Fast Training of Multi-Class Support Vector Machines
B. Smith, R. Gosine, G. Mann
C-CORE and Memorial University of Newfoundland

Abstract
In this paper, we introduce a new approach to using binary support vector machines for multi-class object recognition. The proposed method is a hierarchical classifier utilizing hypercubes and informed search to coarsely partition the input space. Results are shown using a dataset of 100 objects. The method is compared with the one-against-one approach. Training times are shown to be significantly reduced, while test performance is comparable.

1 Introduction
A Support Vector Machine (SVM) [1], is a binary pattern classifier developed around the idea of minimizing the expected risk as a function of the amount of data that is available and the complexity of the classifier. Matching the complexity of the classification function to the size of the available data gives optimal generalization performance. To discriminate between non-linearly separable data in input space, a SVM maps the data to some high dimensional feature space in which the data are linearly separable. The linear decision surface in this feature space corresponds to a non-linear decision surface in the original input space. A SVM finds the hyperplane \( \langle \vec{w}, \vec{x} \rangle + b \) in feature space which separates two classes with the most generality. Here \( \langle \vec{w}, \vec{x} \rangle \) denotes the dot product, \( \sum_i w_i x_i \). The hyperplane is composed of \( \vec{w} \) a weight vector, \( \vec{x} \) a sample vector, and \( b \) a bias term. The best hyperplane is the one which maximizes the distance or margin between the two classes.

Methods have been devised to extend binary SVMs to multi-class classification. Practical methods are based on the sequential evaluation of many binary SVMs including the one-against-all (OAA) [7], one-against-one (OAO) [4] and decision directed acyclic graph SVM (DAGSVM) [6] methods. Only the OAO method will be considered here.

Although the OAO Max Wins strategy has high classification rates, it suffers from several deficiencies. These include long training times and large memory requirements [3]. In addition, some classifications can be returned with no classification result. This can happen when the decision functions all give one vote to each class, indicating that the new point belongs to each class. Another shortcoming is that all the decision surfaces may not be needed, as in the case when classes are linearly aligned [4]. The redundant decision surfaces add computational complexity, but no classification advantage.

The proposed is a hierarchical classifier to build on the weaknesses of existing methods. It uses an iterative bisection of the input space axes to coarsely discriminate between different classes of data. Binary SVMs are then applied to the new subspaces formed as a result of this bisection. We conduct an experiment using 100 objects to compare the classifier to the OAO method. The proposed method is shown to have comparable classification rates and times, and over 20 fold reduction in training times compared with OAO.

2 Methodology
Existing methods for multi-class SVMs proceed by applying SVM classifiers directly to all the data in the input space. This often needlessly incorporates large amounts of data to determine the classification boundaries and results in the creation of redundant boundaries. In a general multi-class application, some of the data classes may be intermixed in a complex way, but other data classes may be sparsely separated. For example, in Fig. 2, the circles (C) and squares (S) are located in very close proximity and may require a complex decision surface to separate. The diamonds (D), however, are located far away from both of these. Using the OAO approach to separate the data results in the consideration of three pairs of data (D-C, D-S and C-S) to construct 3 hyperplanes. Inspection of Fig. 2 reveals that the diamonds are located far enough away from the other two classes that using SVMs to distinguish between them is not necessary. Simply partitioning the input space requires much less computing time and resources. A single binary SVM classifier is then all that is needed to correctly partition the data.
Figure 1: Application of binary SVMs to classify the given data according to (a) the OAO approach, and (b) by partitioning the data by bisecting each axis in the input space.

Solely using this partitioning technique to distinguish between different data patterns can result in a very deep tree of partitioned subspaces. This gives long test times, due to the deep traversing of the tree that must take place, and an overfitted discrimination function, due to the overwhelming subspace splitting that can result if many patterns are complexly intermixed. To reduce the number of levels in this tree, a binary SVM is applied at various points in the tree according to the training rules.

There are three situations that can arise during training. A given subspace can contain points from one, two, or more than two different classes. If a given subspace contains points from only one class, then any test points lying in this cube will belong to that class. If a given subspace contains data points of more than two classes, then it is split into smaller subspaces. If a given subspace contains data points from exactly two classes, then a binary SVM is applied to this subspace to separate the classes.

The training samples determine the size of the root hypercube. The extreme values, minimum and maximum, of each dimension determine the length of that dimension. The maximum, $q_i$, of each axis, $j$, is given by $q_j = \max_{i=1,\ldots,L}(\vec{x}_{ij})$, while the minimum, $p_i$, of each axis, $j$, is given by $p_j = \min_{i=1,\ldots,L}(\vec{x}_{ij})$, where $\vec{x}_{ij}$ denotes the $j^{th}$ value of the training sample $\vec{x}_i$, and $L$ is the number of training samples.

The hypercube is divided up by successively bisecting each axis such that each child hypercube has data from at most two classes. The training vector, $x$, with associated label, $c_i$, helps to train the classifier in the following way:

**Training Rule 1**

$$\text{train}(H^k_i) = \{c_i\} \quad \text{if } (A1 \lor A6) \land A3$$

**Training Rule 2**

$$\text{train}(H^k_i) = \{c_i, c_l\} \quad \text{if } (A2 \land c_l \in C_j \land A3)$$

**Training Rule 3**

$$\text{train}(H^k_i) = {} \quad \text{if } |C_j| > 2 \land A3 \land |G = \text{split}(H^k_j)| \geq 2 \land |\text{train}(G)| \leq 2$$

$C$ is the set of all classes, $C_i$ is the set of all classes within hypercube $i$, $H^k$ is the set of $k$ ranges that defines the input space, $H^k_i$ is the set of $k$ ranges that defines the input space of hypercube $i$, $D^k$ is the set of all $k$-dimensional data points and corresponding class label, and $D^k_i$ is the set of all $k$-dimensional data points and corresponding class label within hypercube $i$. Additionally, the functions that operate on these sets are:

$$A1(i, j) = \begin{cases} 1 & \text{if } (c_i \in C_j) \land (|C_j| = 1) \\ 0 & \text{otherwise} \end{cases}$$

$$A2(i, j) = \begin{cases} 1 & \text{if } (c_i \in C_j) \land (|C_j| = 2) \\ 0 & \text{otherwise} \end{cases}$$

$$A3(i, j) = \begin{cases} 1 & \text{if } x \cap H^k_j = x \\ 0 & \text{otherwise} \end{cases}$$

$$A4(i, j) = \begin{cases} 1 & \text{if } \text{svm}(x, c_i, D^k_j) > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$A5(i, j) = \begin{cases} 1 & \text{if } \text{arg min}_{j \neq j}(\text{dist}(H^k_j, H^k_j)) \\ 0 & \text{otherwise} \end{cases}$$
bool EvaluationFunction(Test Point, Current Hypercube)
{
    if (Test Point is in Current Hypercube)
        return true
    else
        return false
}

Figure 2: Evaluation Function for the Hypercube Tree Search

\[ A6(i,j) = \begin{cases} 1 & \text{if } |C_j| = 0 \\ 0 & \text{otherwise} \end{cases} \]  \hspace{1cm} (6)

where \( \text{svm}(x, c_i, D_j^k) \) is the binary SVM classification of \( x \) using the two classes of hypercube \( j \) with class \( i \) being designated as being on the positive side of the decision surface, \( \text{dist}(H_I^k, H_J^k) \) is the shortest distance between hypercube \( I \) and hypercube \( J \), and \( \text{split}(H_J^k) \) bisection the hypercube, \( H_J^k \) into a set of child hypercubes, \( G \). Note that the indices, \( i \) and \( j \), are assumed for each function, except where explicitly stated otherwise.

After the input space is partitioned and the training rules are applied, the classifier is ready to classify unknown data. To do this the smallest hypercube that encompasses the data point is sought. Various methods can be used for this search. The method chosen is the informed search method. The classification of a vector, \( x \), is \( c_i \) if any of the following Testing Rules are true:

Testing Rule 1
\((A1 \land A3)\)

Testing Rule 2
\((A2 \land A3 \land A4)\)

Testing Rule 3
\(((A3 \land A5 \land A6) \land (A1(i,l) \lor (A2(i,l) \land A4(i,l))))\)

A simple interpretation of the testing rules is: (Testing Rule 1) if there is only one class in a hypercube then that is the classification result, (Testing Rule 2) if there are two classes in a hypercube then apply the SVM decision function to determine the classification result, (Testing Rule 3) if there are no classes in a hypercube then find the closest sibling hypercube and apply the testing rules to it to determine the classification of the unknown point.

The objective of the search method is to get to the lowest leaf node of the hypercube tree that contains the test sample. Any of the basic search methods can be used to accomplish this, but informed search can be used with higher performance results. Breadth-First Search will search each hypercube at each level of the tree until it finds the hypercube containing the data point or reaches the bottom of the tree. Depth-First Search will search each hypercube to its leaves until all hypercubes have been searched or the hypercube containing the data point is found. A better search method to use in this case would be an informed search technique. It is simple and inexpensive to test if a hypercube contains the given point, just check that the point lies within the boundaries of the hypercube. Employing this knowledge results in the evaluation function shown in Fig. 2 which reduces the searching considerably.

To illustrate the differences between the three search methods, the number of nodes needed to reach a given point in the tree is counted for each method. Fig. 3(a) shows how breadth-first search would locate a node. The nodes of all previous depths of the tree will be expanded before the current depth is searched. This results in 38 nodes being searched. Fig. 3(b) shows how depth-first search would locate a node. The children of the current node are searched before the sibling nodes. This results in 25 nodes being searched. Fig. 3(c) shows how the best-first search would locate a node. For each depth, only the best node is expanded. This results in only 5 nodes being searched, which reduces the search time considerably.
3 Experimental Results

The purpose of this experiment is to compare how the classifier described in this paper performs against the OAO multi-class SVM method when a large number of classes are involved. To do this in an unbiased manner, LIBSVM [2] is used to compute the binary SVMs for both multi-class methods. Thus the difference in results will be due to the multi-class method and not the implementation of the underlying binary SVMs. Experiments exploring the effect of varying the number of classes and number of training images per class is conducted. The performance criteria evaluated includes training time, testing time and classification rate.

The dataset used for this experiment is the Columbia Object Image Library (COIL) [5]. The COIL database consists of images of 100 objects with 72 images per object. Each of the images have been converted to 8-bit grayscale of size 32x32 pixels. A subset of the various objects are shown in Fig. 4.

For these experiments, the number of classes are varied from 10 to 100 in increments of 10. In each of these, the number of training images per class are varied as well, but for the purpose of this experiment, the results using 25 training images per class are discussed. The effect of increasing the number of classes involved has a pronounced effect on the training time of the OAO classifier. As the number of classes are increased the training time rapidly increases in quadratic fashion. This is expected since $N(N-1)/2$ binary SVM classifiers are created for $N$ classes. The proposed classifier also increases quadratically, but at a much lesser rate. This is shown in Fig. 5(a). The differences between the classifiers are much less pronounced for the test times, with the OAO being slightly faster as shown in Fig. 5(b). Both classifiers have similar results for the classification rate. As the number of classes increases with the number of training images per object fixed at 25, classification rates decrease somewhat but overall appear fairly constant. The classification rate of the OAO is lower, by at least 5 percent, in each case. This trend is shown in Fig. 5(c).

The second experiment explores the effect of increasing the number of training images per class on the training time, test time and classification rate. The number of images per class are increased in the following order: 2, 5, 10, 15, 20, 25, 30, 35. The results using 80 objects are arbitrarily chosen to be discussed, as the trends are similar for the different numbers of classes. As the number of training images per class is increased, so too does the amount of time needed to train both classifiers. The time taken to train the OAO classifier increases quadratically as does the time taken to train the proposed classifier. The time taken to train the classifier developed here increases at a much lower rate. This is evident from when examining Fig. 5(d). The test times for both classifiers increase slowly with increasing number of training images per class, as shown in Fig. 5(e). The OAO is slightly faster at classifying an unknown sample. As the number of training images per class is increased, the classification rate for both the classifiers increases in the usual manor. This is shown in Fig. 5(f).

4 Conclusions and Future Work

In this paper a new method for multi-class SVMs is proposed which reduces the training time significantly from commonly used methods while still being easy and straight-forward enough for anyone with some
Figure 5: The effect of increasing the number of classes on the (a) training times, (b) test times and (c) classification rate, and the effect of increasing the number of training images per class on the (d) training times, (e) test times and (f) classification rate. The dotted line represents the results using the OAO approach. The solid line represents the results using the method described in this paper.

knowledge in software development to implement. The experiments conducted compare the performance of the proposed classifier against the performance of the commonly used OAO classifier. Test time and classification rate are comparable for each classifier, with OAO having slightly better test times and the proposed method having slightly better classification rates. The greatest contribution here is that the proposed classifier has greatly reduced training times as compared to the OAO.

Future work includes speeding up the classification time by computing only partial distances and enabling the user to set the number of levels in the hypercube tree. Also, comparisons with other multi-class SVM methods and general multi-class methods will be conducted.

References


