system from minimizing equation (8) during the learning phase.

3.2. The learning algorithm.

In real-life pattern recognition problems training sets usually are large and high dimensional. So algorithmic simplicity of the learning procedure, or the optimization process, is a requirement in order to achieve a solution with moderated computational resources. Online gradient descent algorithms (Bottou, 1998) are one of the most simple computational approaches to the learning problem. Another features of these algorithms include better convergence speed than their batch counterparts in redundant training sets (e.g. real-world databases) and good mathematical characterization (Benveniste, 1990)(Bottou, 1998).

3.2.1. Derivation of the on-line gradient learning algorithm.

We restrict the mixture model parameters to centers and a global defined parameter σ that controls locality. So the goal in the learning phase is reduced to compute a set of labeled codebooks $C_{j=}\{\mathbf{w}_i^j, i=1,...,M_j\}$ j=1,...,C, one for each class, that minimizes an estimator of the expected loss function given in equation (9),

$$\langle L\rangle(\{C_j,\ j=1...C\}) = E_X\left[Q(\mathbf{x},\{C_j,\ j=1...C\})\right] \tag{13}$$

Since $E_X[\bullet]$ is unknown, an empirical estimator is formed with a (training) set of random samples pairs T={($\mathbf{x_i}$, class index($\mathbf{x_i}$))} i=0,...,N-1. The online gradient learning algorithm estimates using the instantaneous loss function $Q(\mathbf{x},\{C_j,\ j=1,...,C\})$. Each step of this algorithm consists of picking up (cyclically or randomly) one sample pair from the training set T and applying the following update equation

$$\mathbf{w}_i^j[k] = \mathbf{w}_i^j[k-1] - \eta_i^j[k]H(\mathbf{w}_i^j[k-1], \mathbf{x}[k]) \tag{14}$$

The learning rates $\eta_i^j[k]$ are positive numbers and are usually made to decrease monotonically with discrete time k, although they can be also constant with time. The term $H(\mathbf{w_i^j}, \mathbf{x})$ is defined as

$$H(\mathbf{w}_i^j, \mathbf{x}) = \begin{cases} \nabla_{\mathbf{w}_i^j} Q(\mathbf{x},\{C_j,\ j=1...C\}) & \text{when differentiable} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

Given a training set of infinite size, the necessary (but not sufficient, see (Bottou, 1998) §5) condition to ensure the convergence of the iterative equation (14) to a local minimum of <L> is that

$$E_X\left[H(\mathbf{w}_i^j, \mathbf{x})\right] = \nabla_{\mathbf{w}_i^j}\langle L\rangle(\{C_j,\ j=1...C\}) \tag{16}$$

Since Q is not differentiable on a finite number of points and the iterative algorithm has a zero probability to reach them, this condition is met (see (Bottou, 1998) p.10). In the case of using a constant η, we might ensure that η was small enough (see (Benveniste et al., 1990) p.48). Practically, an optimal value of η can be estimated using a validation set.

Let $K(\mathbf{x}-\mathbf{w}_i^j)$ be $\exp(\frac{-\|\mathbf{x}-\mathbf{w}_i^j\|^2}{\sigma})$, $\langle L\rangle' = \sigma/2\langle L\rangle$ and $\eta_i^j[k]=\eta$. We have modified the loss function for analytical convenience since the added constant does not affect the desired points of convergence. Hence equation (15) yields

$$H(\mathbf{w}_i^j,\mathbf{x})=\begin{cases}-d_W(\mathbf{x})(1-d_j(\mathbf{x}))(\mathbf{x}-\mathbf{w}_i^j) & \text{if } \mathbf{w}_i^j\in C_{k-NN} \text{ and } j=\text{class index}(\mathbf{x})\\ d_W(\mathbf{x})\,d_j(\mathbf{x})(\mathbf{x}-\mathbf{w}_i^j) & \text{if } \mathbf{w}_i^j\in C_{k-NN} \text{ and } j\neq\text{class index}(\mathbf{x})\\ 0 & \text{otherwise}\end{cases} \tag{17}$$

where $\mathbf{C}_{K\text{-}NN}$ are those prototypes of class j among the k nearest prototypes (in the Euclidean sense) to $\mathbf{x}$ and d_W is equal to

$$d_W(\mathbf{x})=\frac{\exp\left(-\left\|\mathbf{x}-\mathbf{w}_i^{\,j}\right\|^2/\sigma\right)}{\sum_{m=1}^{C}\sum_{u=1}^{K_m}\exp\left(-\left\|\mathbf{x}-\mathbf{w}_u^{\,m}\right\|^2/\sigma\right)} \tag{18}$$

In the case of σ goes to infinite $\exp(\bullet/\sigma)$ tends to the unity. So H can be simplified since d_W tends to 1/K and d_j to K_j/K. Finally the learning equation gives

$$\begin{aligned}&\text{if } \mathbf{w}_i^j[k-1]\in C_{k-NN}(\mathbf{x}[k]) \text{ and } j=\text{class index}(\mathbf{x}[k])\\ &\quad \mathbf{w}_i^j[k]=\mathbf{w}_i^j[k-1]+\eta\, d_W(\mathbf{x}[k])(1-d_j(\mathbf{x}[k]))(\mathbf{x}[k]-\mathbf{w}_i^j[k-1])\\ &\text{if } \mathbf{w}_i^j[k-1]\in C_{k-NN}(\mathbf{x}[k]) \text{ and } j\neq\text{class index}(\mathbf{x}[k])\\ &\quad \mathbf{w}_i^j[k]=\mathbf{w}_i^j[k-1]-\eta\, d_W(\mathbf{x}[k])\,d_j(\mathbf{x}[k])(\mathbf{x}[k]-\mathbf{w}_i^j[k-1])\\ &\text{otherwise}\\ &\quad \mathbf{w}_i^j[k]=\mathbf{w}_i^j[k-1]\end{aligned} \tag{19}$$

where $\mathbf{x}[k]=\mathbf{x}_{(k-1)\bmod N}$ $\quad k\geq 0$ if we pick up cyclically.

3.2.2. Properties of the learning equation.

As we pointed out before, the proposed loss function has the same minimum points than the cross-entropy error function

$$L_{cross-entropy} = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{l=1}^{C} y_{li} \ln d_l(\mathbf{x}_i) \tag{20}$$

If we derive the on-line gradient equations for this error function, we obtain the equation (17) divided by $d_j(\mathbf{x})$. So practical differences between performing on-line gradient descent using one loss function or another must be determined with an analysis of how this term can affect to the evolution of the optimization process. Suppose that several outliers are present in the training set. This is a typical situation in real-world databases. When we pick up one outlier of class j, the amount of change in a winning prototype $\mathbf{w_i^j}$ will be

$$\Delta \mathbf{w}_{i\,cross-entropy}^{j} = d_W(\mathbf{x})\frac{1-d_j(\mathbf{x})}{d_j(\mathbf{x})}\left(\mathbf{x}-\mathbf{w}_i^j\right) \tag{21}$$

Since it is probable that $d_j(\mathbf{x})$ takes a low value when $\mathbf{x}$ is an outlier of class j, the term $(1-d_j(\mathbf{x}))/d_j(\mathbf{x})$ tends to infinite. In this way $\Delta\mathbf{w_i^j}_{cross\text{-}entropy}$ can be arbitrarily large and the normal evolution of the learning process can be enormously degraded by the presence of outliers. By contrast, if we use our proposed loss function, outliers have a little impact since the term d_w dominates the amount of change and tends to zero since the distance between the winning prototype and the outlier can be large.

4. Empirical Results

In this section, we compare Kohonen's LVQX algorithms (Kohonen, 1996) (where X is 1, 2 and 3) and K-NN classifiers with our proposal. We have used the NIST (uppercase & lowercase) handwritten characters database (Garris, 1997), which can be found in directories train/hsf_4, train/hsf_6 and train/hsf_7. We have split directory train/hsf_7 in two sets: one for validation (1100 samples) and one for test (over 10000 samples) preserving the original class distribution in data. The other two directories were chosen for training (above 24000 samples). These images (32x32 pixels) have been preprocessed with a Principal Component Analyzer that extracts 64 components from each image (NIST's mis2evt utility). The correlation matrix of the PCA was computed using the training set.

4.1. Simulations #1: Adaptive Soft K-NN vs. LVQ

Ten runs for each classification problem, learning algorithm and several codebook sizes have been made. We have initialized the codebooks using the LVQ_PAK's eveninit program

(Kohonen et al., 1995). Then we have applied every learning algorithm until classification error in validation set increases or stands. (We have monitored this error every 200000 iterations. In the case of LVQX algorithms, every time we monitored the error, we restart its α value). Optimal parameters of algorithms have been estimated using the validation set.

4.1.1. Results

Two main aspects are of interest in this section. The first one would be concerned with the comparison of our classifier and their crisp counterparts when our system is not constrained to use also the crisp K-NN estimation. The second one would compare the classifier when all of them make use of this classification rule. In this way we can evaluate if the use of a more complex classification procedure, the soft K-NN estimation, is of practical interest and, secondly, what happens when we force our system to compete with other well-established methods to design prototypes sets for crisp K-NN classifiers.

Tables 1 and 2 summarize the average classification error on uppercase and lowercase test sets. Each result is the average of ten trials for each learning algorithm (except in the case of K-NN classifiers that the test error is computed once using the whole training set as the prototypes set). In table 1, we show the average misclassification rate of our proposed classifier in four different cases. In the first case, we fix the number of nearest-prototypes (k) equal to 2 and σ to infinite. While in the second case with k fixed to 2, we make use of the best σ value estimated by cross-validation. In these two first cases, the resulting classification systems are crisp 1-NN classifiers since only two nearest-prototypes contribute to form the soft estimations. In the third case, we fix σ to infinite and make use of the best-estimated k value (with k greater than 2). As above, our classifier then is the crisp K-NN classifier. Finally in the fourth case, we constrain k to be greater than 2 and allow σ to take a finite value. Both parameters are estimated with the validation set using these constraints. Table 2 shows classification results of 1-NN classifiers whose codebooks are designed using Kohonen's LVQ algorithms and crisp K-NN that use the whole training database as the prototypes set.

The condensed soft K-NN classifiers (the fourth case) always outperforms crisp 1-NN classifiers based on LVQ algorithms for each tested codebook size. In uppercase and lowercase recognition, best results are achieved when codebook size is 800. In this case, our classification systems also outperform K-NN classifiers that use the whole training database as the prototypes set. On average, our learning system correctly recognized 94,62% of the uppercase handwritten letters and 87,46% of the lowercase handwritten characters, in front of 91,77% and 85,94% respectively for LVQ2, the best of the LVQ algorithms. Differences between our procedure and

LVQ algorithms are notably greater as codebook size decreases. We also note that the evolution of the misclassification rate, as codebook size grows, is less abrupt in our procedure. This behavior suggests that our procedure achieves a better graceful degradation in the classification accuracy as the codebook size is smaller.

Our best crisp 1-NN classifiers, the best solutions of the two first cases, clearly outperform LVQ-based crisp 1-NN classifiers except in one case. When the codebook size is equal to 200, codebooks designed with the LVQ2 algorithm are better than ours. It is interesting to compare the classification accuracy between case 1 and case 2 since both learning algorithms design codebooks for crisp 1-NN classifier although they differ in the way of doing it. While the first algorithm uses an infinite σ, the second one do not and can better define how local the solution is, since σ regulate how many training data contribute to update each prototype during the learning phase. In both classification problems we observe the same behavior. For small codebooks (100 and 200), the effect of using a finite σ allows to achieve an optimal solution. However, the use of an infinite σ works better for medium codebooks.

Classifiers of the third case only achieve better classification accuracy than K-NN classifiers (that use over 25000 prototypes) when their codebook size is 800. Although these classifiers are inferior to the best solutions achieved in the above crisp cases. Finally we compare best solutions of cases 1,2 and 4. We can observe that the soft estimation is in six out of eight cases superior than the crisp estimation. Only in lowercase hand-written recognition, the crisp estimation is superior on average when codebook sizes are 400 and 800.

Codebook Size	Upper				Lower			
	$\sigma\to\infty$ K=2	Best $\sigma<\infty$ K=2	$\sigma\to\infty$ Best K>2	Best $\sigma<\infty$ Best K>2	$\sigma\to\infty$ K=2	Best $\sigma<\infty$ K=2	$\sigma\to\infty$ Best K>2	Best $\sigma<\infty$ Best K>2
100	18,2	14,52	25,95(3)	**11,94**(5)	24,7	20,92	31,01(3)	**18,8** (11)
200	10,98	10,63	15,8(3)	**9,01**(5)	17,62	17,26	22,69(3)	**15,93**(5)
400	7,36	7,97	10,09(3)	**6,53**(5)	**13,3**	15,28	17,13(3)	14,02(5)
800	5,45	7,22	5,74(3)	**5,38**(11)	**11,28**	14,11	11,82(3)	12,54(5)

Table 1. Experimental Results for our proposed classifier. We show for each algorithm the average test error over ten runs. The number in parenthesis denotes the value of k.

Codebook Size	Upper				Lower			
	LVQ1	LVQ2	LVQ3	K-NN	LVQ1	LVQ2	LVQ3	K-NN
100	21,5	15,82	16,02	-	27,89	22,4	22,29	-
200	15,3	9,941	11,4	-	20,85	16,81	18,39	-
400	11,8	8,24	9,08	-	17,27	14,84	15,54	-
800	9,61	8,23	8,26	-	15,53	14,06	14,43	-
Whole database	-	-	-	5,84(1)	-	-	-	13,1(7)

Table 2. Experimental Results for LVQ-based and K-NN classifiers. We show for each algorithm the average test error over ten runs, except in the case of K-NN classification. In K-NN classifiers, the number in parenthesis denotes the value of k.

4.2. Simulations #2: The utility of Soft outputs for post-processing purposes

An additional feature the soft K-NN approach is that soft labels can be assigned to test patterns. This can enhance the overall classification accuracy when some post-processing is performed using context (e.g. relations of patterns in time). This section explores how the misclassification rate is improved when we accept that the correct solution was among the top M greatest activation of the discriminant functions. In this way, we can detect if a correct solution is among the top greatest activation and consequently a post-processor would recover the correct solution.

We use the best soft K-NN classifiers of section 4.2 for the upper problem and now we perform ten runs for each classifier checking the validation error every 50000 steps. Table 3 and 4 show the averaged validation and test error respectively using the top M highest activation. (The classifier will make a correct decision if the label of the training pattern agrees with the one of the labels of the M discriminant functions with highest activation.) Note that top 1 results in table 4 are slightly different than the fourth column of table 1 since now the classification error is checked every 50000 steps so the algorithm stops at different points. As table 3 and 4 show if a correct solution is between the top 2 highest activation the classification accuracy is reduced by more than a half (except in the case of 100 prototypes).

Top M	100	200	400	800
Top 1	11.2805	8.2913	6.4394	5.2925
Top 2	7.0670	4.6494	2.9465	2.1813
Top 3	5.5482	3.4134	1.9760	1.3812

Table 3. Averaged validation error over ten runs for the best soft K-NN classification in the upper problem. (Note: a classification error is produced if the label of the training pattern does not agree with the one of the labels of the M discriminant functions with highest activation.)

Top M	100	200	400	800
Top 1	11.4351	8.5248	6.6061	5.4136
Top 2	7.0222	4.6464	3.0445	2.2498
Top 3	5.5754	3.4656	2.1373	1.4601

Table 4. Averaged test error over ten runs for the best soft K-NN classification in the upper problem.

4.3. Simulations #3: Overtraining and Overfitting in adaptive soft K-NN classifiers

Overtraining (that is, the excessive tuning to the training set of the solution due to reaching a deep minimum of the cost function) can lead to solutions with poor generalization capabilities. If the number of free parameters of the classifier is excessive, overtraining leads to overfitting (the solution is too fitted to noisy samples). The usual way of avoiding them is to stop at a (local) minimum of the validation set. However this practice can be too conservative since the learning algorithm could converge to a local minimum of the training cost function far from the global minimum so the risk of overtraining would be low. (See e.g. multi-layer perceptrons trained with the back-propagation algorithm (Lawrence et al., 1997)).

In table 5 we show the results of the best soft K-NN classifiers for the upper problem over 50 runs when the learning algorithm stops at minimum of training and validation errors. Observe that classifiers achieve their best results when they stop at a minimum of the training error.

Stop at a minimum of the	Codebook size			
	100	200	400	800
Training error	10.1048 (1.30)	7.2448 (0.4689)	6.2897 (0.2336)	5.2886 (0.143)
Validation error	12.117 (1.4342)	8.7315 (0.837)	6.5257 (0.297)	5.366 (0.18)

Table 5. Averaged test error and variance over 50 runs for the best soft K-NN classification in the upper problem when the learning algorithm stop at a minimum of the training and validation errors.

5. Discussion

5.1. Soft K-NN vs. Crisp K-NN

The Soft K-NN algorithm uses the distance to test patterns for its estimation of posteriori class probabilities. This additional information enhances the approximation quality of the estimation since it introduces a smooth interpolation in relation with the crisp K-NN estimate. We have shown in figure 1 that the classification borders of the soft K-NN estimate are a smooth version of the crisp K-NN borders. As section 4.1 demonstrated, the resulting smoothed borders increase the classification rate for a hard decision at the expense of a little more computation. Besides the soft K-NN rule allows a fuzzy classification (Bezdek & Pal, 1992) where soft labels can be used for post-processing purposes. Section 4.3 show that if a post-processor detect the correct label among the two greatest outputs of the soft K-NN classifier the classification accuracy can be notably reduced (more than a half).

5.2. The Adaptive Soft K-NN algorithm vs. LVQ algorithms

The adaptive soft K-NN algorithm computes a reduced set of prototypes for the soft K-NN classification rule. The learning algorithm minimises a modification of the cross-entropy function whose absolute minimum points ensure the minimisation of the training classification error. However the learning rule resembles a fuzzified version of Kohonen's LVQ algorithm (Kohonen, 1996) so this fact can indicate some similarity between our algorithm and LVQ when hard decisions are employed ($\sigma \rightarrow \infty$) and K=2. Suppose that $\mathbf{m}_i[k]$ and $\mathbf{m}_j[k]$ are the two nearest prototypes to pattern $\mathbf{x}[k]$ then equation 19 yields

$$\text{if } \mathbf{m}_i[k-1], \mathbf{x}[k] \in \text{ same class and } \mathbf{m}_j[k-1], \mathbf{x}[k] \notin \text{ same class}$$

$$\begin{aligned} \mathbf{m}_i[k] &= \mathbf{m}_i[k-1] + \frac{\eta}{4}\left(\mathbf{x}[k] - \mathbf{m}_i[k-1]\right) \\ \mathbf{m}_j[k] &= \mathbf{m}_j[k-1] - \frac{\eta}{4}\left(\mathbf{x}[k] - \mathbf{m}_j[k-1]\right) \end{aligned} \tag{22}$$

Equation 21 is the LVQ2.1 algorithm when the training pattern **x**[k] falls in a window defined in LVQ2 and LVQ3 algorithms. This result is not a surprise since (Bottou, 1998) showed that the LVQ2 algorithm performs gradient descent over an instantaneous loss function based on a continuos approximation of the (binary) classification error function.

5.3. The loss function

The loss function that we use in the learning phase has interesting properties. It is less sensitive to outliers as section 3.2.2 shows (see (Bermejo, 2000) for an additional discussion). Besides, our error function is, for a two-class problem, the margin function of the classifier averaged over the whole training patterns. If K=2 for a two-class problem, the learning rule coincides with the learn1NN algorithm (Bermejo, 2000). So the computed prototypes uses the support vectors to build Optimal Margin (OM) hyperplanes (Schölkopf, 1997) as it was empirically shown in (Bermejo, 2000).

5.4. Overtraining and Overfitting

Experimental section 4.4 shows that if we stop at a local minimum of the training classification error instead of using the early stopping criterion, the test error is reduced. This result can suggest that overtraining is harder that expected (Lawrence, 1997). Since the learning algorithm can hardly achieve a global minimum the risk of overtraining is reduced so the use of a validation set makes the algorithm to stop too early. Furthermore, If overtraining is difficult then the risk of overfitting is notably reduced.

6. Conclusion

Crisp K-nearest-neighbour classifiers estimate posterior class probabilities using a fixed number of training data. This feature prevents them, unlike other approaches, for not having in some input regions enough data to ensure good generalization. Although the quality of their estimations (a simple ratio of integers) is too poor to properly approximate the true probabilities when finite data sets are employed.

An extension of K-NN estimation was presented in order to overcome this limitation: for each testing pattern **x**, a Parzen window estimate is computed using the K nearest (in the

Euclidean sense) training points. This simple algorithm also estimates the posterior class probabilities although all the training data points must be retained in order to compute the estimations.

An improvement of this approach can be achieved, in terms of storage and processing requirements, by replacing the local kernel estimation with a simplified adaptive mixture model: since each model of the mixture is locally defined, for each testing pattern we only take into account its K-nearest-models. The total number of mixture models will be typically much smaller than the number of training points and the mixture models centers are no longer constrained to coincide with the training points. We have proposed a supervised learning procedure to estimate the mixture models parameters based on minimizing a modification of the cross-entropy error function. This error function has the same minimum points that the cross-entropy error function while it allows the derived on-line gradient learning algorithm to have a more robust behavior in presence of outliers in the training set.

Experimental results in two handwritten classification problems demonstrate the potential and versatility of the proposed classification system. Since the soft K-NN classifier includes as a special case the crisp version, in numerical simulations we always find an optimal solution that achieves better classification accuracy than crisp 1-NN classifiers designed with LVQ algorithms and K-NN classifiers.

References

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

Bermejo, S. (2000). Large Margin Adaptive 1-Nearest Neighbour Classifiers, in Bermejo, S. Learning with Nearest Neighbour Classifiers, Ph.D. Thesis, Barcelona: Universitat Politècnica de Catalunya.

Bezdek, J.C & Pal, S.K. (Eds.) (1992). Fuzzy Models for Paterrn Recognition, NY: IEEE Press.

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

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

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

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

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

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

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

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

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

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

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

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

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

Appendix: The Soft K-NN Kernel Estimation

Consider a training set of random pairs T={($\mathbf{x_i}$, class label($\mathbf{x_i}$))} i=0, ..., N-1 where $\mathbf{x_i}$ is a random sample belonging to one of the C existing classes. Suppose we take the region R($\mathbf{x}$) to be a small hypersphere centered at the point $\mathbf{x}$ that contains precisely K training data points. Let us denote R as the event X belongs to R(X). This event is always true due to any vector belongs to a region centered at itself. So the following equation can be written

$$P(C_l|\mathbf{x}) = P(C_l|\mathbf{x}, R) \tag{A.1}$$

Making use of Bayes' theorem in the right hand-side of equation (A.1) we have

$$P(C_l|\mathbf{x}, R) = \frac{f_{X|C_l,R}(\mathbf{x})P(C_l|R)}{f_{X|R}(\mathbf{x})} \tag{A.2}$$

If the training data belonging to class l that fall in R($\mathbf{x}$) are denoted by {($\mathbf{x_i^l}$)} i=1...K_l l=1...C, we obtain the following Parzen window estimates (equation (2.57) (Bishop, 1995)) for the probability density functions $f_{X\backslash R}$ and $f_{X\backslash Cl,R}$

$$\begin{aligned} f_{X|C_l,R}(\mathbf{x}) &\approx \frac{1}{K_l}\sum_{i=1}^{K_l}\frac{1}{h^d}G\left(\frac{\mathbf{x}-\mathbf{x}_i^l}{h}\right) \\ f_{X|R}(\mathbf{x}) &\approx \frac{1}{K}\sum_{j=1}^{C}\sum_{m=1}^{K_j}\frac{1}{h^d}G\left(\frac{\mathbf{x}-\mathbf{x}_m^j}{h}\right) \end{aligned} \tag{A.3}$$

Since the probability of belonging to class l conditioned to belong to region R, $P(C_l \backslash R)$ can be approximated by K_l/K, we have as an estimation of $P(C_l \backslash \mathbf{x})$

$$P(C_l|\mathbf{x}) \approx \frac{\sum_{i=1}^{K_l} G\left(\frac{\mathbf{x}-\mathbf{x}_i^l}{h}\right)}{\sum_{j=1}^{C}\sum_{m=1}^{K_j} G\left(\frac{\mathbf{x}-\mathbf{x}_m^j}{h}\right)} \tag{A.4}$$