A Multi-class SVM Classifier Utilizing Binary Decision Tree

Gjorgji Madzarov, Dejan Gjorgjevikj and Ivan Chorbev
Department of Computer Science and Engineering
Faculty of Electrical Engineering and Information Technology
Karpos 2 b.b., 1000 Skopje, Macedonia
E-mail: madzarovg@feit.ukim.edu.mk

Keywords: Support Vector Machine, multi-class classification, clustering, binary decision tree architecture

Received: July 27, 2008

In this paper a novel architecture of Support Vector Machine classifiers utilizing binary decision tree (SVM-BDT) for solving multiclass problems is presented. The hierarchy of binary decision subtasks using SVMs is designed with a clustering algorithm. For consistency between the clustering model and SVM, the clustering model utilizes distance measures at the kernel space, rather than at the input space. The proposed SVM based Binary Decision Tree architecture takes advantage of both the efficient computation of the decision tree architecture and the high classification accuracy of SVMs. The SVM-BDT architecture was designed to provide superior multi-class classification performance. Its performance was measured on samples from MNIST, Pendigit, Optdigit and Statlog databases of handwritten digits and letters. The results of the experiments indicate that while maintaining comparable or offering better accuracy with other SVM based approaches, ensembles of trees (Bagging and Random Forest) and neural network, the training phase of SVM-BDT is faster. During recognition phase, due to its logarithmic complexity, SVM-BDT is much faster than the widely used multi-class SVM methods like "one-against-one" and "one-against-all", for multiclass problems. Furthermore, the experiments showed that the proposed method becomes more favourable as the number of classes in the recognition problem increases.

Povzetek: Predstavljena je metoda gradnje binarnih dreves z uporabo SVM za večrazredne probleme.

1 Introduction

The recent results in pattern recognition have shown that support vector machine (SVM) classifiers often have superior recognition rates in comparison to other classification methods. However, the SVM was originally developed for binary decision problems, and its extension to multi-class problems is not straightforward. How to effectively extend it for solving multi-class classification problem is still an on-going research issue. The popular methods for applying SVMs to multi-class classification problems usually decompose the multi-class problems into several two-class problems that can be addressed directly using several SVMs.

For the readers' convenience, we introduce the SVM briefly in section 2. A brief introduction to several widely used multi-class classification methods that utilize binary SVMs is given in section 3. The Kernel-based clustering introduced to convert the multi-class problem into SVM-based binary decision-tree architecture is explained in section 4. In section 5, we discuss related works and compare SVM-BDT with other multi-class SVM methods via theoretical analysis and empirical estimation. The experimental results in section 6 are presented to compare the performance of the proposed SVM-BDT with traditional multi-class approaches based on SVM, ensemble of decision trees and neural network. Section 7 gives a conclusion of the paper.

2 Support vector machines for pattern recognition

The support vector machine is originally a binary classification method developed by Vapnik and colleagues at Bell laboratories [1][2], with further algorithm improvements by others [3]. For a binary problem, we have training data points: $\{\mathbf{x}_i, y_i\}$, $i=1,...,l$, $y_i \in \{-1, 1\}$, $\mathbf{x}_i \in \mathbf{R}^d$. Suppose we have some hyperplane which separates the positive from the negative examples (a "separating hyperplane"). The points $\mathbf{x}$ which lie on the hyperplane satisfy $\mathbf{w}\cdot\mathbf{x} + b = 0$, where $\mathbf{w}$ is normal to the hyperplane, $|b|/||\mathbf{w}||$ is the perpendicular distance from the hyperplane to the origin, and $||\mathbf{w}||$ is the Euclidean norm of $\mathbf{w}$. Let d_+ (d_-) be the shortest distance from the separating hyperplane to the closest positive (negative) example. Define the "margin" of a separating hyperplane to be $d_+ + d_-$. For the linearly separable case, the support vector algorithm simply looks for the separating hyperplane with largest margin. This can be formulated as follows: suppose that all the training data satisfy the following constraints:

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 \text{ for } y_i = +1, \quad (1)$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 \text{ for } y_i = -1, \quad (2)$$

These can be combined into one set of inequalities:

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \quad \forall i, \quad (3)$$

Now consider the points for which the equality in Eq. (1) holds (requiring that there exists such a point) is equivalent to choosing a scale for w and b. These points lie on the hyperplane H_1: $\mathbf{x}_i \cdot \mathbf{w} + b = 1$ with normal $\mathbf{w}$ and perpendicular distance from the origin $|1-b|/\|\mathbf{w}\|$. Similarly, the points for which the equality in Eq. (2) holds lie on the hyperplane H_2: $\mathbf{x}_i \cdot \mathbf{w} + b = -1$, with normal again $\mathbf{w}$ and perpendicular distance from the origin $|-1-b|/\|\mathbf{w}\|$. Hence $d_+ = d_- = 1/\|\mathbf{w}\|$ and the margin is simply $2/\|\mathbf{w}\|$.

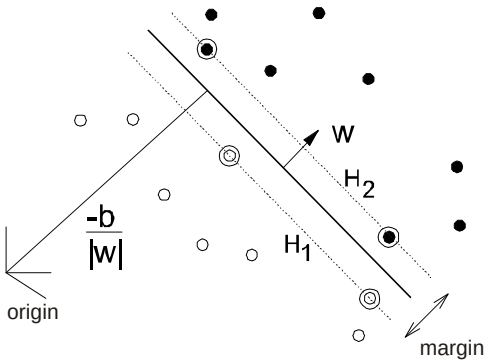

Figure 1 – Linear separating hyperplanes for the separable case. The support vectors are circled.

Note that H_1 and H_2 are parallel (they have the same normal) and that no training points fall between them. Thus we can find the pair of hyperplanes which gives the maximum margin by minimizing $\|\mathbf{w}\|^2$, subject to constraints (3).

Thus we expect the solution for a typical two dimensional case to have the form shown on Fig. 1. We introduce nonnegative Lagrange multipliers α_i, $i = 1,..., l$, one for each of the inequality constraints (3). Recall that the rule is that for constraints of the form $c_i \geq 0$, the constraint equations are multiplied by *nonnegative* Lagrange multipliers and subtracted from the objective function, to form the Lagrangian. For equality constraints, the Lagrange multipliers are unconstrained. This gives Lagrangian:

$$L_p \equiv \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_{i=1}^{l} \alpha_i, \quad (4)$$

We must now minimize L_p with respect to $\mathbf{w}$, b, and maximize with respect to all α_i at the same time, all subject to the constraints $\alpha_i \geq 0$ (let's call this particular set of constraints C_1). Now this is a convex quadratic programming problem, since the objective function is itself convex, and those points which satisfy the constraints also form a convex set (any linear constraint defines a convex set, and a set of N simultaneous linear constraints defines the intersection of N convex sets, which is also a convex set). This means that we can equivalently solve the following "dual" problem: *maximize* L_P, subject to the constraints that the gradient of L_P with respect to w and b vanish, and subject also to the constraints that the $\alpha_i \geq 0$ (let's call that particular set of constraints C_2). This particular dual formulation of the problem is called the Wolfe dual [4]. It has the property that the maximum of L_P, subject to constraints C_2, occurs at the same values of the w, b and α, as the minimum of L_P, subject to constraints C_1.

Requiring that the gradient of L_P with respect to w and b vanish gives the conditions:

$$w = \sum_i \alpha_i y_i \mathbf{x}_i, \quad (5)$$

$$\sum_i \alpha_i y_i = 0. \quad (6)$$

Since these are equality constraints in the dual formulation, we can substitute them into Eq. (4) to give

$$L_D \equiv \sum_i \alpha_i - \frac{1}{2}\sum_{i,j}^{l} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j, \quad (7)$$

Note that we have now given the Lagrangian different labels (P for primal, D for dual) to emphasize that the two formulations are different: L_P and L_D arise from the same objective function but with different constraints; and the solution is found by minimizing L_P or by maximizing L_D. Note also that if we formulate the problem with $b = 0$, which amounts to requiring that all hyperplanes contain the origin, the constraint (6) does not appear. This is a mild restriction for high dimensional spaces, since it amounts to reducing the number of degrees of freedom by one.

Support vector training (for the separable, linear case) therefore amounts to maximizing L_D with respect to the α_i, subject to constraints (6) and positivity of the α_i, with solution given by (5). Notice that there is a Lagrange multiplier α_i for every training point. In the solution, those points for which $\alpha_i > 0$ are called "support vectors", and lie on one of the hyperplanes H_1, H_2. All other training points have $\alpha_i = 0$ and lie either on H_1 or H_2 (such that the equality in Eq. (3) holds), or on that side of H_1 or H_2 such that the strict inequality in Eq. (3) holds. For these machines, the support vectors are the critical elements of the training set. They lie closest to the decision boundary; if all other training points were removed (or moved around, but so as not to cross H_1 or H_2), and training was repeated, the same separating hyperplane would be found.

The above algorithm for separable data, when applied to non-separable data, will find no feasible solution: this will be evidenced by the objective function (i.e. the dual Lagrangian) growing arbitrarily large. So how can we extend these ideas to handle non-separable data? We would like to relax the constraints (1) and (2), but only

when necessary, that is, we would like to introduce a further cost (i.e. an increase in the primal objective function) for doing so. This can be done by introducing positive slack variables e_i; $i = 1,..., l$, in the constraints, which then become:

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 - e_i \text{ for } y_i = +1, \quad (8)$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 + e_i \text{ for } y_i = -1, \quad (9)$$

$$e_i \geq 0 \forall i. \quad (10)$$

Thus, for an error to occur, the corresponding e_i must exceed unity, so $\Sigma_i e_i$ is an upper bound on the number of training errors. Hence a natural way to assign an extra cost for errors is to change the objective function to be minimized from $||\mathbf{w}||^2/2$ to $||\mathbf{w}||^2/2 + C(\Sigma_i e_i)$, where C is a parameter to be chosen by the user, a larger C corresponding to assigning a higher penalty to errors.

How can the above methods be generalized to the case where the decision function (f($\mathbf{x}$) whose sign represents the class assigned to data point $\mathbf{x}$) is not a linear function of the data? First notice that the only way in which the data appears in the training problem, is in the form of dot products, $\mathbf{x}_i \cdot \mathbf{x}_j$. Now suppose we first mapped the data (Figure 2) to some other (possibly even infinite dimensional) Euclidean space H, using a mapping which we will call Φ:

$$\Phi : \mathbf{R}^d \mapsto H, \quad (11)$$

Then of course the training algorithm would only depend on the data through dot products in H, i.e. on functions of the form $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$. Now if there were a "kernel function" K such that $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$, we would only need to use K in the training algorithm, and would never need to explicitly even know what Φ is. The kernel function has to satisfy Mercer's condition [1].One example for this function is Gaussian:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right), \quad (12)$$

In this particular example, H is infinite dimensional, so it would not be very easy to work with Φ explicitly. However, if one replaces $\mathbf{x}_i \cdot \mathbf{x}_j$ by $K(\mathbf{x}_i, \mathbf{x}_j)$ everywhere in the training algorithm, the algorithm will happily produce a support vector machine which lives in an infinite dimensional space, and furthermore do so in roughly the same amount of time it would take to train on the un-mapped data. All the considerations of the previous sections hold, since we are still doing a linear separation, but in a different space. But how can we use this machine? After all, we need $\mathbf{w}$, and that will live in H. But in test phase an SVM is used by computing dot products of a given test point $\mathbf{x}$ with $\mathbf{w}$, or more specifically by computing the sign of

$$f(\mathbf{x}) \equiv \sum_{i=1}^{N_S} \alpha_i y_i \Phi(\mathbf{s}_i) \cdot \Phi(\mathbf{x}) + b = \sum_{i=1}^{N_S} \alpha_i y_i K(\mathbf{s}_i, \mathbf{x}) + b \quad (13)$$

where the $\mathbf{s}_i$ are the support vectors. So again we can avoid computing $\Phi(\mathbf{x})$ explicitly and use the $K(\mathbf{s}_i, \mathbf{x}) = \Phi(\mathbf{s}_i) \cdot \Phi(\mathbf{x})$ instead.

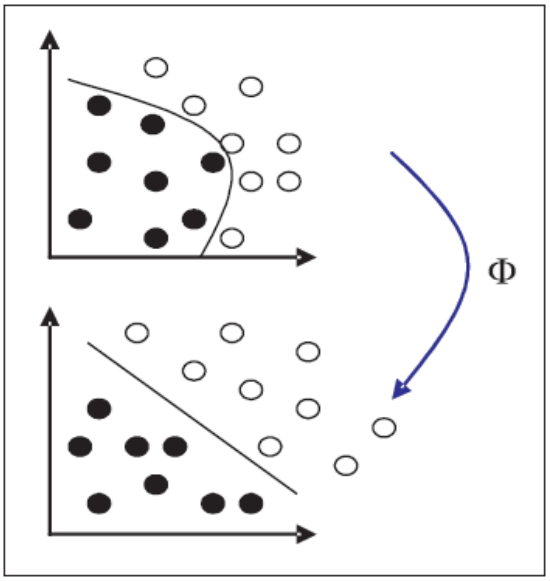

Figure 2 – General principle of SVM: projection of data in an optimal dimensional space.

3 An overview of widely used multi-class SVM classification methods

Although SVMs were originally designed as binary classifiers, approaches that address a multi-class problem as a single "all-together" optimization problem exist [5], but are computationally much more expensive than solving several binary problems.

A variety of techniques for decomposition of the multi-class problem into several binary problems using Support Vector Machines as binary classifiers have been proposed, and several widely used are given in this section.

3.1 One-against-all (OvA)

For the N-class problems (N>2), N two-class SVM classifiers are constructed [6]. The i^{th} SVM is trained while labeling the samples in the i^{th} class as positive examples and all the rest as negative examples. In the recognition phase, a test example is presented to all N SVMs and is labelled according to the maximum output among the N classifiers. The disadvantage of this method is its training complexity, as the number of training samples is large. Each of the N classifiers is trained using all available samples.

3.2 One-against-one (OvO)

This algorithm constructs $N(N\text{-}1)/2$ two-class classifiers, using all the binary pair-wise combinations of the N classes. Each classifier is trained using the samples of the

first class as positive examples and the samples of the second class as negative examples. To combine these classifiers, the Max Wins algorithm is adopted. It finds the resultant class by choosing the class voted by the majority of the classifiers [7]. The number of samples used for training of each one of the OvO classifiers is smaller, since only samples from two of all N classes are taken in consideration. The lower number of samples causes smaller nonlinearity, resulting in shorter training times. The disadvantage of this method is that every test sample has to be presented to large number of classifiers $N(N\text{-}1)/2$. This results in slower testing, especially when the number of the classes in the problem is big [8].

3.3 Directed acyclic graph SVM (DAGSVM)

Introduced by Platt [1] the DAGSVM algorithm for training an $N(N\text{-}1)/2$ classifiers is the same as in one-against-one. In the recognition phase, the algorithm depends on a rooted binary directed acyclic graph to make a decision [9]. DAGSVM creates a model for each pair of classes. When one such model, which is able to separate class c_1 from class c_2, classifies a certain test example into class c_1, it does not really vote "for" class c_1, rather it votes "against" class c_2, because the example must lie on the other side of the separating hyperplane than most of the class c_2 samples. Therefore, from that point onwards the algorithm ignores all the models involving the class c_2. This means that after each classification with one of the binary models, one more class can be thrown out as a possible candidate, and after only N-1 steps just one candidate class remains, which therefore becomes the prediction for the current test example. This results in significantly faster testing, while achieving similar recognition rate as One-against-one.

3.4 Binary tree of SVM (BTS)

This method uses multiple SVMs arranged in a binary tree structure [10]. A SVM in each node of the tree is trained using two of the classes. The algorithm then employs probabilistic outputs to measure the similarity between the remaining samples and the two classes used for training. All samples in the node are assigned to the two subnodes derived from the previously selected classes by similarity. This step repeats at every node until each node contains only samples from one class. The main problem that should be considered seriously here is training time, because aside training, one has to test all samples in every node to find out which classes should be assigned to which subnode while building the tree. This may decrease the training performance considerably for huge training datasets.

4 Support vector machines utilizing a binary decision tree

In this paper we propose a binary decision tree architecture that uses SVMs for making the binary decisions in the nodes. The proposed classifier architecture SVM-BDT (Support Vector Machines utilizing Binary Decision Tree), takes advantage of both the efficient computation of the tree architecture and the high classification accuracy of SVMs. Utilizing this architecture, N-1 SVMs needed to be trained for an N class problem, but only at most $\lceil \log_2 N \rceil$ SVMs are required to be consulted to classify a sample. This can lead to a dramatic improvement in recognition speed when addressing problems with big number of classes.

An example of SVM-BDT that solves a 7 - class pattern recognition problem utilizing a binary tree, in which each node makes binary decision using a SVM is shown on Figure 3. The hierarchy of binary decision subtasks should be carefully designed before the training of each SVM classifier.

The recognition of each sample starts at the root of the tree. At each node of the binary tree a decision is being made about the assignment of the input pattern into one of the two possible groups represented by transferring the pattern to the left or to the right sub-tree. Each of these groups may contain multiple classes. This is repeated recursivly downward the tree until the sample reaches a leaf node that represents the class it has been assigned to.

There exist many ways to divide N classes into two groups, and it is critical to have proper grouping for the good performance of SVM-BDT.

For consistency between the clustering model and the way SVM calculates the decision hyperplane, the clustering model utilizes distance measures at the kernel space, rather than at the input space. Because of this, all training samples are mapped into the kernel space with the same kernel function that is to be used in the training phase.

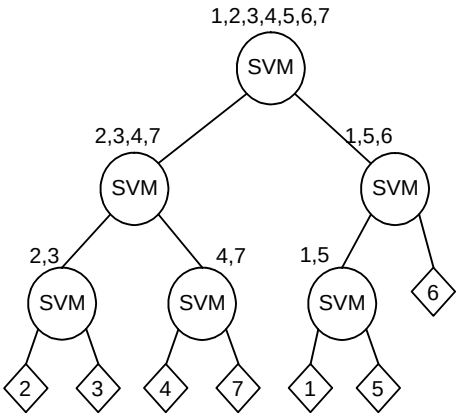

Figure 3: Illustration of SVM-BDT.

The SVM-BDT method that we propose is based on recursively dividing the classes in two disjoint groups in every node of the decision tree and training a SVM that will decide in which of the groups the incoming unknown sample should be assigned. The groups are determined by a clustering algorithm according to their class membership.

Let's take a set of samples x_1, x_2, ..., x_M each one labeled by $y_i \in \{c_1, c_2, ..., c_N\}$ where N is the number of classes. SVM-BDT method starts with dividing the classes in two disjoint groups g_1 and g_2. This is performed by calculating N gravity centres for the N different classes. Then, the two classes that have the biggest Euclidean distance from each other are assigned to each of the two clustering groups. After this, the class with the smallest Euclidean distance from one of the clustering groups is found and assigned to the corresponding group. The gravity center of this group is then recalculated to represent the addition of the samples of the new class to the group. The process continues by finding the next unassigned class that is closest to either of the clustering groups, assigning it to the corresponding group and updating the group's gravity center, until all classes are assigned to one of the two possible groups.

This defines a grouping of all the classes in two disjoint groups of classes. This grouping is then used to train a SVM classifier in the root node of the decision tree, using the samples of the first group as positive examples and the samples of the second group as negative examples. The classes from the first clustering group are being assigned to the first (left) subtree, while the classes of the second clustering group are being assigned to the (right) second subtree. The process continues recursively (dividing each of the groups into two subgroups applying the procedure explained above), until there is only one class per group which defines a leaf in the decision tree.

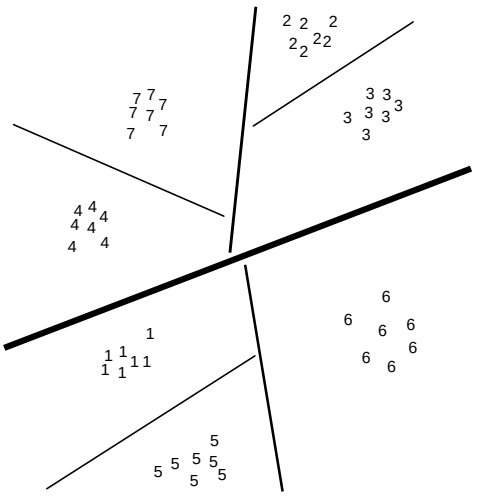

Figure 4: SVM-BDT divisions of the seven classes.

For example, Figure 4 illustrates grouping of 7 classes, while Figure 3 shows the corresponding decision tree of SVMs. After calculating the gravity centers for all classes, the classes c_2 and c_5 are found to be the furthest apart from each other, considering their Euclidean distance and are assigned to group g_1 and g_2 accordingly. The closest to group g_1 is class c_3, so it is assigned to the group g_1, followed by recalculation of the g_1's gravity center. In the next step, class c_1 is the closest to group g_2, so it is assigned to that group and the group's gravity center is recalculated. In the following iteration, class c_7 is assigned to g_1 and class c_6 is assigned to g_2, folowed by recalculating of group's gravity centers. Finally class c_4 is assigned to g_1. This completes the first round of grouping that defines the classes that will be transferred to the left and the right subtree of the root node. The SVM classifier in the root is trained by considering samples from the classes $\{c_2, c_3, c_4, c_7\}$ as positive examples and samples from the classes $\{c_1, c_5, c_6\}$ as negative examples.

The grouping procedure is repeated independently for the classes of the left and the right subtree of the root, which results in grouping c_7 and c_4 in $g_{1,1}$ and c_2 and c_3 in $g_{1,2}$ in the left node of the tree and c_1 and c_5 in $g_{2,1}$ and c_6 in $g_{2,2}$ in the right node of the tree. The concept is repeated for each SVM associated to a node in the taxonomy. This will result in training only N-1 SVMs for solving an N-class problem.

5 Related work and discussion

Various multi-class classification algorithms can be compared by their predictive accuracy and their training and testing times. The training time T for a binary SVM is estimated empirically by a power law [13] stating that $T \approx \alpha M^d$, where M is the number of training samples and α is a proportionality constant. The parameter d is a constant, which depends of the datasets and it is typically in the range [1, 2]. According to this law, the estimated training time for OvA is

$$T_{OvA} \approx N \alpha M^d , \quad (11)$$

where N is the number of classes in the problem.

Without loss of generality, let's assume that each of the N classes has the same number of training samples. Thus, each binary SVM of OvO approach only requires $2M/N$ samples. Therefore, the training time for OvO is:

$$T_{OvO} \approx \alpha \frac{N(N-1)}{2}\left(\frac{2M}{N}\right)^d \approx N^{2-d}\alpha M^d , \quad (12)$$

The training time for DAGSVM is same as OvO.

As for BTS and SVM-BDT, the training time is summed over all the nodes in the $\lceil \log_2 N \rceil$ levels. In the i^{th} level, there are 2^{i-1} nodes and each node uses $2M/N$ for BTS and $M/2^{i-1}$ for SVM-BDT training samples. Hence, the total training time for BTS is:

$$T_{BTS} \approx \sum_{i=1}^{\lceil \log_2(N) \rceil} \alpha 2^{i-1}\left(2\frac{M}{N}\right)^d \approx \alpha\left(2\frac{M}{N}\right)^d \sum_{i=1}^{\lceil \log_2(N) \rceil} 2^{i-1} \approx N^{1-d}\alpha M^d , \quad (13)$$

and for SVM-BDT is:

$$T_{SVM-BDT} \approx \sum_{i=1}^{\lceil \log_2(N) \rceil} \alpha 2^{i-1} \left(\frac{M}{2^{i-1}} \right)^d \approx \alpha M^d , \quad (14)$$

It must be noted that $T_{SVM\text{-}BDT}$ in our algorithm does not include the time to build the hierarchy structure of the N classes, since it consumes insignificant time compared to the quadratic optimization time that dominates the total SVM training time. On the other hand, in the process of building the tree, BTS requires testing of each trained SVM with all the training samples in order to determine the next step, therefore significantly increasing the total training time.

According to the empirical estimation above, it is evident that the training speed of SVM-BDT is comparable with OvA, OvO, DAGSVM and BTS.

In the testing phase, DAGSVM performs faster than OvO and OvA, since it requires only N-1 binary SVM evaluations. SVM-BDT is even faster than DAGSVM because the depth of the SVM-BDT decision tree is $\lceil \log_2 N \rceil$ in the worst case, which is superior to N-1, especially when N>>2.

While testing, the inner product of the sample's feature vector and all the support vectors of the model are calculated for each sample. The total number of support vectors in the trained model directly contributes to the major part of the evaluation time, which was also confirmed by the experiments.

A multistage SVM (MSVM) for multi-class problem has been proposed by Liu et al. [11]. They use Support Vector Clustering (SVC) [12] to divide the training data into two parts that are used to train a binary SVM. For each partition, the same procedure is recursively repeated until the binary SVM gives an exact label of class. An unsolved problem in MSVM is how to control the SVC to divide the training dataset into exact two parts. However, this procedure is painful and unfeasible, especially for large datasets. The training set from one class could belong to both clusters, resulting in decreased predictive accuracy.

There are different approaches for solving multi-class problems which are not based on SVM. Some of them are presented in the following discussion. However, the experimental results clearly show that their classification accuracy is significantly smaller than the SVM based methods.

Ensemble techniques have received considerable attention within the recent machine learning research [16][17][18][19]. The basic goal is to train a diverse set of classifiers for a single learning problem and to vote or average their predictions. The approach is simple as well as powerful, and the obtained accuracy gains often have solid theoretical foundations [20][20][21]. Averaging the predictions of these classifiers helps to reduce the variance and often increases the reliability of the predictions. There are several techniques for obtaining a diverse set of classifiers. The most common technique is to use subsampling to diversify the training sets as in Bagging [21] and Boosting [20]. Other techniques include the use of different feature subsets for every classifier in the ensemble [23], to exploit the randomness of the base algorithms [24], possibly by artificially randomizing their behavior [25], or to use multiple representations of the domain objects. Finally, classifier diversity can be ensured by modifying the output labels, i.e., by transforming the learning tasks into a collection of related learning tasks that use the same input examples, but different assignments of the class labels. Error-correcting output codes are the most prominent example for this type of ensemble methods [22].

Error-correcting output codes are a popular and powerful class binarization technique. The basic idea is to transform an N-class problem into n binary problems ($n > N$), where each binary problem uses a subset of the classes as the positive class and the remaining classes as a negative class. As a consequence, each original class is encoded as an n-dimensional binary vector, one dimension for each prediction of a binary problem (+1 for positive and −1 for negative). The resulting matrix of the form $\{-1, +1\}$ $N \times n$ is called the coding matrix. New examples are classified by determining the row in the matrix that is closest to the binary vector obtained by submitting the example to the n classifiers. If the binary problems are chosen in a way that maximizes the distance between the class vectors, the reliability of the classification can be significantly increased. Error-correcting output codes can also be easily parallelized, but each subtask requires the total training set.

Similar to binarization, some approaches suggest mapping the original multiple classes into three clsses. A related technique where multi-class problems are mapped to 3-class problems is proposed by Angulo and Catal'a [26]. Like with pairwise classification, they propose generating one training set for each pair of classes. They label the two class values with target values +1 and −1, and additionally, samples of all other classes are labeled to a third class, with a target value of 0. This idea leads to increased size of the training set compared to the binary classification. The mapping into three classes was also used by Kalousis and Theoharis [27] for predicting the most suitable learning algorithm(s) for a given dataset. They trained a nearest-neighbor learner to predict the better algorithm of each pair of learning algorithms. Each of these pairwise problems had three classes: one for each algorithm and a third class named "tie", where both algorithms had similar performances.

Johannes Fürnkranz has investigated the use of round robin binarization (or pair-wise classification) [28] as a technique for handling multi-class problems with separate-and-conquer rule learning algorithms (aka covering algorithms). In particular, round robin binarization helps Ripper [29] outperform C5.0 on multi-class problems, whereas C5.0 outperforms the original version of Ripper on the same problems.

6 Experimental results

In this section, we present the results of our experiments with several multi-class problems. The performance was measured on the problem of recognition of handwritten digits and letters.

Here, we compare the results of the proposed SVM-BDT method with the following methods:

1) one-against-all (OvA);
2) one-against-one (OvO);
3) DAGSVM;
4) BTS;
5) Bagging
6) Random Forests
7) Multilayer Perceptron (MLP, neural network)

The training and testing of the SVMs based methods (OvO, OvA, DAGSVM, BTS and SVM-BDT) was performed using a custom developed application that uses the Torch library [14]. For solving the partial binary classification problems, we used SVMs with Gaussian kernel. In these methods, we had to optimize the values of the kernel parameter σ and penalty C. For parameter optimization we used experimental results. The achieved parameter values for the given datasets are given in Table 1.

Table 1. The optimized values for σ and C for the used datasets.

	MNIST	Pendigit	Optdigit	Statlog
σ	2	60	25	1.1
C	100	100	100	100

We also developed an application that uses the same (Torch) library for the neural network classification. One hidden layer with 25 units was used by the neural network. The number of hidden units was determined experimentally.

The classifications based on ensembles of decision trees [30] (Bagging and Random Forest) was performed by Clus, a popular decision tree learner based on the principles stated by Blockeel et al. [31]. There were 100 models in the ensembles. The pruning method that we used was C4.5. The number of selected features in the Random Forest method was $\lfloor \log_2 M \rfloor$, where M is the number of features in the dataset.

The most important criterion in evaluating the performance of a classifier is usually its recognition rate, but very often the training and testing time of the classifier are equally important.

In our experiments, four different multi-class classification problems were addressed by each of the eight previously mentioned methods. The training and testing time and the recognition performance were recorded for every method.

The first problem was recognition of isolated handwritten digits (10 classes) from the MNIST database. The MNIST database [15] contains grayscale images of isolated handwritten digits. From each digit image, after performing a slant correction, 40 features were extracted. The features are consisted of 10 horizontal, 8 vertical and 22 diagonal projections [25]. The MNIST database contains 60.000 training samples, and 10.000 testing samples.

The second and the third problem are 10 class problems from the UCI Repository [33] of machine learning databases: Optdigit and Pendigit. Pendigit has 16 features, 7494 training samples, and 3498 testing samples. Optdigit has 64 features, 3823 training samples, and 1797 testing samples.

The fourth problem was recognition of isolated handwritten letters – a 26-class problem from the Statlog collection [34]. Statlog-letter contains 15.000 training samples, and 5.000 testing samples, where each sample is represented by 16 features.

The classifiers were trained using all available training samples of the set and were evaluated by recognizing all the test samples from the corresponding set. All tests were performed on a personal computer with an Intel Core2Duo processor at 1.86GHz with the Windows XP operating system.

Tables 2 through 4 show the results of the experiments using 8 different approaches (5 approaches based on SVM, two based on ensembles of decision trees and one neural network) on each of the 4 data sets. The first column of each table describes the classification method. Table 2 gives the prediction error rate of each method applied on each of the datasets. Table 3 and table 4 shows the testing and training time of each algorithm, for the datasets, measured in seconds, respectively.

The results in the tables show that SVM based methods outperform the other approaches, in terms of classification accuracy. In terms of speed, SVM based methods are faster, with different ratios for different datasets. In overall, the SVM based algorithms were significantly better compared to the non SVM based methods.

The results in table 2 show that for all datasets, the one-against-all (OvA) method achieved the lowest error rate. For the MNIST, Pendigit and Optdigit datasets, the other SVM based methods (OvO, DAGSVM, BTS and our method - SVM-BDT) achieved higher, but similar error rates. For the recognition of handwritten letters from the Statlog database, the OvO and DAGSVM methods achieved very similar error rates that were about 1.5% higher than the OvA method. The BTS method showed the lowest error rate of all methods using one-against-one SVMs. Our SVM-BDT method achieved better recognition rate than all the methods using one-against-one SVMs, including BTS. Of the non SVM based methods, the Random Forest method achieved the best recognition accuracy for all datasets. The prediction performance of the MLP method was comparable to the Random Forest method for the 10-class problems, but noticeably worse for the 26-class problem.

The MLP method is the fastest one in terms of training and testing time, which is evident in Table 3 and Table 4. The classification methods based on ensembles of trees were the slowest in the training and the testing phase, especially the Bagging method. Overall, the Random Forest method was more accurate than the other non SVM based methods, while the MLP method was the fastest.

The results in Table 3 show that the DAGSVM method achieved the fastest testing time of all the SVM based methods for the MNIST dataset. For the other datasets, the testing time of DAGSVM is comparable

with BTS and SVM-BDT methods and their testing time is noticeably better than the one-against-all (OvA) and one-against-one (OvO) methods. The SVM-BDT method was faster in the recognition phase for the Pendigit dataset and slightly slower than DAGSVM method for the Statlog dataset.

Table 2. The prediction error rate (%) of each method for every dataset

Classifier	MNIST	Pendigit	Optdigit	Statlog
OvA	1.93	1.70	1.17	3.20
OvO	2.43	1.94	1.55	4.72
DAGSVM	2.50	1.97	1.67	4.74
BTS	2.24	1.94	1.51	4.70
SVM-BDT	2.45	1.94	1.61	4.54
R. Forest	3.92	3.72	3.18	4.98
Bagging	4.96	5.38	7.17	8.04
MLP	4.25	3.83	3.84	14.14

Table 3. Testing time of each method for every dataset measured in seconds

Classifier	MNIST	Pendigit	Optdigit	Statlog
OvA	23.56	1.75	1.63	119.50
OvO	26.89	3.63	1.96	160.50
DAGSVM	9.46	0.55	0.68	12.50
BTS	26.89	0.57	0.73	17.20
SVM-BDT	25.33	0.54	0.70	13.10
R. Forest	39.51	3.61	2.76	11.07
Bagging	34.52	2.13	1.70	9.76
MLP	2.12	0.49	0.41	1.10

Table 4. Training time of each method for every dataset measured in seconds

Classifier	MNIST	Pendigit	Optdigit	Statlog
OvA	468.94	4.99	3.94	554.20
OvO	116.96	3.11	2.02	80.90
DAGSVM	116.96	3.11	2.02	80.90
BTS	240.73	5.21	5.65	387.10
SVM-BDT	304.25	1.60	1.59	63.30
R. Forest	542.78	17.08	22.21	50.70
Bagging	3525.31	30.87	49.4	112.75
MLP	45.34	2.20	1.60	10.80

In terms of training speeds, it is evident in Table 4 that among the SVM based methods, SVM-BDT is the fastest one in the training phase. For the three 10-class problems the time needed to train the 10 classifiers for the OvA approach took about 4 times longer than training the 45 classifiers for the OvO and DAGSVM methods. Due to the huge number of training samples in the MNIST dataset (60000), SVM-BDT's training time was longer compared to other one-against-one SVM methods. The huge number of training samples increases the nonlinearity of the hyperplane in the SVM, resulting in an incresed number of support vectors and increased training time. Also, the delay exists only in the first level of the tree, where the entire training dataset is used for training. In the lower levels, the training time of divided subsets is not as significant as the first level's delay.

In the other 10 class problems, our method achieved the shortest training time. For the Statlog dataset, the time needed for training of the 26 one-against-all SVMs was almost 7 times longer than the time for training the 325 one-against-one SVMs. The BTS method is the slowest one in the training phase of the methods using one-against-one SVMs. It must be noted that as the number of classes in the dataset increases, the advantage of SVM-BDT becomes more evident. The SVM-BDT method was the fastest while training, achieving better recognition rate than the methods using one-against-one SVMs. It was only slightly slower in recognition than DAGSVM.

7 Conclusion

A novel architecture of Support Vector Machine classifiers utilizing binary decision tree (SVM-BDT) for solving multiclass problems was presented. The SVM-BDT architecture was designed to provide superior multi-class classification performance, utilizing a decision tree architecture that requires much less computation for deciding a class for an unknown sample. A clustering algorithm that utilizes distance measures at the kernel space is used to convert the multi-class problem into binary decision tree, in which the binary decisions are made by the SVMs. The results of the experiments show that the speed of training and testing are improved, while keeping comparable or offering better recognition rates than the other SVM multi-class methods. The experiments showed that this method becomes more favourable as the number of classes in the recognition problem increases.

References

[1] V. Vapnik. *The Nature of Statistical Learning Theory*, 2nd Ed. Springer, New York, 1999.

[2] C. J. C. Burges. *A tutorial on support vector machine for pattern recognition*. Data Min. Knowl. Disc. 2 (1998) 121.

[3] T. Joachims. Making large scale SVM learning practical. in B. Scholkopf, C. Bruges and A. Smola (eds). *Advances in kernel methods-support vector learning*, MIT Press, Cambridge, MA, 1998.

[4] R. Fletcher. *Practical Methods of Optimization*. 2nd Ed. John Wiley & Sons. Chichester (1987).

[5] J. Weston, C. Watkins. *Multi-class support vector machines*. Proceedings of ESANN99, M. Verleysen, Ed., Brussels, Belgium, 1999.

[6] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[7] J. H. Friedman. Another approach to polychotomous classification. Technical report. Department of Statistics, Stanford University, 1997.

[8] P. Xu, A. K. Chan. Support vector machine for multi-class signal classification with unbalanced samples. *Proceedings of the International Joint Conference on Neural Networks 2003*. Portland, pp.1116-1119, 2003.

[9] Platt, N. Cristianini, J. Shawe-Taylor. Large margin DAGSVM's for multiclass classification. *Advances in Neural Information Processing System*. Vol. 12, pp. 547–553, 2000.

[10] B. Fei, J. Liu. Binary Tree of SVM: A New Fast Multiclass Training and Classification Algorithm. *IEEE Transaction on neural networks*, Vol. 17, No. 3, May 2006.

[11] X. Liu, H. Xing, X. Wang. A multistage support vector machine. *2nd International Conference on Machine Learning and Cybernetics*, pages 1305-1308, 2003.

[12] A. Ben-Hur, D. Horn, H. Siegelmann, V. Vapnik. Support vector clustering. *Journal of Machine Learning Research*, vol. 2:125-137, 2001.

[13] J. Platt. Fast training of support vector machines using sequential minimal optimization. *In Advances in Kernel Methods - Support Vector Learning*. Pages 185-208, Cambridge, MA, 1999. MIT Press.

[14] R. Collobert, S. Bengio, J. Mariéthoz. Torch: a modular machine learning software library. Technical Report IDIAP-RR 02-46, IDIAP, 2002.

[15] __, MNIST, MiniNIST, USA http://yann.lecun.com/exdb/mnist

[16] T. G. Dietterich. Machine learning research: Four current directions. *AI Magazine*, 18(4): 97–136, Winter 1997.

[17] G. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli (eds.) *First International Workshop on Multiple Classifier Systems*, pp. 1–15. Springer-Verlag, 2000a.

[18] D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[19] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–169, 1999.

[20] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[21] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[22] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[23] S. D. Bay. Nearest neighbor classification from multiple feature subsets. *Intelligent Data Analysis*, 3(3):191–209, 1999.

[24] J. F. Kolen and J. B. Pollack. Back propagation is sensitive to initial conditions. *In Advances in Neural Information Processing Systems 3 (NIPS-90)*, pp. 860–867. Morgan Kaufmann, 1991.

[25] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–158, 2000b.

[26] C. Angulo and A. Catal`a. K-SVCR. A multi-class support vector machine. In R. L´opez de M´antaras and E. Plaza (eds.) *Proceedings of the 11th European Conference on Machine Learning (ECML-2000)*, pp. 31–38. Springer-Verlag, 2000.

[27] A. Kalousis and T. Theoharis. Noemon: Design, implementation and performance results of an intelligent assistant for classifier selection. *Intelligent Data Analysis*, 3(5):319–337, 1999.

[28] Johannes Fürnkranz, Round robin classification, *The Journal of Machine Learning Research*, 2, p.721-747, 3/1/2002

[29] W. W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell (eds.) *Proceedings of the 12th International Conference on Machine Learning* (ML-95), pp. 115–123, Lake Tahoe, CA, 1995. Morgan Kaufmann.

[30] D. Kocev, C. Vens, J. Struyf and S. Dˇzeroski. Ensembles of multi-objective decision trees. *Proceedings of the 18th European Conference on Machine Learning* (pp. 624–631) (2007). Springer.

[31] H. Blockeel, J. Struyf. Efficient Algorithms for Decision Tree Cross-validation. *Journal of Machine Learning Research* 3:621-650, 2002.

[32] D. Gorgevik, D. Cakmakov. An Efficient Three-Stage Classifier for Handwritten Digit Recognition. *Proceedings of 17th Int. Conference on Pattern Recognition*, *ICPR2004*. Vol. 4, pp. 507-510, IEEE Computer Society, Cambridge, UK, 23-26 August 2004.

[33] C. Blake, E. Keogh and C. Merz. UCI Repository of Machine Learning Databases, (1998). Statlog Data Set, http://archive.ics.uci.edu/ml/datasets.html [Online]

[34] Statlog Data Set, http://archive.ics.uci.edu/ml/-datasets/Letter+Recognition [Online]