SUPPORT VECTOR MACHINES FOR THE CLASSIFICATION OF WESTERN HANDWRITTEN CAPITALS

FUSI WANG, LOUIS VUURPIJL AND LAMBERT SCHOMAKER

Nijmegen Institute for Cognition and Information
P.O.Box 9104, 6500 HE Nijmegen, The Netherlands
E-mail: fswang(vuurpijl,schomaker)@nici.kun.nl
http://hwr.nici.kun.nl

In this paper, new techniques are presented using Support Vector Machines (SVMs) for multi-class classification problems. The issue of decomposing a N-class classification problem into a set of 2-class classification questions is discussed. In particular, the technique for normalizing the outputs of several SVMs is presented. Based on these techniques, support vector classifiers for the recognition of Western handwritten capitals are realized. Comparisons to several other classification methods are also presented.

1 Introduction

Support vector machines (SVMs) are primarily designed for 2-class classification problems[1]. Although SVMs achieve substantial improvements over the currently best performing methods and behave robustly over a variety of different learning tasks when a problem is treated as a binary classification problem[2,3], the application of SVMs to multi-class classification problems is still a challenge.

Whereas in theory, the combination of n SVMs can be used to solve a N-class ($N > 2$) classification problem, such a procedure requires some care when applied to practical problems[4]. In this paper, the issue of decomposing a N-class classification problem into a set of 2-class classification questions is discussed. For combining the output of a set of SVMs, it is required that their outputs are normalized. In this paper, we will address the normalization of SVMs' output, testing the proposed technique on the classification problem of simple bitmaps of Western handwritten capitals.

Also, the use of several other classification methods, such as the Nearest Neighbor (1NN), k-Nearest Neighbor (kNN), Hidden Markov Model (HMM) and Multi-Layer Perceptron (MLP) is discussed in this paper. The experiments are performed with handwritten isolated uppercase English characters which are extracted from the UNIPEN[5] data base and converted to pixel images.

In this paper, section 2 is concerned with the basic idea of the Support Vector Machine (SVM) and the problems faced to the SVM when it is applied

In: L.R.B. Schomaker and L.G. Vuurpijl (Eds.), Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition, September 11-13 2000, Amsterdam, ISBN 90-76942-01-3, Nijmegen: International Unipen Foundation, pp 167-176

to multi-class classification. In section 3, four other classifiers are briefly introduced. Sections 4 and 5 show experimental results and give a summary of our approach.

2 SVMs for multi-class classification

2.1 The Basic Idea of SVM

The Support Vector Machine is a classification technique which was developed at AT&T Bell laboratories by Vapnik and co-workers[1]. SVMs are trained to perform pattern recognition between two classes by finding a decision surface determined by a subset of the complete training set, termed *Support Vectors* (SV). For more details on SVMs, one can see for example the tutorials[6,7,8].

Given a 2-class training set $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_L, y_L)\}$, where $\mathbf{x}_i \in R^d$, $y \in \{+1, -1\}$ and $i = 1, ..., L$, the goal of training a SVM is to find the optimal hyperplane defining the decision boundary between the two classes. In general case, a separating hyperplane should satisfy:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \tag{1}$$

Where the ξ_i is a nonnegative variable introduced in the non-separable case, in the separable case $\xi_i = 0$. The pair $\{\mathbf{w}, b\}$ defines a separating hyperplane:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{2}$$

If we denote with $||\mathbf{w}||$ the Euclidean norm of $\mathbf{w}$, the distance $d(\mathbf{x}_i)$ of a point $\mathbf{x}_i$ from the separating hyperplane $(\mathbf{w}, b)$ is given by:

$$d(\mathbf{x}_i) = \frac{\mathbf{w} \cdot \mathbf{x}_i + b}{||\mathbf{w}||} \tag{3}$$

Since the distance of the closest point equals $1/||\mathbf{w}||$, the optimal separating hyperplane can be regarded as the solution of the convex optimization problem of:

$$\begin{array}{ll} \text{Minimize} & \frac{1}{2}\mathbf{w} \cdot \mathbf{w} + C \sum \xi_i \\ \text{Subject to} & y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \end{array}$$

The term $C \sum \xi_i$ can be thought of as some measure of the amount of misclassification, C is an penalty parameter. This problem can be solved by

means of the classical method of Lagrange multipliers[9], whose optimal solution pair $\{\mathbf{w}_o, b_o\}$ defines the so-called optimal separating hyperplane. The $\mathbf{w}_o$ can be written as

$$\mathbf{w}_o = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i \tag{4}$$

Here the feature vectors in set $N_{sv} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_l, y_l)\}$ are the so-called *Support Vectors*. Thus, we can get a linear classifier $f(\mathbf{x})$ for a binary classification problem of the form:

$$f(\mathbf{x}) = sgn(\mathbf{w}_o \cdot \mathbf{x} + b_o) \tag{5}$$

Non-linear SVMs are defined as linear separators in a high-dimensional space R^h in which the input space R^d is mapped through a non-linear mapping function $\phi(\mathbf{x})$. A non-linear classifier $f(x)$ for a binary classification problem has the form:

$$f(\mathbf{x}) = sgn(\sum_{i=1}^{l} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b_o) \tag{6}$$

where

$$K(\mathbf{x}_i, \mathbf{x}) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) \tag{7}$$

is a so-called kernel function which can be used to form arbitrarily complex decision surfaces.

2.2 SVM output normalization

SVM is well suited for binary classification problems, because the optimal hyperplane defines the decision surface between two classes. When SVM technique is applied to multi-class classification problems, usually the multi-class classification problem is decomposed into a set of 2-class classification problems[4]. The output class should be determined by choosing the maximum of the outputs of all sub-SVMs. However, before these outputs may be compared, they have to be normalized. This subsection describes a technique for normalizing the output of SVMs in a multi-class classification system.

The output $g(\mathbf{x})$ of a SVM, on the decision between two classes, is defined as:

$$g(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b_o \tag{8}$$

In the linear separable case, the mapping function $\phi(\mathbf{x}) = \mathbf{x}$. The Euclidean distance $d(\mathbf{x})$ of a point $\mathbf{x}$ from the separating hyperplane $g(\mathbf{x}) = 0$ is given by:

$$d(\mathbf{x}) = \frac{g(\mathbf{x})}{||\mathbf{w}_o||} \tag{9}$$

The $\mathbf{w}_o$ can be obtained by applying $\phi(\mathbf{x})$ of the support vectors.

$$\mathbf{w_o} = \sum_{i=1}^{l} \alpha_i y_i \phi(\mathbf{x}_i) \tag{10}$$

The Euclidean norm $||\mathbf{w}_o||$ of $\mathbf{w}_o$ can be calculated by

$$\frac{1}{\pi_1^2} = ||\mathbf{w}_o||^2 = \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{11}$$

To make it possible to compare the distance output of different SVMs, for each SVM we define the scaling factor π_2 as the mean over the positive set. That is

$$\frac{1}{\pi_2} = \frac{1}{p} \sum_{i=1}^{p} g(\mathbf{x}_i)\pi_1 \tag{12}$$

Here the p is the number of positive samples in training set. Finally, the comparable distance output of one SVM can be get by

$$s(\mathbf{x}) = g(\mathbf{x})\pi_1\pi_2 \tag{13}$$

2.3 Decomposing N-class classification problem

Among the wide variety of methods available in the literature to learn classification problems, some are able to handle many classes, while others are specific to 2-class problems. Traditionally, when the latter are used to solve N-class classification problems, N-classifiers are typically trained to separate

one class from the N-1 others. The same idea can be applied with SVMs. This way of decomposing a general classification problem into 2-class problems is known as a $one-against-others$[4] decomposition, and is independent of the learning method used to train the classifiers.

Suppose we need to classify an unknown sample into a set of classes $\{c_1, ..., c_N\}$, given the N-classes training set: $S = \{\mathbf{x}_1, ..., \mathbf{x}_V\}$, divide it into N subsets $\{S_1, ...S_N\}$, where S_i contains all training samples belong to c_i. The i-th SVM is trained in the set of $\{PE, NE\}$, where $PE = S_i$ and $NE = S - S_i$. If $s_i(\mathbf{x})$ is the normalized output of the i-th SVM, which is described in section 2.2, then the unknown example $\mathbf{x}$ will be classified into classes c_j if:

$$f_j(\mathbf{x}) = argmax(s_i(\mathbf{x})) \quad i = 1, ..., N \tag{14}$$

If there are not too many classes, the $pairwise-coupling$ decomposition scheme can be used to replace the $one-against-others$. In this scheme one classifier is trained to discriminate between each pair of classes, ignoring the other classes. This schemes make it possible to use the bubble sort algorithm to get out the final category.

3 Description of the other classifiers used

For comparing the performance of the SVM to other classifiers, the Nearest Neighbor (1NN), k-Nearest Neighbor (kNN), Hidden Markov Model (HMM) and Multi-Layer Perceptron (MLP) are introduced in this section.

3.1 kNN and 1NN method

For an unknown sample $\mathbf{x}$, the kNN follows the following principle. Search the k nearest neighbors[10] of $\mathbf{x}$ from V samples in the training set using an appropriate distance measure. Suppose there are N categories in the given samples, k_i samples in the k nearest neighbor of $\mathbf{x}$ coming from category c_i, now, if

$$k_j = max(k_i); \quad i = 1, ..., N; \tag{15}$$

then $\mathbf{x}$ can be classified to class c_j. This is the basic rule of kNN. When $k = 1$, the output is the nearest neighbor of $\mathbf{x}$. kNN becomes the nearest neighbor (1NN) method in this case. For a large training set, the kNN method implements the Bayes rule, and furthermore kNN is very easy to implement. This is why kNN has become one of the most important pattern recognition

methods. The major problem of kNN is that all training samples have to be available during classification, and for a unknown pattern $\mathbf{x}$, the distance of $\mathbf{x}$ to all samples has to be computed. This requires a lot of memory and time resources. In references[10,11] there are some algorithms to speed up a kNN classifiers.

3.2 MLP method

As a representative of neural network classifiers, the multi-layer perceptron[12] (MLP) is used in our approach. To design a MLP classifier, two problems have to be solved. The first is what kind of neural network structure should be adopted. For instance, how many hidden layers, how many nodes in each layer and what kind of activation function should be used for each node. Up to now there is no theory which can guide one to yield the optimal structure of a neural network.

The second problem is how to choose a learning algorithm for the neural network. Statistical methods can reach the global minimum point of a MLP and yield the best performance, but this learning method needs a long training time. Back-propagation is applied widely, but it may be plagued by the local minima problem.

Our MLP classifier uses the Back-propagation learning algorithm and a $256 * 128 * 32 * 26$ neural network. The input layer represent the image of the input characters and is fully connected to the hidden units in the second layer. All units in the classifier have the same transfer function: They sum their inputs, add a constant b called bias, and take a fixed function f, called an activation function, of the result. The activation function of all hidden and output layer units is the sigmoid function.

3.3 HMM method

Hidden Markov Models (HMMs) a popular method of statistical representation in speech processing is based on the representation of an object as a random process that generates a sequence of states. The elements of a HMM can be described as follows[13,14]:

- N : the number of states
- M : the number of output symbols
- T : the number of observations
- $Q = \{q_t\}$: the set of states, $t = \{1, 2, ..., N\}$
- $V = \{Vk\}$: the set of output symbols, $k = \{1, 2, ..., M\}$

- $A = \{A_{ij}\}$: the transition matrix of the underlying Markov chain. Here A_{ij} is the probability of transition to state j given the current state i.
- $B = \{B_j(V_k)\}$: the model output symbol probability matrix, where $B_j(V_k)$ is the probability of output symbol V_k given the state q_j.
- $a = \{a_i\}$: the initial probability vector, $i = 1, 2, ..., N$

The character in this classifier is scanned vertically and horizontally to generate the corresponding out-contour profiles. The discrete hidden Markov models are generated using the Segmental k-means algorithm[13] while a scoring mechanism based on the Viterbi algorithms[13] is used in test phase. Some parameters of the HMM classifier are $N = 8$, $M = T = 16$.

4 Experimental results

We have extracted 14967 uppercase isolated English characters from the UNIPEN data base of the NICI. 5200 were used as training set (200 for each letter), the rest was used as test set. This is a very difficult data set, with characters written by many writers, in a wide range of tablets and systems. The raw data of each isolated letter is converted into a 16x16 bitmap image. The pen width is 2 pixels. Examples of test data can be observed in figure 1. The raw 256 dimensional bitmap vector is used as the input for each classifier. All classifiers are programmed in the C programming language, running on a Pentium-II 400Hz PC.

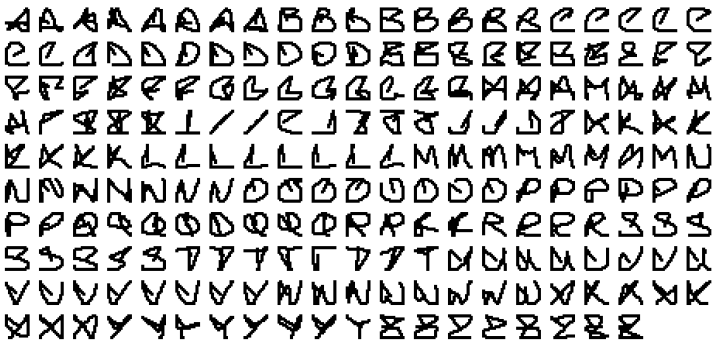

Figure 1. *Some bitmap images of testing examples.*

We designed some experiments to compare the performances of each method. In the first experiment, we test the training time as a function

of the number of samples. There are 26 classes in these experiments. The result is listed in Table 1. The 1NN and kNN classifiers are not listed in this table, because they do not need to be trained as the prototype set is the whole training set.

Table 1. training time in seconds as a function of the number of training samples

S_N	130	260	650	1300	2600	5200
HMM	30	57	133	262	519	1046
MLP	81	283	693	1422	6062	1561
SVC_2	7	15	67	123	960	8075
SVC_1	4	6	12	28	204	67628

We use a polynomial degree 2 kernel function for SVM classifier. The $one-against-others$ category is used when training SVC_1. The $pairwise-coupling$ decomposition scheme is used when training SVC_2. From Table 1 we can deduce that there is little difference between methods when the number of training samples is small. When the number of samples increases, the HMM classifier linearly increases its training time, whereas the SVC's training time increases dramatically.

Table 2. Test set correct recognition rate in percentages.

	SVC_2	SVC_1	HMM	kNN	$1NN$	MLP
top_1	81.2	79.6	78.4	77.4	76.9	76.9
top_2	89.0	86.2	86.5	86.7	87.4	85.2
top_3	92.1	88.6	89.8	88.9	91.1	88.4
top_4	93.9	90.2	91.9	89.5	93.2	90.2
top_5	95.1	91.2	93.2	89.6	94.5	91.4

The second experiment tests the recognition rate. The experimental results are shown in Table 2. In these experiments, the polynomial degree 2 kernel function and a penalty parameter 1000 are used for all SV classifiers. The SVC_2 uses the $pairwise-coupling$ decomposition scheme. The $one-against-others$ decomposition scheme is used in the SVC_1. In the SVC_1 all SVMs outputs are normalized as described in section 2.2. For the SVC_2 the bubble sort algorithm was used to sort the letters in the candidate set.

The pairwise-coupling decomposition scheme is more efficient than one-against-others as can be seen in table 2. Although we need to train more

sub-SVMs, In the case of large training set sizes, it takes less training time. This is because in the pairwise-coupling decomposition scheme each sub-SVM uses less training samples as they are coming from two classes only. In the one-against-others case each SVM is trained with data originating from all classes.

In the recognition procedure, if we focus only on the top_1 result, the pairwise-coupling decomposition scheme takes less time. If we need a candidate set, the SVC_2 recognition time will increase quadratic in the number of classes. This can be observed in table 3.

Table 3. Recognition time in seconds for the classification of 9767 samples.

	SVC_2	SVC_1	HMM	kNN	$1NN$	MLP
Need only top_1	284	692	227	523	522	46
Need top_5	4397	702	240	572	542	54

From these observations we can conclude that the SVC_2 is more robust than the other classifiers.

5 Conclusions

In this paper, a technique is introduced for using support vector machines for multi-class classification. A solution for the problem of normalizing the output of several SVMs is highlighted. Different decomposition techniques are discussed. The result of several experiments are presented. The classification performances are not so high because of the difficult data and crude feature vectors. In contrast to the conventional classification methods, when considering training and recall times, support vector classifiers are considerably slower. However when considering recognition rates, the pair-wise SVC proves to outperform all other classifiers. Our future work will be focused on the improvement of the feature vectors and the combination of SVC_2 with the other classifiers.

References

1. V. Vapnik. *The nature of statistical learning theory.* Springer-Verlag, Berlin, 1995.
2. D.M.J. Tax, R.P.W. Duin, and M. van Breukelen. Support vector classifiers: A first look. In H.E. Bal, H. Corporaal, P.P. Jonker, and J.F.M.

Tonino, editors, *ASCI'97, Proc. 3rd Annual Conference of the Advanced School for Computing and Imaging*, pages 253–258, Delft, 1997.
3. T. Joachims. Making large-scale svm learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*. MIT – Press, 1999.
4. Ethem Alpaydm Eddy Mayoraz. Support vector machines for multi-class classification. Technical report, IDIAP, 1998.
5. I. Guyon, L. Schomaker, R. Plamondon, and S. Liberman, R.and Janet. Unipen project of on-line data exchange and recognizer benchmarks. In *Proceedings of the 12th International Conference on Pattern Recognition, ICPR'94*, pages 29–33, Jerusalem, Israel, October 1994. IAPR-IEEE.
6. Christopher C.J. Burgess. A tutorial on support vector machines for pattern recognition. available at `http://svm.research.bell-labs.com`, 1998.
7. Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. Technical Report NC2-TR-1998-030, GMD, October 1998.
8. Massimiliano Pontil and Alessandro Verri. Properties of support vector machines. Technical report, Massachusetts Institute of Technology Artificial Intelligence Laboratory, 1997.
9. M Bazaraa and C.M. Shetty. *Nonlinear programming*. Jon Wiley, New York, 1979.
10. D. Mangalagiu and M. Weinfeld. Tree search technique for the optimization of the k nearest neighbors algorithm. In *Proceedings of the 6th IWFHR*, Taejon, Korea, August 1998.
11. L. Schomaker, D. Mangalagiu, L. Vuurpijl, and M. Weinfeld. Two tree-formation methods and fast pattern search using nearest neighbour and nearest-centroid matching. Technical report, NICI, Nijmegen University, The Netherlands, 2000.
12. C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995. ISBN 0 19 853864 2.
13. Rakesh Dugad and U. B. Desai. A tutorial on hidden markov models. Technical report, Department of Electrical Engineering, Indian Institute of Technology – Bombay, 1996.
14. Sang Kyoon Kim Hang Joon Kim, Kyung Hyun Kim and Jong Kook Lee. On-line recognition of handwritten chinese characters based on hidden markov models. *Pattern Recognition*, 30(9):1489–1500, 1997.