Chapter 4

Classification of On-Line Handwritten Symbols

4.1 Introduction

As stated in Chap. 2, almost every existing recognition method can be used for the labeling of on-line characters. The methods presented in that chapter attempt to improve recognition accuracy by using selected descriptors and other important features of on-line symbols.

Following this approach to design specialized classifiers, we designed a classifier based on the number of strokes. In total, we constructed three classifiers, one each corresponding to symbols having one stroke, two strokes, and three *or more* strokes. Fig. 4.1. After preprocessing, we obtain a simplified version of the symbol, which is used to construct the feature vector which is fed into the classifier. See Fig. 4.1.

This chapter has two sections, one gives an overview of classification methods and the other describes our experimental results. For a complete reference of the classification methods we used in this chapter, we refer the reader to [93, 72, 24, 21, 101].

4.2 Classification Approaches

In artificial intelligence, classification means the use and implementation of algorithms to map objects of datasets into a set of labels: computers "learn" to classify objects without human guidance or intervention. Human influence in the learning process is generally reduced to the construction of different representations of these objects.

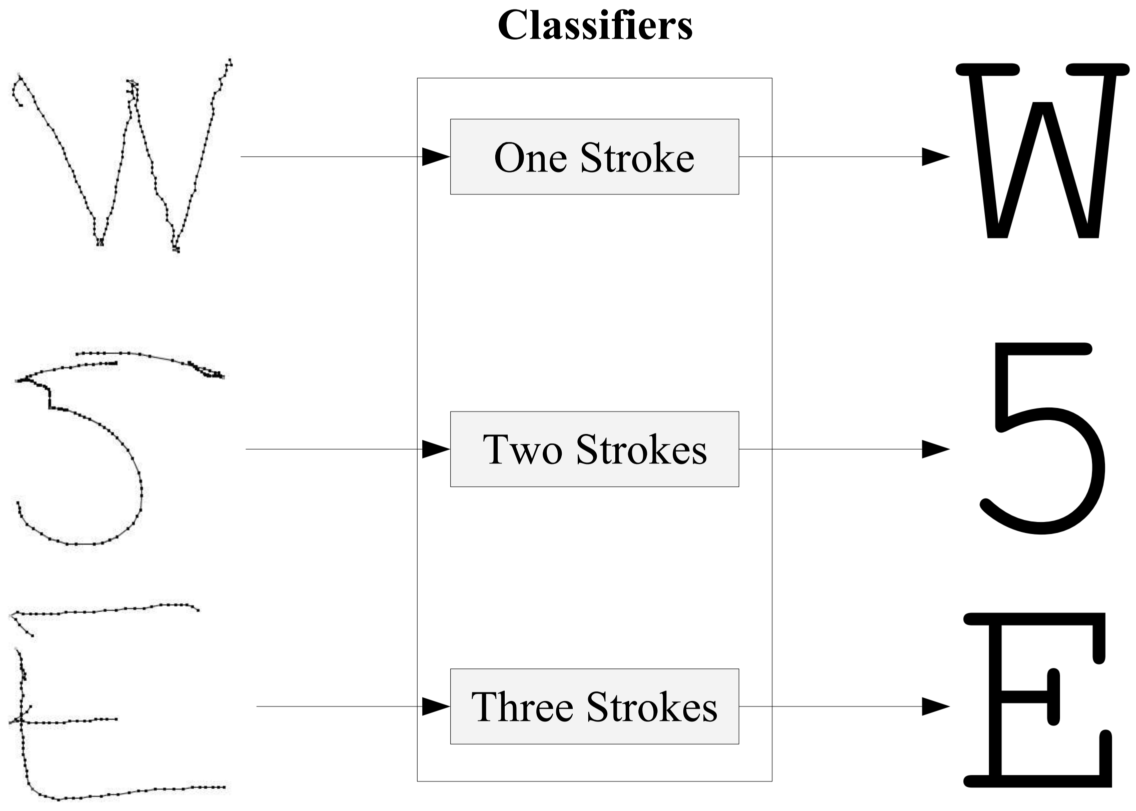

Figure 4.1: *Classifiers.*

A variety of authors consider two types of learning algorithms:

- **Supervised Learning.** In this case, a dataset
$$D = \{(x^{(i)}, y^{(i)}) : i \in I\} \tag{4.1}$$
consists of data pairs, where $x^{(i)}$ represent an object, $y^{(i)}$ its corresponding label, and i is an element of the index set I. Algorithms use the dataset to "learn" to classify new data. These algorithms construct a function $f(x) = y$, which assigns an object x to a label y.

- **Unsupervised Learning.** In this case, we don't know a priori which label corresponds to a given object. Thus, unsupervised learning deals with sets of unlabeled objects
$$D = \{x^{(i)} : i \in I\}. \tag{4.2}$$
Algorithms for unsupervised learning try to describe the underlying nature and structure of the data. This is accomplished by constructing groups (clusters) of

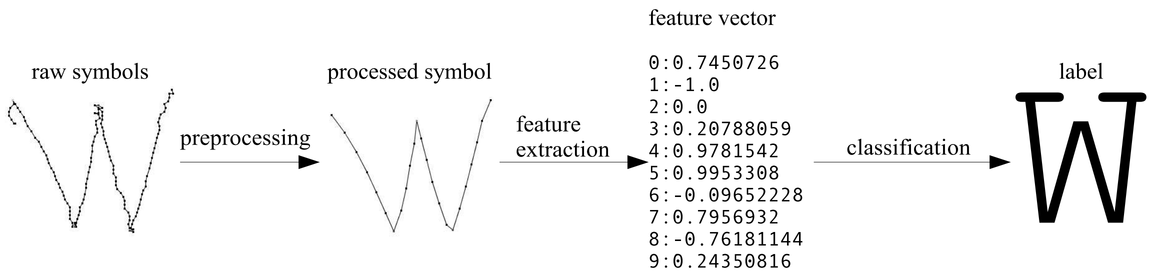

Figure 4.2: *Classification procedure.*

data or by selecting the “right” features from data. These algorithms not only construct a classification function but also construct labels which are associated with different clusters.

The set D is also known as a *training set*, because the function f is the result of “training” functions (or the parameters) using the data set D.

The objects $x^{(i)}$ are normally represented by vectors taking numerical values, e.g. $x^{(i)} \in \mathbb{R}^n$. An object in D is also known as a pattern, sample, example, vector, etc. When the label $y^{(i)}$ takes two values, frequently from the sets $\{0, 1\}$ or $\{-1, 1\}$, we are dealing with *binary classification.* When $y^{(i)}$ takes the values $\{c_1, \ldots, c_N\}$, $N > 2$, we have *multiclass classification.* If the value $y^{(i)}$ takes continuous numerical values, we have *regression.*

Normally, it is assumed that the map f belongs to a family of functions which is characterized completely by a set of parameters Λ. This parameter dependence is indicated by $f = f_\alpha$, $\alpha \in \Lambda$. Learning algorithms normally consist of methods to obtain the parameters $\alpha^* \in \Lambda$, which optimize certain “performance” or loss criterion involving the example patterns. The selection of the parameter dependence and the loss criterion characterize the different classification approaches.

4.2.1 Bayesian Classification

Bayesian classification uses Bayes’ Theorem as follows. If we assume that $P(c_j)$, $j = 1, \ldots, N$, describe the prior probability of the objects belonging to the class c_j, then the posterior probabilities $P(c_j|x)$, i.e. the probability of label c_j given the pattern x, can be calculated with the formula

$$P(c_j|x) = \frac{P(x|c_j)P(c_j)}{P(x)}. \tag{4.3}$$

Given a new example x, the Bayes classification rule consists of labeling x with the class having the maximum probability:

$$f(x) = c_{j_{\max}}, \tag{4.4}$$

where

$$j_{\max} = arg \max_j P(c_j|x) = \arg\max_j P(x|c_j)P(c_j). \tag{4.5}$$

The quantity $P(x)$ is eliminated, because does not play any role to obtain the maximum index. The prior probabilities $P(c_j)$ are taken as the proportion of each class in the training set. Note that the conditional probability $P(x|c_j)$ characterizes completely the classification rule.

Training Bayesian Classifiers

It is commonly assumed that $P(x|c_j)$ is described as a multivariate normal distribution:

$$P(x|c_j) = \frac{1}{\sqrt{2\pi \det \Sigma_j}} \exp \frac{1}{2}(x-\mu_j)^T \Sigma_j^{-1}(x-\mu_j). \tag{4.6}$$

Thus, the learning process consists of finding the optimal parameters Σ_j^* and μ_j^* using the dataset D. Under the assumption of independence of features and classes, it suffices to estimate the priors of each class independently using the methods of *maximum likelihood*. Using this method, one can show that the optimal parameters Σ_j^* and μ_j^* correspond to the covariance matrix and the mean vector of the data D respectively.

Because independence assumptions do not necessarily describe the complex nature underlying the data in D, this classification schema is also known as *naive Bayesian classification*. Actually, these "naive" assumptions simplify calculations during the learning process. Bayes classification also has an theoretical importance because there are some theorems relating the *Bayes error* to errors of other classification methods.

4.2.2 Nearest Neighbors

One of the simplest approaches for classification is the *nearest neighbor* rule. In this classification criterion we assume that a certain *distance* function d is defined in the set of patterns. Thus, the label of an unclassified object x is the same of the nearest $x^{(i_{\min})}$ pattern in D:

$$f(x) = y^{(i_{\min})}, \tag{4.7}$$

where

$$i_{\min} = \arg\min_i d(x, x^{(i)}). \tag{4.8}$$

Note that this classification function classifies correctly all data in D. A variation of this rule is to take k nearest neighbors $x^{(i_1)}, \ldots, x^{(i_k)}$ of x. A simple voting scheme assigns a label to the unclassified vector: the label $\{c_1, \ldots, c_N\}$ which most frequently appears in the set $\{y^{(i_1)}, \ldots, y^{(i_k)}\}$ is assigned to x. This rule guarantees that a numerical output is generated which can be used to estimate the degree of class membership of x, which is useful for post-processing purposes. This estimation is accomplished by counting the times a certain label appears in $\{y^{(i_1)}, \ldots, y^{(i_k)}\}$, and dividing this quantity by k.

Computationally, the nearest neighbor makes no effort to learn from examples. But a price must to be paid for this *lazy* classification: this rule does not guarantee good performance for *new* patterns, unless we provide a really big dataset D. In the limit, when the number of examples approaches infinity, the error rate of the single nearest neighbor rule is not worse that twice the optimal Bayes error. However, having a big database increases the classification time, and the performance of the nearest neighbor classifier in terms of speed deteriorates.

4.2.3 Classification Trees

Classification trees work by partitioning the training set using a set of decision rules. Figure 4.3 shows the structure of a *binary* classification tree. The nodes of the tree corresponds to a test or decision, which can be either true of false. The evaluation of input data begins at the root and, depending on the test, we follow the right or the left branch until we reach a leaf. There, the final decision assigns a class to the pattern. No binary trees are allowed to have more than two branches at any node. One of the benefits of classification trees is that their structure allows interpretation of the rules generated during training: a path joining the root to one of the leaves corresponds to a decision rule, constructed with a conjunction (AND operator) of all the tests taken at each node.

In the case of numeric data, the test taken at each node $\mathcal{N}$ is to compare some of the coordinates x_i of the sample x to a determined value $x_{i\mathcal{N}}$, using the rule $x_i < x_{i\mathcal{N}}$. Geometrically, this corresponds to constructing hyperplanes perpendicular to the coordinate axes which separate the data. Thus, every time the we reach a node, the training set is partitioned into two sets, one satisfying the evaluation rule and the

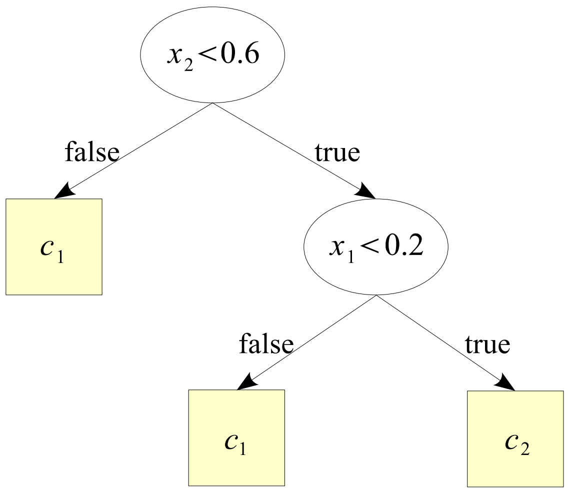

Figure 4.3: *Structure of a classification tree.*

other not satisfying it. This decision outcome is called a *split*, because it corresponds to a *splitting* of the data. We concentrate in this section on binary classification trees.

Training Classification Trees

The process of learning the structure of a classification tree is also known as *rule induction*. The induction method starts with the root node and the training set. We select a value $x_{i\mathcal{N}}$ to create a new rule which divides the training data into two disjunct sets. Two new nodes are created and assigned to this sets. The procedure is repeated in these nodes and the split data, unless the nodes themselves become a leaf. A node becomes a leaf if all the elements in the set belong to a single class or when the examples satisfy certain criteria or confidence levels.

Most of the rule creation criteria reduce *impurity* within successive nodes when splitting the current one. The most used impurity functions are the *entropy impurity*

$$i(\mathcal{N}) = -\sum_{j=1}^{N} p_j \log p_j \tag{4.9}$$

and the *Gini impurity*

$$i(\mathcal{N}) = 1 - \sum_{j=1}^{N} p_j^2, \tag{4.10}$$

where p_j is the fraction of patterns at node $\mathcal{N}$ that belongs to the class c_j.

In the training algorithm, the value $x_{i\mathcal{N}}$ is selected as the one which minimizes the increment of impurity:

$$\Delta i(\mathcal{N}) = i(\mathcal{N}) - p_l i(\mathcal{N}_l) - p_r i(\mathcal{N}_r), \tag{4.11}$$

where p_l and p_r are the fractions of patterns associated to the left node $\mathcal{N}_l$ and right node $\mathcal{N}_r$ respectively.

The criterion to stop splitting mentioned above, when patterns in the node belong to a single class, can generate very large classification trees which correctly classify training data but the accuracy on new samples is poor. This undesirable phenomenon also occurs when the tree continues to grow until the leaves reach minimal impurity. To void this, we can use some other criteria to stop splitting. This can be, for example, stopping when the number of examples in the split falls below fixed threshold, e.g. 5 % of the training data. Similarly, when the impurity is below some threshold. There are more criteria based on exceeding thresholds, e.g. the one estimating the *chi square statistic* χ^2. Other criterion is *cross validation*: the tree is trained on a fraction of the data (for example 90 % of it) and the rest is kept as a validation set. Splitting continues until the classification error on the validation set increases. In all cases, the class assigned to the new leaf corresponds to the label having the greatest frequency.

4.2.4 Artificial Neural Networks

Artificial neural networks are one of the most popular classification techniques widely applied to a great quantity of pattern-recognition problems in the areas of computer vision, character recognition, speech recognition, etc. Artificial neural networks try to mathematically model biological networks, i.e. neurons are represented by single calculation units highly interconnected each other. For this reason, this research area of artificial intelligence is also known as *connectionism*.

The Perceptron

An artificial neural net is constituted by numerical values, weighted edges, and calculation units. Figure 4.4(a) represents the minimal net calculation unit, also known as *perceptron*. When the edges connect numerical values, they transmit these values multiplied by the corresponding weights, as shown in Figure 4.4(b). The circles represent the nodes of the net and constitute the calculation units. The edges connected at the left of units transmit values which are summed. The function s associated to this node is applied to the result of the sum and further transmitted to the left to obtain the final output $f(x)$, expressed as

$$f(x) = s(\sum_{i=1}^{n} w_i x_i + b) = s(w \cdot x + b). \qquad (4.12)$$

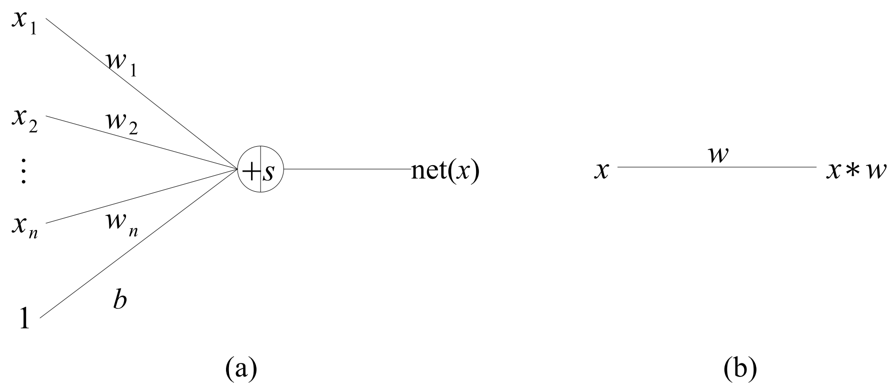

Figure 4.4: *A weighted edge and the result of the transmission.*

The parameters w and b are known as *weight* and *bias* respectively.

The perceptron rule is normally used for binary classification: the function s is taken as the sign function, where $s(u) = 1$ if $x \geq 0$ and $s(u) = -1$ if $x < 0$. The perceptron has a geometrical interpretation: it divides the real space $\mathbb{R}^n$ into two regions separated by the hyperplane defined by the equation

$$w \cdot x + b = 0. \tag{4.13}$$

If there is a such hyperplane that correctly classify the training set D, D is said to be *linearly separable.*

The training algorithm for the perceptron consist of the correction of the parameters w and b when the i-th training pattern is incorrectly classified: initialize both parameters to zero and for each $i = 1, \ldots, \ell$ take $(x^{(i)}, y^{(i)})$ and update the parameters

$$\begin{aligned} w &\leftarrow w + y^{(i)} x^{(i)} & (4.14) \\ b &\leftarrow b + y^{(i)}, & (4.15) \end{aligned}$$

when

$$y^{(i)}(w \cdot x^{(i)} + b) > 0 \tag{4.16}$$

does not hold, i.e. $x^{(i)}$ is not correctly classified. Theoretical results show that the perceptron algorithm takes a finite number of steps when D is linearly separable. However, it is not possible to know that beforehand, so we can stop the algorithm when a determined number of steps has been reached or when all patterns in D are correctly classified.

Some authors work with an extended version of the input vector x and the weight w:

$$\hat{x} = (x_1, \ldots, x_n, 1)^T \text{ and } \hat{w} = (w_1, \ldots, w_n, b)^T. \tag{4.17}$$

The bias b is now included in the weight vector and the rules (4.12)-(4.15) are simplified to:

$$f(x) = s(\hat{w} \cdot \hat{x}), \tag{4.18}$$

$$y(\hat{w} \cdot \hat{x}) > 0, \tag{4.19}$$

$$\hat{w} \leftarrow \hat{w} + y^{(i)}\hat{x}^{(i)}. \tag{4.20}$$

Neural Networks

Figure 4.5 shows the structure of a neural network with a hidden layer, a more complex structure of calculation units. In total, the net consists of three *layers* $x(k)$, $k = 0, 1, 2$. The input layer $x(0)$ is an extended version $(x_1, \ldots, x_n, 1)^T$ of the input vector x. The vector $x(1)$ of dimension m corresponds to the output of the hidden layer. The vector $x(2)$ of dimension N is the final output layer. The n entries of x are *propagated* through the net to obtain the N output values $x(2)$ at the left. Thus, the classification rule corresponds to

$$f(x) = c_{j_{\max}}, \tag{4.21}$$

where

$$j_{\max} = \arg\max_j x_j(2). \tag{4.22}$$

The layers are connected with two matrices: $W^{(1)}$ of size $(n+1) \times (m+1)$ for the weights connecting the input with the hidden layer, and $W^{(2)}$ of size $(m+1) \times N$ for the ones connecting the hidden values with the output layer. (Note that this matrices include also a bias, as in the extended version of the perceptron rule.) For example, the value $W^{(1)}_{i,j}$ is the weight associated with the edge connecting the i-th entry of x with the j-th hidden unit. In this way, the calculation of the final output of the net can be seen as a product of matrices and function composition:

$$x(k) = s(x(k-1)W^{(k)}), \quad k = 1, 2. \tag{4.23}$$

Note that this formula can be easily generalized for any number of hidden units.

The function s in (4.23) is applied to each element and is selected to be continuous and differentiable. A classic selection of this function is the *sigmoid*

$$s(x) = \frac{1}{1 + e^{-x}} \tag{4.24}$$

or the *hyperbolic tangent*

$$s(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{4.25}$$

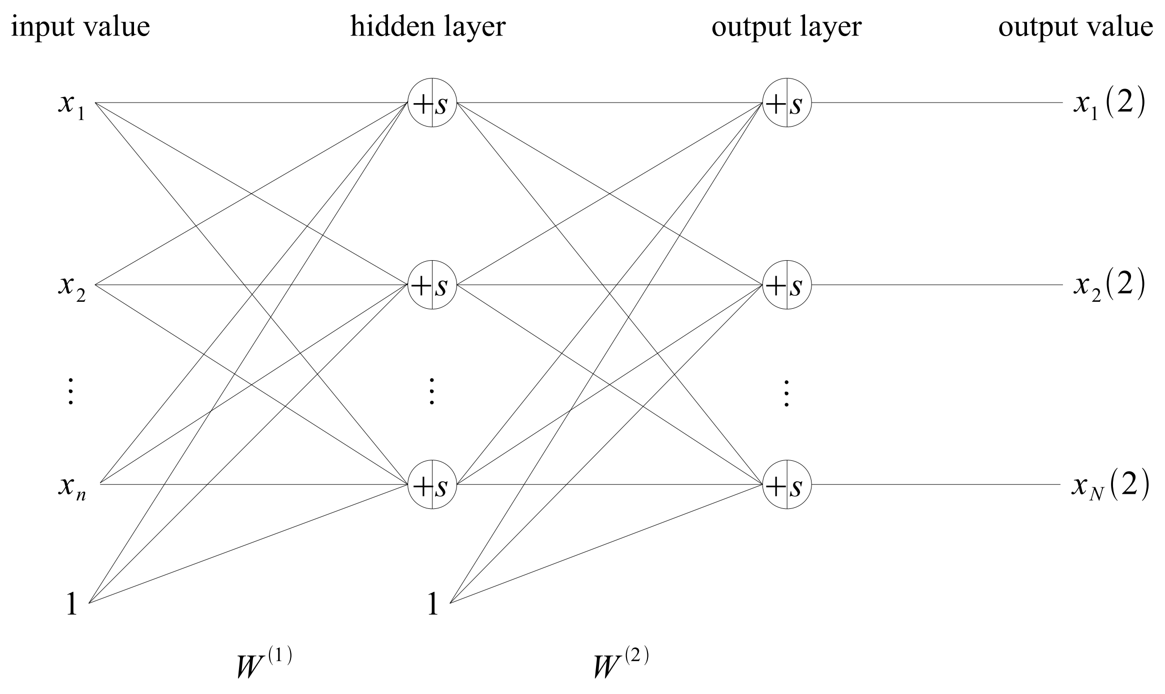

Figure 4.5: *Structure of an artificial neural network with a hidden layer.*

Training Artificial Neural Networks

As we can see from the structure of the network, the function is characterized completely by the edge weights. The weights are selected such that they minimize the *quadratic error*

$$E = \sum_{i=1}^{\ell} \|x^{(i)}(2) - y\|^2, \tag{4.26}$$

where $x^{(i)}(2)$ represent the net's output of the training pattern $x^{(i)}$ and

$$y_j = \begin{cases} 1, & \text{if } x \text{ belongs to class } c_j, \\ 0, & \text{otherwise.} \end{cases} \tag{4.27}$$

The choice of the sigmoid is not a coincidence: this function makes also (4.26) to be continuous and differentiable with respect to the weights. This allows the application of a *gradient descendent* method to obtain the optimal weight values that minimize the quadratic error. The idea is to iteratively update the weights using the formula

$$W_{ij}^{(k)} \leftarrow W_{ij}^{(k)} - \eta \frac{\partial E}{\partial W_{ij}^{(k)}}, \tag{4.28}$$

where η is the *learning factor*. The calculation of the partial derivative at the left of (4.28) involves various steps which constitute the well-known *back-propagation* *algorithm* for training neural networks [72].

One of the drawbacks in the method is that the weights found by back-propagation can correspond to a local minimum of (4.26). To overcome this problem, the calculation of the partial derivatives used in (4.28) takes place in a subset of randomly selected training patterns rather than in the complete dataset, also known as *stochastic training*. Other problems correspond to the long duration of the training, which can be hours or even days when dealing with big databases. To accelerate the training, authors use the second derivative of (4.26) (second-order methods), adjust dynamically the learning rates, or implement algorithms in parallel computers, among other methods. Another important problem is the so called *over-specialization*, which corresponds to obtaining excellent classification rates on a training set but poor accuracy on new patterns. As a solution, authors multiply the first term in (4.28) by a factor < 1 (weight decay), remove the weights that are least needed (weight pruning), and stop training when the error in an independent dataset increases (early stopping).

4.2.5 Support-Vector Machines

Most of the pattern-recognition methods, as in the case of neural networks, are based on minimizing an empirical function that measures the classification error with given patterns. An empirical performance measure is the *empirical risk*, defined as

$$R_{\text{emp}}(\alpha) = \frac{1}{n} \sum_{i=1}^{\ell} \lambda(y^{(i)}, f_\alpha(x^{(i)})), \tag{4.29}$$

where λ is the *loss function*. When learning binary classification, taking the labels $y \in \{-1, 1\}$, one uses the 0/1-loss function

$$\lambda(y, f(x)) = \begin{cases} 1 & yf(x) \geq 0 \\ 0 & yf(x) < 0 \end{cases}. \tag{4.30}$$

Learning algorithms try to find the parameters α^* which minimize the risk R_{emp}. However, the parameters α^* not necessarily minimize the *structural risk*

$$R(\alpha) = \int \lambda(y, f_\alpha(x))\, dp(x, y), \tag{4.31}$$

which is the *actual error* of misclassification of the new patterns generated according to probability distribution p. Unfortunately, the minimization of (4.31) cannot be done directly, because the fixed probability distribution p is not known.

The theory developed by Vapnik and Chervonenkis [94] provides upper bounds for the structural risk. These bounds depend on the empirical risk and the *VC dimension* $h = \text{VCdim}(F)$ of the family of classification functions $F = \{f_\alpha\}_{\alpha\in\Lambda}$: for all $\delta > 0$ and $\alpha \in \Lambda$ the inequality

$$R(\alpha) \leq R_{\text{emp}}(\alpha) + \sqrt{\frac{h(\ln\frac{2n}{h} + 1) - \ln\frac{\delta}{4}}{n}} \tag{4.32}$$

holds with probability $1 - \delta$ for $h < n$ and λ is the 0/1-loss function.

The VC dimension $h = \text{VCdim}(F)$ associated to the family of functions F, the *learning machines*, is an integer value (or infinite) which is a measure of the classification *capacity* of these machines. Given a number ℓ of patterns, they can be labeled in 2^ℓ ways, using binary labeling. If there is a function in F which correctly classifies these data, we say that the data is shattered by F. The VC dimension of F is defined as the maximum number of training points which can be shattered by the function family.

Given a set of learning machines in F, a structure in F is a sequence of nested subsets $F_1 \subset F_2 \subset \ldots \subset F_M \subset F$. This structure has the property that

$$R_{\text{emp}}(\alpha_1^*) \geq R_{\text{emp}}(\alpha_2^*) \geq \ldots \geq R_{\text{emp}}(\alpha_M^*) \geq R_{\text{emp}}(\alpha^*), \tag{4.33}$$

and

$$R(h) \leq R(h_1) \leq R(h_2) \leq \ldots \leq R(h_M), \tag{4.34}$$

where α_j^* is the optimal parameter which minimizes R_{emp} in F_j, and $R(h_j)$ is the term at the left of (4.32), see Fig. 4.6. The *Structural Risk Minimization Principle* (*SRMP*) is a learning scheme which uses the properties of (4.33)-(4.34). It consist of fixating a structure in F and taking the index j^* which minimizes the right side of (4.32).

Training Support-Vector Machines

The idea behind *support-vector machines* (*SVM*s) is to define a structure on *linear machines*, the classification functions defined by the perceptron rule

$$f(x) = \text{sign}(w \cdot x + b), \tag{4.35}$$

under the condition

$$\min_{i\in I} |w \cdot x^{(i)} + b| = 1. \tag{4.36}$$

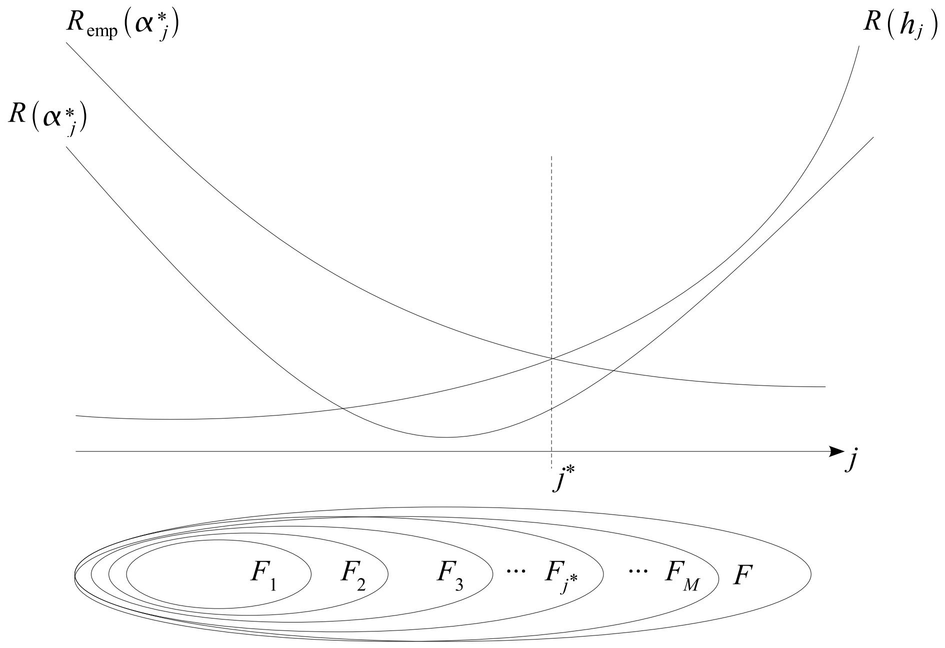

Figure 4.6: *Illustration of the Structural Risk Minimization Principle. The curve labeled $R(h_j)$ represents the values of the second term of the sum in (4.32). The learning principle selects the optimal index j^* which minimizes the sum of the values $R_{\text{emp}}(\alpha_j^*)$ and $R(h_j)$.*

If the hyperplane $w \cdot x + b = 0$ fulfills (4.36), we say that it has the canonical form. Note that the set of canonical hyperplanes coincides with all hyperplanes, because canonical hyperplanes use a normalized form of the parameters w and b.

Vapnik proved that the VC dimension h of the canonical hyperplanes satisfies the inequality

$$h \leq R^2 A^2 + 1, \tag{4.37}$$

where R is the radius of the smallest sphere containing D and $\|w\| \leq A$. Defining a structure on canonical hyperplanes corresponds to taking a sequence of values A_j [94].

Training linear machines, consist of minimizing $\|w\|$ while keeping correct classification of the data. If the data is linearly separable, this is accomplished by solving the following quadratic programming problem:

$$\text{minimize} \quad \frac{1}{2}\|w\| \tag{4.38}$$

$$\text{subject to} \quad y^{(i)}(w \cdot x^{(i)} + b) \geq 1 \tag{4.39}$$

The solution of (4.38)-(4.39) has an interesting geometric interpretation. If $x \in D$ and γ_x is the distance from the separating hyperplane to point x, then we know that

$$\gamma_x = \frac{|w \cdot x_+ + b|}{\|w\|} \tag{4.40}$$

The minimum of the distance

$$\gamma = \min_{x \in D} \gamma_x \tag{4.41}$$

is called the margin. Observe that

$$\begin{aligned}
\gamma &= \min_{x \in D} \gamma_x && (4.42)\\
&= \min_{x \in D} \frac{|w \cdot x_+ + b|}{\|w\|} && (4.43)\\
&= \frac{1}{\|w\|} \min_{x \in D} |w \cdot x_+ + b| && (4.44)\\
&= \frac{1}{\|w\|}, && (4.45)\\
& && (4.46)
\end{aligned}$$

since we are dealing with canonical hyperplanes. Thus, minimizing $\|w\|/2$, corresponds to the maximization of $2/\|w\|$ we can deduce that parameters obtained by training linear machines correspond to the hyperplane having the maximum margin. The points of D where the maximum margin is reached – the case of equality in (4.39) – are known as support vectors. (See Fig. 4.7.) The margin is related to the VC dimension by (4.37).

However, linear separability is a very strong assumption which in general is not satisfied. The optimizing problem (4.38)-(4.39) is modified to handle non-linearly-separable data, and one solves the following quadratic programming problem:

$$\begin{aligned}
\text{minimize} \quad & \frac{1}{2}\|w\| + C \sum_{i=1}^{\ell} \xi_i && (4.47)\\
\text{subject to} \quad & y^{(i)}(w \cdot x^{(i)} + b) \geq 1 - \xi_i && (4.48)\\
& \xi_i \geq 0. && (4.49)
\end{aligned}$$

The constrain (4.48) assures the classification of the pattern $x^{(i)}$ with a tolerance ξ_i. The parameter C can be regarded as a regularization parameter.

To avoid certain numeric and implementation problems, the problem (4.47)-(4.49) is transformed into a dual problem using the technique of Lagrange multipliers:

$$\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y^{(i)} y^{(j)} (x^{(i)} \cdot x^{(j)}) && (4.50)\\
\text{subject to} \quad & 0 \leq \alpha_i \leq C, && (4.51)\\
& \sum_{i=1}^{\ell} y^{(i)} \alpha_i = 0. && (4.52)
\end{aligned}$$

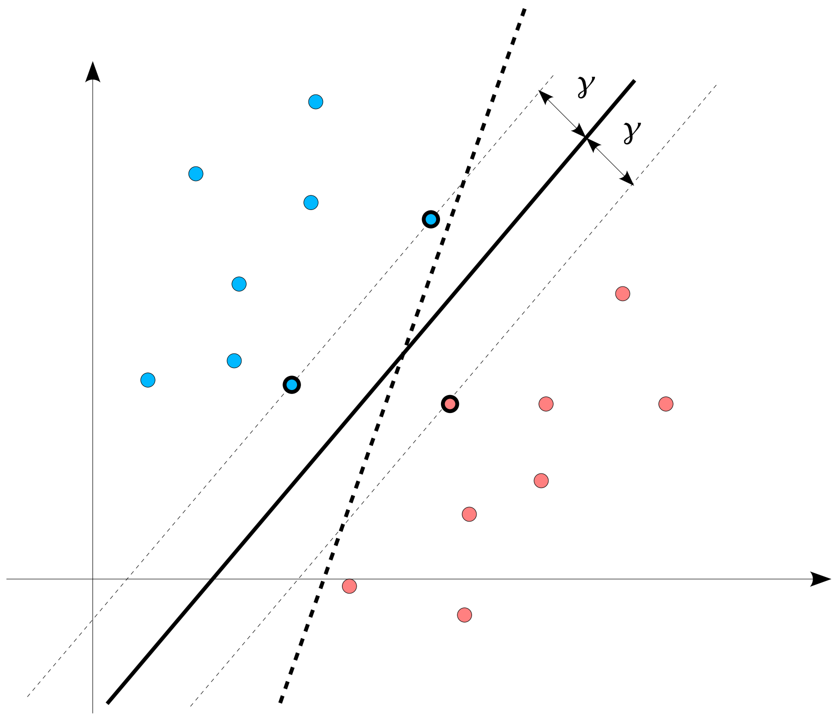

Figure 4.7: *Geometric interpretation of the linear machine trained on linearly separable data. The maximum margin $\gamma = 1/\|w\|$ is reached by the hyperplane represented by the continuous line. The dotted line is another separating hyperplane, but with a smaller margin. The support vectors are represented by wide-margin circles.*

An important characteristic of this learning scheme is that the classification function f can be expressed in terms of the training vectors $x^{(i)}$ for which the solutions α_i of (4.50)-(4.52) are positive, called *support vectors* (SVs), by setting w and b as follows:

$$w = \sum_{\alpha_i \in \mathrm{SV}} y^{(i)} \alpha_i (x^{(i)} \cdot x), \quad b = \frac{1}{2}(w \cdot x_+ + w \cdot x_-), \tag{4.53}$$

where x_+ and x_- are two SVs which belong to the positive and negative class respectively. The SVs are interpreted as the relevant patterns in the classification process. Frequently, the number of SVs is small with respect to the number of training data, thus giving a compact representation and efficient implementation of the classifier.

Another important characteristic of SVMs is that non-linearity can be introduced by replacing the inner product in (4.52) by a *kernel function*

$$K(x^{(i)}, x^{(j)}) = \phi(x^{(i)}) \cdot \phi(x^{(j)}), \tag{4.54}$$

where ϕ maps the patterns into a high (possibly infinite) dimensional inner product

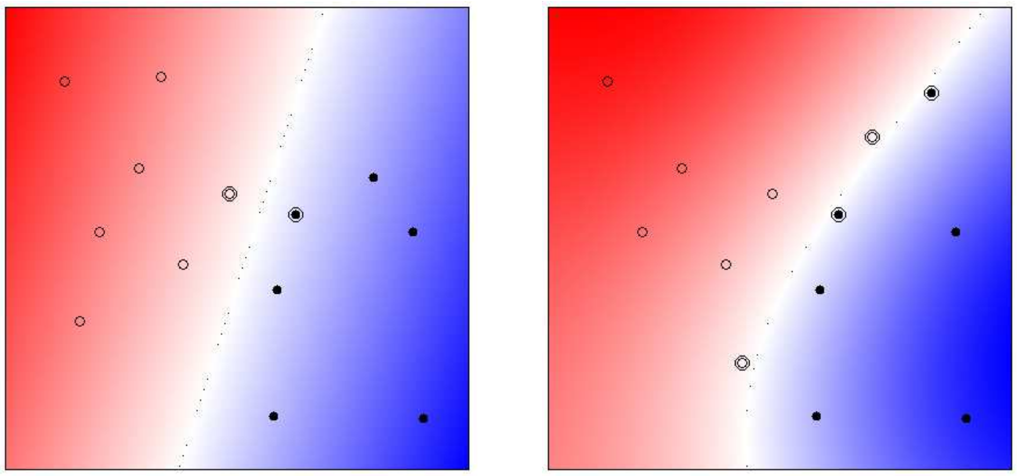
(a) (b)

Figure 4.8: *(a) Binary classification applied to a separable set using a linear SVM. (b) Binary classification of a non-separable set using an RBF function.*

space. Some of the best known (and most widely used) kernel functions are:

$$K(x,z) = \exp\left(-\gamma\|x-z\|\right), \tag{4.55}$$
$$K(x,z) = \left(\gamma((x\cdot z)+1)\right)^d, \tag{4.56}$$
$$K(x,z) = \tanh(\gamma(x\cdot z)-\theta)), \tag{4.57}$$

which are know as *radial basis functions* (*RBFs*), *polynomial kernels*, and *hyperbolic kernels* respectively.

Multi-Class Support-Vector Machines

For multi-class SVMs, one can use *Directed Acyclic Graphs SVM* (*DAG-SVM*) [69]. The classification criterion is determined via a rooted binary graph. The nodes are arranged in a triangle with the root at the top and the leaves at the bottom, as showed in Fig. 4.9. Binary SVM classifiers are implemented in the $k(k-1)/2$ internal nodes of the graph. The k leaves indicate the class predicted label. Given a new example x, classification starts with the classifier in the root. A node is exited via the left edge if the label in the classification is positive, or the right edge if the label is negative. In this way, the next classification node is evaluated. One continues this procedure until reaching a leaf.

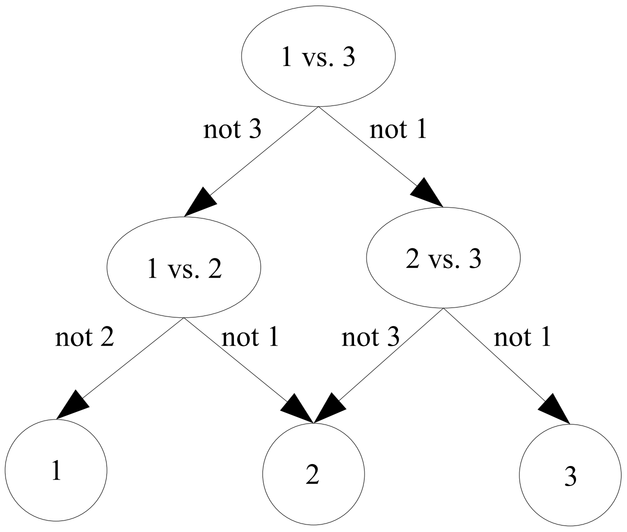

Figure 4.9: *DAG for three classes.*

4.3 Experimental Results

In this section we present the results of our experiments with two kind of data: *user-dependent data* and the *UNIPEN* database. We used the user dependent data in our first implementation of the support-vector approach for on-line symbol classification. As expected, the results in this database were very good. To verify whether these results could also be accomplished with user independent data, we continued our experiments with the UNIPEN database.

4.3.1 User-Dependent Classification

Our data were obtained from a Wacom Graphire 2 tablet connected to a Sony Vaio PCG-FX502 notebook. The symbols were written by a volunteer, and each class contains fifty symbols (see Fig. 4.10). The number of symbols was artificially increased ten times by means of transformations related to the tangent distance, as described in Sect. 3.4. We randomly selected 50 % of the data for training, 25 % for testing and 25 % for validation. We assumed that the symbols were composed at most of tree strokes, and tree classifiers were used to classify them, depending of the number of strokes. They were preprocessed as described in Chap. 3. The number of points of each stroke was limited to sixteen. The features used are the ones described in Sect. 3.5, and if a symbol had more than one stroke, we constructed their corresponding feature vectors and then concatenated them to obtain the final vector.

Our system uses a DAG-SVM as multi-class SVM-classifier. *Sequential Minimal*

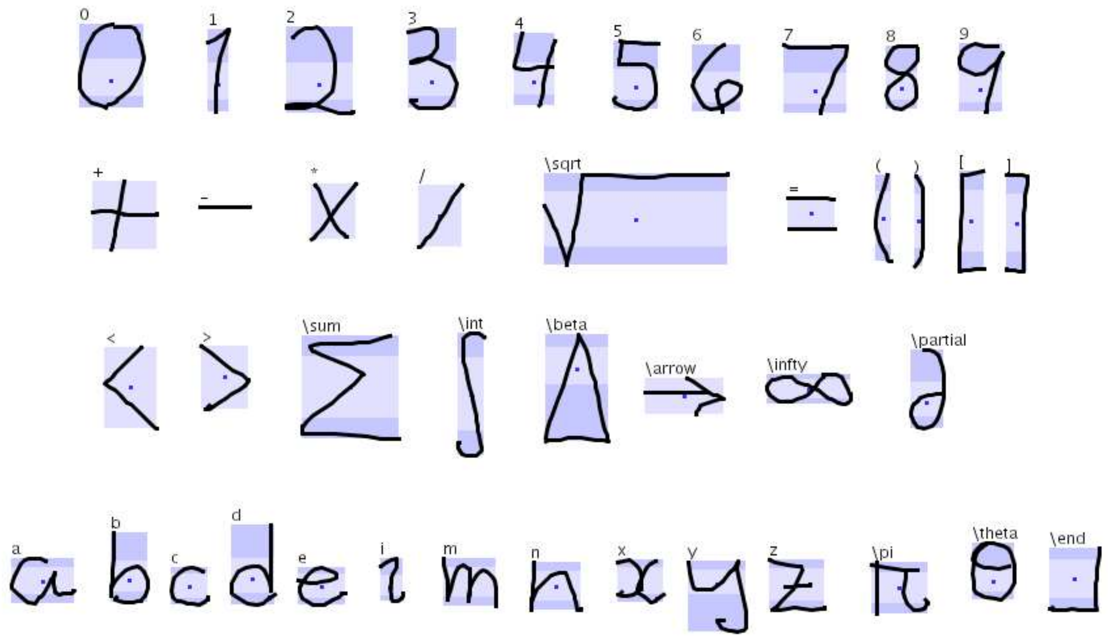

Figure 4.10: *Symbols constituting the user-dependent database.*

Optimization (*SMO*) algorithm was used to train the classification nodes [68, 43]. We used the RBF kernel for all the classification nodes of the DAG-SVM. Experiments suggested a value of $\gamma = 0.001$. For comparison, we trained a neural network with the RPROP algorithm with the standard sigmoid and its standard parameter values [71]. The training was finished by using an early stopping criterion with the validation set. The number of hidden units for each net were the same as those of the dimension of the feature vector.

Table 4.1 summarizes the results obtained for each classifier. The number indicates the correspondence of the classifier and the number of strokes in the symbols. We trained each ANN five times and selected the best one with respect to the error rate on the validation set. We used the support vectors found in the SVM training to train other ANN (marked with an asterisk in the table). Because of the size of the support vectors, we found that the training time is faster and classification rates improve.

When we added more features to the vector, we found some improvement in the classification rates and a reduction in the number of SVs, which is a good result because it increases the speed of the classification. The results for the one-stroke classifiers are shown in Table 4.2. The features were added in their order of description in Sect. 3.5.

Table 4.1: *Classification rates for user-dependent classification.*

Classifier	Support Vectors	Training error	Test error	Val
SVM1	29.65 %	0.31 %	0.93 %	0.76 %
ANN1	-	0.72 %	1.17 %	1.27 %
ANN1*	-	0.98 %	1.0 %	1.06 %
SVM2	36.73 %	0.00 %	0.62 %	0.70 %
ANN2	-	0.69 %	1.31 %	2.09 %
ANN2*	-	1.26 %	1.78 %	2.08 %
SVM3	39.41 %	0.00 %	0.00 %	1.17 %
ANN3	-	0.00 %	1.17 %	1.17 %
ANN3*	-	0.00 %	1.17 %	1.17 %

Table 4.2: *Classification rates with feature integration.*

Classifier	Support vectors	Training error	Test error	Val
SVM1	45.45 %	2.43 %	2.86 %	3.07 %
SVM2	46.64 %	1.84 %	2.76 %	2.27 %
SVM3	**27.85 %**	1.00 %	1.41 %	1.31 %
SVM4	29.44 %	0.29 %	0.86 %	1.07 %
SVM5	29.23 %	**0.17 %**	1.31 %	**0.76 %**
SVM6	29.13 %	0.24 %	0.79 %	1.07 %
SVM7	29.70 %	0.31 %	**0.75 %**	0.82 %
SVM8	29.65 %	0.31 %	0.93 %	**0.76 %**

4.3.2 Experiments with the UNIPEN Database

The UNIPEN foundation provides a public database of on-line handwritten data donated by researchers from universities and research centers around the world [36]. The version of the database we used in the experiments is UNIPEN Train-R01/V07. This data is formed by isolated characters, isolated words, as well as by free text. Because of the great variety of sources, the database is considered as difficult and challenging.

Unfortunately, this database does not contain Greek letters or mathematical operators, let alone complete mathematical expressions. For that reason, our experiments with this database consider only the data 1a, 1b, and 1c, which correspond to isolated

Table 4.3: *Lost of data when reading the UNIPEN database.*

Database	Number of symbols	Read symbols	Lost
1a	15953	15950	0.0188 %
1b	28069	28066	0.0106 %
1c	61351	57484	6.0303 %

digits, isolated upper letters, and isolated lower case letters respectively. In each case, the data were preprocessed as described in Chapt. 3. In addition to these preprocessing steps, strokes that had a length smaller than 12 % of the symbol's diagonal were removed. The data were extracted from the database using the `uptools3` programs which can be found on the UNIPEN Foundation website. During the extraction process, we encountered some problems when reading the files from the database and some data where not included in our experiments. Table 4.3 summarizes the amount of data that was lost.

The library libsvm-2.6 by Chang and Lin [16] was used to train the SVM-classifier. Their software offers n-fold cross-validation to verify the accuracy of the classification. This accuracy proof consist of randomly dividing the data into n subsets of the same size and taking $n-1$ sets for training and one for testing the obtained classifier. This is repeated n times, so that all the n accuracy measures are obtained and averaged to obtain the final accuracy estimation. We used the RBF kernel for all the classification nodes of the DAG-SVM. We used the values of C and γ as indicated in Table 4.4. These values were obtained using a grid search and five-fold cross-validation in the intervals $0 \leq \log_2 C \leq 16$ and $-16 \leq \log_2 \gamma \leq 0$, see Fig. 4.11. The results of the accuracy for the classification is shown in Table 4.5.

The total accuracy on the whole is calculated using the formula

$$\text{Total accuracy} = \frac{\sum_{i=1}^{3} a_i n_i}{\sum_{i=1}^{3} n_i}, \tag{4.58}$$

where a_i, n_i, i represent the accuracy, the number of symbols in the subset, and the stroke number respectively.

To compare the accuracy of the SVM-based classifier, we trained other classifiers using the data-mining tool weka-2.4.2 [101]. We used a computer with a Pentium III processor at 800 MHzand 512 MB of RAM running Debian GNU/Linux operating system. The accuracy is measured using stratified 10-fold cross validation, an evaluation option available in the software. For classifier training, we used the default

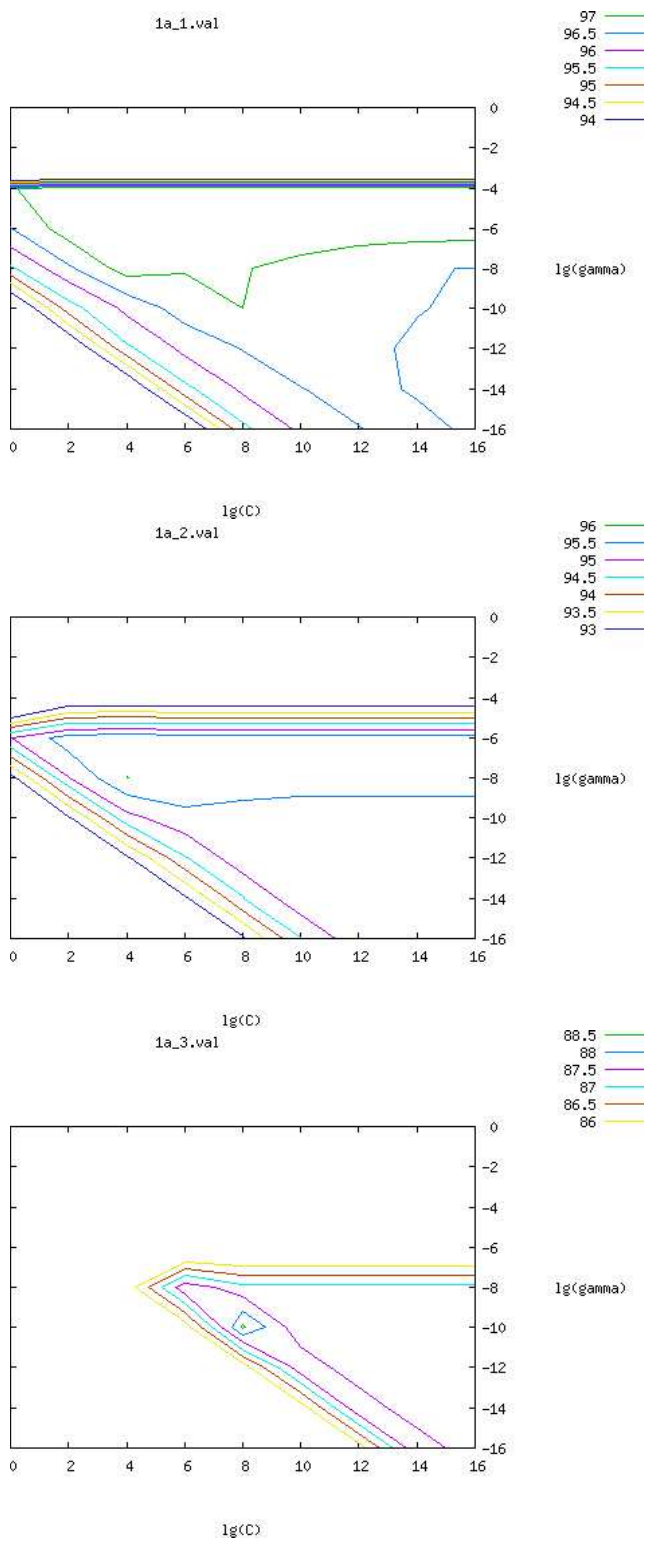

Figure 4.11: *Contour plot of the classification error using the 1a subset of the UNIPEN data. The error was calculated by five-fold cross-validation of 20 % of the original data. The figures correspond to the symbols of one, two, and three strokes respectively.*

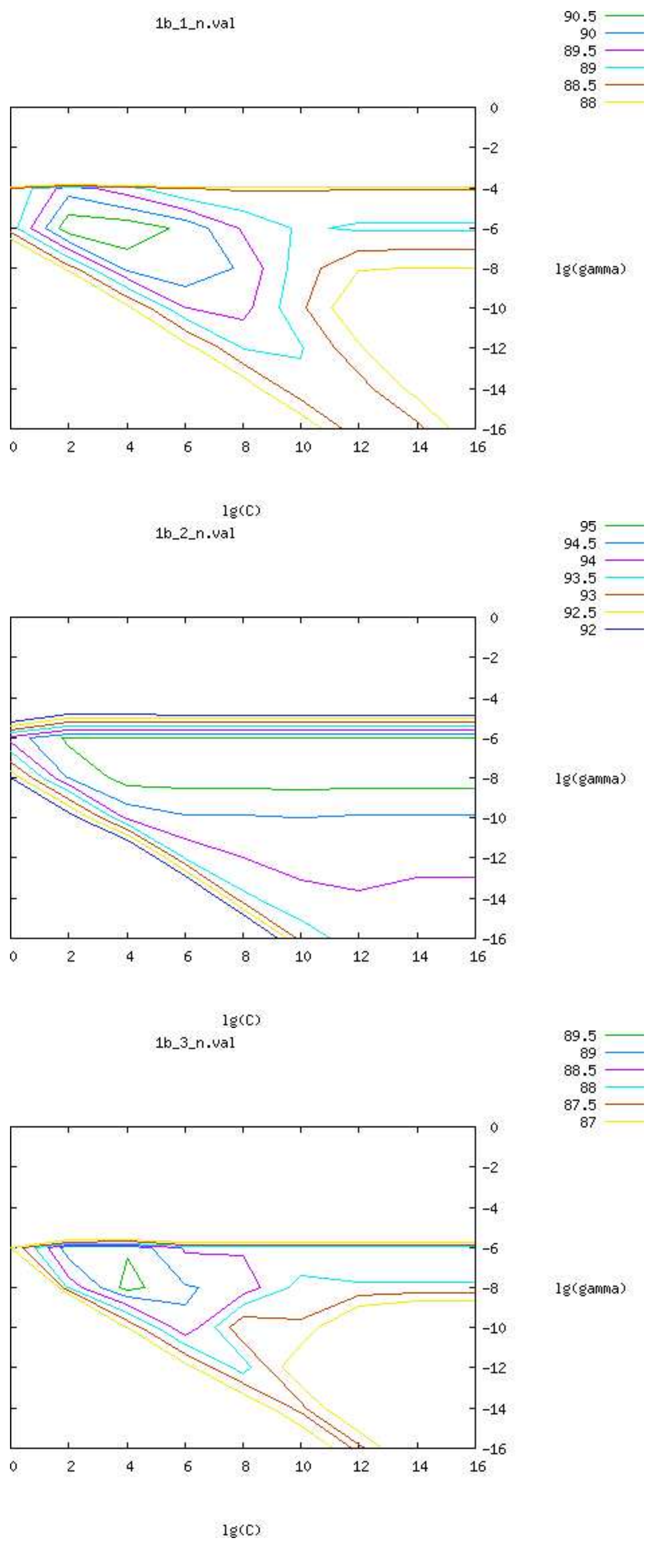

Figure 4.12: *Contour plot of the classification error using the 1b subset of the UNIPEN data.*

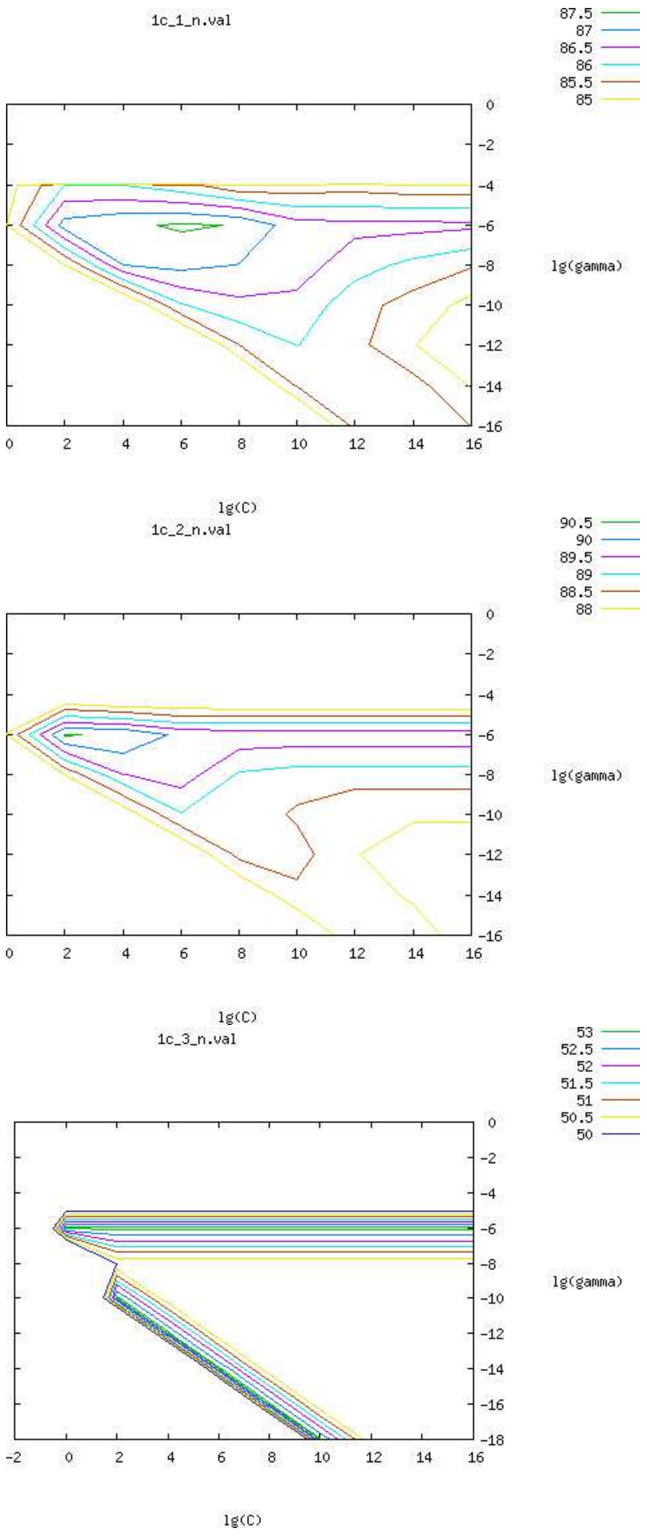

Figure 4.13: *Contour plot of the classification error using the 1c subset of the UNIPEN data.*

Table 4.4: *Optimal parameters of the RBF kernel for the UNIPEN database.*

Database	**Strokes**	C	γ
1a (digits)	1	64	0.015625
	2	16	0.00390625
	3	256	0.0009765625
1b (upper case)	1	16	0.015625
	2	16	0.00390625
	3	16	0.00390625
1c (lower case)	1	256	0.015625
	2	4	0.015625
	3	128	0.015625

parameter values of weka-2.4.2, except when training the *MultilayerPerceptron*. In this case, the dimension of the input data was reduced using principal component analysis, and 20 % of the corresponding training set was used for early stopping.

To get an idea of the effectiveness of our approach, we compared the results of all the classifiers we trained with other results from the literature that use the UNIPEN database. We should point out that, although these results where obtained using the UNIPEN database, symbol sets differ. Different authors use different versions of the database, different subsets of the database for tests, and training and tests are done on multi and omni-writer sets. For this reason, comparisons of classification errors should be made carefully. Tables 4.6-4.7 summarize the results obtained in this section.

Table 4.5: *Classification rates for the UNIPEN database using SVM-DAG classifiers with the RBF kernel.*

Database	Strokes	Symbols	Accuracy	Total accuracy	Total error
1a (digits)	1	11867	98.7191 %	98.6520 %	1.3480 %
	2	3943	98.8587 %		
	3	140	87.1429 %		
1b (upper case)	1	13509	94.0928 %	94.8299 %	5.1701 %
	2	10184	97.2407 %		
	3	4373	91.4932 %		
1c (lower case)	1	48194	91.6317 %	91.9455 %	8.0545 %
	2	9030	94.0532 %		
	3	260	76.9231 %		

Table 4.6: *Comparison of different classification approaches using the 1a (digits) subset. Approaches without references correspond to results presented in this chapter.*

Approach	Error	Database
Support Vector	1.3480 %	
Nearest neighbor	1.4799 %	
Neural network	2.5021 %	
Perceptron [64]	3.0000 %	DevTest-R02/V02
Hidden Markov Model [39]	3.2000 %	Train-R01/V06
Fuzzy-Geometric [37]	3.7000 %	
Decision tree	4.0070 %	
Naive Bayes	8.3841 %	

Table 4.7: *Classification rates for the subset 1a (digits) of the UNIPEN database using different classifiers.*

			Time rates		**Classification rates**		
Classifier	**Strokes**	**Symbols**	**Training**	**Test**	**Accuracy**	**Total accuracy**	**Total error**
	1	11867	187.54	0.56	96.1396 %		
Decision tree	2	3943	75.14	0.25	96.0183 %	95.9930 %	4.0070 %
	3	140	1.87	0.02	82.8571 %		
	1	11867	9.76	63.34	91.5964 %		
Naive Bayes	2	3943	5.28	44.73	91.9858 %	91.6159 %	8.3841 %
	3	140	0.25	1.85	82.8571 %		
	1	11867	-	9829.58	98.5587 %		
Nearest neighbor	2	3943	-	2370.98	98.6558 %	98.5201 %	1.4799 %
	3	140	-	3.86	91.4286 %		
	1	11867	2806.76	10.43	97.6062 %		
Neural network	2	3943	1346.16	8.61	97.7682 %	97.4979 %	2.5021 %
	3	140	10.05	0.1	80.7143 %		
	1	11867	236.1	39.327	98.7191 %		
Support vector	2	3943	64.348	9.31	98.8587 %	98.6520 %	1.3480 %
	3	140	1.337	0.102	87.1429 %		