Support Vector Machine Classification for Large Datasets Using Decision Tree and Fisher Linear Discriminant

Asdrúbal López Chau\textsuperscript{a,b}, Xiaoou Li\textsuperscript{b}, Wen Yu\textsuperscript{c}

\textsuperscript{a}Universidad Autónoma del Estado de México, CU UAEM Zumpango, Zumpango Edo Mex, 55600, México
\textsuperscript{b}Departamento de Computación, CINVESTAV-IPN, Mexico City, 07360, México
\textsuperscript{c}Departamento de Control Automático, CINVESTAV-IPN, Mexico City, 07360, México

Abstract

The training of a support vector machine (SVM) has a time complexity between $O(n^2)$ and $O(n^3)$. Most training algorithms for SVM are not suitable for large data sets. Decision trees can simplify SVM training, however the classification accuracy becomes lower when there are inseparable points. This paper introduces a novel method for SVM classification. A decision tree is used to detect low entropy regions in input space. We use Fisher’s linear discriminant to detect the data near to support vectors.

Experimental results demonstrate that our approach has good classification accuracy and low standard deviation, the training is significantly faster than other training methods.

Keywords: Support vector machine, decision tree, Fisher linear discriminant

1. Introduction

Support vector machine (SVM) is a well known classifier due to its excellent classification accuracy, generalization and compact model. It offers a hyperplane that represents the largest separation (or margin) between two classes [4]. In the linearly separable case the hyperplane is easy to compute, however in the general case it is necessary to use a soft margin SVM by introducing slack variables. In this way it is possible to find a hyperplane that splits the examples as cleanly as possible. In order to find a separation hyperplane, it is necessary to solve a Quadratic Programming Problem (QPP). This task is computational expensive, it has $O(n^3)$ time and $O(n^2)$ space complexities with $n$ data [3]. The standard SVM is unfeasible for large data sets [20].

Many researchers have tried to find possible methods to apply SVM classification for large data sets. These methods can be divided into three types: a) reducing training data sets (data selection or data reduction) [14], b) using geometric properties of SVM [1], c) modifying SVM classifiers [24].
d) decomposition [20]. Clustering is an effective tool to reduce data set size, for example, hierarchical clustering [15] and parallel clustering [19]. The sequential minimal optimization (SMO) [20] breaks the large QP problem into a series of smallest possible QP problems. The projected conjugate gradient (PCG) chunking scales somewhere between linear and cubic in the training set [8].

The random sampling method [14] is simple and commonly used for large data sets. However it needs to be applied several times, and the obtained results are not repeatable. Data selection methods choose objects which are support vectors (SV). These data are used for training the SVM classifier. Generally, the number of the support vectors is a small subset of the whole data [25]. The goals of this type of method are: (a) fast detection of support vectors (SV) which define the optimal separating hyperplane, (b) remove the data which are impossible to be SVs, (c) obtain similar accuracy using the reduced data set. In this paper, we use the data selection technique to train SVM with large data sets.

Decision tree (DT) is classification method commonly used in many applications. A tree can be “learned” or induced by splitting the input space into subsets, based on an attribute value test. This process is repeated on each derived subset in a recursive manner [2]. Generally, the classification accuracy of SVM is better than DT. The training time of SVM is longer than DT for large data sets. The combination of SVM and DT can overcome two shortcomings of SVM: Computational burden and multiclass classification. In [7] each partition separated by a decision tree is used to train SVM. It is similar with SMO, the large QP problem is transformed into several small SVMs by DT. In [13], the decision boundary of SVM is approximated by the boundaries of the decision tree. The multiclass SVM classification can be realized by the standard two-class classification by a binary tree [10]. There are two main problems: SVM has to be use to compute the margins at each leaf of DT [16]; the classification accuracy is poor when DT is not big enough [21], or the data are imbalance [27]. To improve the classification accuracy, the separability (margin measures) information are inserted in [5].

Fisher linear discriminant (FLD) is used to find a linear combination of features which separates two or more classes of objects. It is different from principal component analysis (PCA), in the last it is not an interdependence technique: a distinction between independent variables and dependent variables (also called criterion variables) must be made. In fact, SVM can be seen as a way to “sparsity” FLD [23]. FLD has been successfully applied in feature selection [26] and face recognition. The geometric properties of SVM can also be used to reduce the training data. In the linearly separable case, the maximum-margin hyperplane is equivalent to finding the closest pair of points in the convex hulls[1]. Neighborhood properties of the objects can be applied to detect SV and to improve classification accuracy of SVM. SVM with FLD uses the geometric properties of SVM. It is difficult to apply geometric properties in high-dimension and multiclass classification. While DT can overcome this disad-
vantage of SVM+FLD. In this paper, we use FLD to select part of data in each partition generated by DT.

In this paper, we propose a novel data reduction method for SVM, it uses DT and FLD. The combination of decision tree and Fisher linear discriminant enable SVM for large data sets.

With our method, low entropy regions in input space are detected. Only the data near to the support vectors are used to train SVM. We use 17 benchmark data sets to test our methods. The results show that our new classification method is very competitive compared with the other popular methods, such as LibSVM and SMO, especially for large data sets.

2. Three Classification Methods: SVM, DT, and FLD

2.1. Support Vector Machine

The training set $X$ is given as

$$X = \{x_i, y_i\}_{i=1}^{n}$$

where $x_i \in \mathbb{R}^d$, $y_i \in (1, \ldots, C_L)$. The $C_L$ is the number of classes. Only continuous type attributes are used in this paper. SVM classifies data sets with an optimal separating hyperplane, which is given by

$$w^T \varphi(x_i) + b$$

This hyperplane is obtained by solving the following quadratic programming problem

$$\min_{w, b} J(w) = \frac{1}{2}ww^T + c \sum_{i=1}^{n} \xi_i$$

such that: $y_i [w^T \varphi(x_i) + b] \geq 1 - \xi_i$ (3)

where $\xi_i > 0$, $i = 1 \ldots n$, are the slack variables, to tolerate mis-classifications. $c > 0$ is a regularization parameter. (3) is equivalent to the following dual problem with the Lagrange multipliers $\alpha_i > 0$

$$\max_{\alpha_i} J(w) = -\frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i \cdot x_j) + \sum_{i=1}^{n} \alpha_i$$

such that : $\sum_{i=1}^{n} \alpha_i y_i = 0, C \geq \alpha_i \geq 0, i = 1, 2, \ldots, n$ (4)

with $C > 0$, $\alpha_i \geq 0$, $i = 1, 2, \ldots, n$, the coefficients corresponding to $x_i$. All $x_i$ with nonzero $\alpha_i$ are called support vectors. The function $K$ is the kernel which must satisfy the Mercer condition [4]. The resulting optimal decision function is

$$y_i = \text{sign} \left( \sum_{i=1}^{n} \alpha_i y_i K(x_i \cdot x_j) + b \right)$$

where $x = [x_1, x_2, \ldots, x_n]$ is the input data, $\alpha_i$ and $y_i$ are Lagrange multipliers.

A previously unseen sample $x$ can be classified by (5). There is a Lagrangian multiplier $\alpha$ for each training point. When the maximum margin of the hyperplane is found, only the closed points to the hyperplane satisfy $\alpha > 0$. These points are called support vectors (SV), the other points satisfy $\alpha = 0$. So the solution is sparse. Here $b$ is determined by Kuhn-Tucker conditions:

$$\frac{\partial L}{\partial w} = 0, \quad w = \sum_{i=1}^{n} \alpha_i y_i \varphi(x_i)$$

$$\frac{\partial L}{\partial b} = 0, \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

(6)

$$\frac{\partial L}{\partial \xi_i} = 0, \quad \alpha_i - c \geq 0$$

$$\alpha_i \{ y_i [w^T \varphi(x_i) + b] \geq 1 - \xi_i \} = 0$$
2.2. Decision Trees

When a data set $X$ is large, the computational burden of (4) is heavy. We first use decision tree to separate $X$ into several subsets.

A decision tree is classifier whose model resembles a tree structure $T$, this structure is built from a labeled data set $X$. A decision tree is composed of nodes, and edges that connect the nodes. There are two type nodes: internal and terminal. The internal nodes have branches to connect to other ones, called its sons. The terminal nodes do not have any sons. A terminal node is called a leaf $L$ of $T$.

In general, a decision tree is generated by partitioning input data recursively into "pure" regions with respect to certain measurement of the impurity. The measurements are defined in terms of instance distribution of the input splitting region. We use the following three impurity measurements: entropy, misclassification and the Gini impurities

$$\text{Entropy}(t) = -\sum_{i=0}^{C_L-1} p(i|t) \log_2 p(i|t)$$

$$\text{Classification error}(t) = 1 - \max[p(i|t)]$$

$$\text{Gini} = 1 - \sum_{i=0}^{C_L-1} [p(i|t)]^2$$

(7)

where $C_L$ is the number of classes, $p(i|t)$ is the probability of of example $i$ to be of the class $t$, which is the number of examples in the class $t$ divided by the size of the set $X$, i.e., $p(i|t) = \frac{|y_i = t|}{|X|}$

In order to create partitions in the input space, we use the following induction method

$$\text{att}_i < \text{split point}_i$$

(8)

where $\text{att}_i$ is an attribute of training set $X$, the “split point” is a value of the same type with the attribute $i$.

The decision boundaries that are discovered by a decision tree are perpendicular to the axes of features. These boundaries can be seen as hyper rectangles. They are represented by the leaves $L$ of $T$. A decision tree is shown in Figure 1.

![Figure 1: Example of a decision tree](image)

Any complex decision boundary can be approximated by using a large enough decision tree [9]. However, large decision trees can yield an over-fitting problem. The pruning method can be used to solve over-fitting in decision trees. There are two basic strategies to prune a decision tree: pre-pruning and post-prunning. The former strategy uses a criterion to stop the branch growing. The second strategy allows the tree to grow completely and then adjust it. In this paper, we use the post-pruning strategy, like C4.5 [21] and CART [2], because it is easy to obtain a desired accuracy.
2.3. Fisher Linear Discriminant

Although the decision tree method can help SVM to solve the computation problem for large data sets, the quadratic programming (4) is still slow, because SVM has to be applied to all data of each node of the decision tree [16]. Base on the geometric properties of SVM, we use Fisher linear discriminant method to find the most possible data. We only use these data for SVM training.

Considering a binary classification problem, i.e., the training data set \( X \) in (1) has \( C_L = 2 \) (two classes). Let’s separate \( X \) into two subsets \( X^+ \) and \( X^- \).

\[
X^+ = \{ x_i \in X \text{ s.t. } y_i = c_1 \} \]
\[
X^- = \{ x_i \in X \text{ s.t. } y_i = c_2 \} \]

The Fisher’s linear discriminant method searches for a vector \( \omega \) that maximizes the separation between the means of \( X^+ \) and \( X^- \), and at the same time that minimizes their scattering. In order to measure the scattering of \( X^+ \) and \( X^- \), the scatter \( \tilde{s}^2 \) for projected objects on \( \omega \) is defined as

\[
\tilde{s}^2_+ = \frac{1}{|X^+|} \sum_{i=1}^{|X^+|} (y_i - \mu_+) \\
\tilde{s}^2_- = \frac{1}{|X^-|} \sum_{i=1}^{|X^-|} (y_i - \mu_-) \\
\mu^- = \frac{1}{|X^-|} \sum_{i=1}^{|X^-|} x, \ x \in X^- \quad \mu^+ = \frac{1}{|X^+|} \sum_{i=1}^{|X^+|} x, \ x \in X^+ \] (9)

where \( |X| \) is the cardinality or number of elements in set \( X \). \( \tilde{s}^2 \) is also called within-class variance [3].

The vector \( \omega \) is obtained by solving the following optimization problem

\[
\max_{\omega} J(\omega) = \frac{|m^+ - m^-|^2}{\tilde{s}^2_+ + \tilde{s}^2_-} \tag{10}
\]

Or equivalently solving

\[
\max_{\omega} J(\omega) = \omega^T S_B \omega \\
\omega^T S_W \omega \tag{11}
\]

where

\[
m^+ = \frac{1}{|X^+|} \sum_{i=1}^{|X^+|} \omega^T x = \omega^T \frac{1}{|X^+|} \sum_{i=1}^{|X^+|} x = \omega^T \mu^+ \\
m^- = \frac{1}{|X^-|} \sum_{i=1}^{|X^-|} \omega^T x = \omega^T \frac{1}{|X^-|} \sum_{i=1}^{|X^-|} x = \omega^T \mu^-
\]

\( S_B = (\mu_- - \mu_+) (\mu_- - \mu_+)^T \) is called the between-class covariance matrix, \( S_W = \sum_{x_i \in X^+} (x_i - \mu_+) (x_i - \mu_+)^T + \sum_{x_j \in X^-} (x_j - \mu_-) (x_j - \mu_-)^T \) is called the total within-class covariance matrix.

The solution of (11) is given by

\[
\omega = S_W^{-1} (\mu_+ - \mu_-) \tag{12}
\]

This solution can be obtained in \( O(d^2 |X|) \).

An example of FLD is shown in Figure 2.

3. Support Vector Machine with Decision Tree and Fisher Linear Discriminant

According to the geometric properties of SVM, the separating hyperplane of SVM is based on the support vectors which is a small subset of whole data. The support vector are close to their opposite class, i.e., the support vectors are near to the boundaries [25] [12]
The aim of using the decision tree and the Fisher linear discriminant is to find the data which are near to the support vectors. Our classification strategy can be divided in the following steps:

1) Discover regions that contain all or most of their examples with the same label. This label is the majority class for that region.

2) For each region, we determine all its adjacent or neighbor regions whose majority class is opposite. This is because we are interested in detecting data that are located to others with opposite label. Let’s $\mathcal{T}$ represent an induced decision tree, and $\mathcal{L}_i$ a leaf of $\mathcal{T}$. The leaves of $\mathcal{T}$ can be considered the low entropy regions that we need for implementing our strategy. After training a decision tree $\mathcal{T}$, we detect the adjacent regions which contain opposite majority class in each region (leaf) $\mathcal{L}_i$. The decision tree partitions the input space such that it has lower entropy value. This partition needs $O(n \cdot d)$ time with $n$ the size of training set and $d$ the number of features.

3) Search the data with shorter distances. We apply the Fisher’s linear discriminant to each pair of adjacent low entropy regions. The use of this linear discriminant is based on the fact that for linearly separable cases, the discriminant produces similar results of SVM [23]. Now we select the closest data between these regions. The selected data are called reduced data set $X_R$. Once two adjacent regions have been detected, all the points in $\mathcal{L}_j$ are projected on vector $\omega$, see (12). The data with the shortest projections are added to the reduced set $X_R$. The total number of datum is $d \times |\mathcal{L}_j|$, where $|\mathcal{L}_j|$ is the data number in the leaf $\mathcal{L}_j$, $d$ is the dimension of the data of the induced tree $\mathcal{T}$.

It is important to notice that high entropy regions will contain support vectors, they do not need to be analyzed with Fisher linear discriminant in Step 3, but they are included directly in $X_R$. Algorithm 1 shows the pseudo code of the proposed method.

We use a simple example to show how our algorithm works. The data set used in the explanation is shown in Figure 3.

After the decision tree is trained, the input space is partitioned into six regions $\mathcal{L}_1$ to $\mathcal{L}_6$. Each region is associated to a majority class $y = \{C_1, C_2\}$, with $C_1 = +1$ or $C_2 = -1$. Table 1 shows the discovered partitions, the majority class, and the adjacent regions.

For a region $\mathcal{L}_i$, we use FLD to select the data which are close to adjacent regions $\mathcal{L}_j$, that contain different class examples. For example, for region $\mathcal{L}_4 (y = -1)$ the adjacent
Algorithm 1: General Algorithm

2 Train a decision tree \( \mathcal{T} \); // X_R Begins empty
3 \( X_R \leftarrow \text{NULL} \)
4 For each leaf \( \mathcal{L}_i \) of \( \mathcal{T} \) do
5    for each opposite class neighbor \( \mathcal{L}_j \) do
6        if entropy of \( \mathcal{L}_j \) is low then
7            //Select closest examples
8            Use \( \mathcal{L}_i \) and \( \mathcal{L}_j \) to build \( X^+ \);
9            Compute \( \omega \) (eq. (12));
10           Add \( x_i \in \mathcal{L}_j \) to \( X_R \) according to (12);
11        end if
12    end for
13 else
14    //Add all the elements in \( \mathcal{L}_j \) to \( X_R \).
15    \( X_R \leftarrow X_R \cup \mathcal{L}_j \);
16 end if
17 end for
18 return \( X_R \)
19 End

Table 1: Partition of input space and adjacent regions

<table>
<thead>
<tr>
<th>Region</th>
<th>Majority class</th>
<th>Adjacent regions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{L}_1 )</td>
<td>+1</td>
<td>( \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4 )</td>
</tr>
<tr>
<td>( \mathcal{L}_2 )</td>
<td>mixed</td>
<td>( \mathcal{L}_1, \mathcal{L}_4, \mathcal{L}_5 )</td>
</tr>
<tr>
<td>( \mathcal{L}_3 )</td>
<td>+1</td>
<td>( \mathcal{L}_1, \mathcal{L}_4 )</td>
</tr>
<tr>
<td>( \mathcal{L}_4 )</td>
<td>-1</td>
<td>( \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_5, \mathcal{L}_6 )</td>
</tr>
<tr>
<td>( \mathcal{L}_5 )</td>
<td>-1</td>
<td>( \mathcal{L}_2, \mathcal{L}_4, \mathcal{L}_6 )</td>
</tr>
<tr>
<td>( \mathcal{L}_6 )</td>
<td>+1</td>
<td>( \mathcal{L}_4, \mathcal{L}_5 )</td>
</tr>
</tbody>
</table>

Figure 3: Decision boundaries for SVM and Decision Tree classifiers

regions are \( \mathcal{L}_1, \mathcal{L}_3 \) and \( \mathcal{L}_6 \) (\( y = +1 \)). The region with high entropy value such as the \( \mathcal{L}_2 \) in Figure 3 is added directly to \( X_R \).

3.1. Adjacent regions

Detecting the adjacent regions is the key step in Algorithm 1. The leaves of an induced decision tree represent partitions or regions of input space. A decision tree splits the input space into low entropy regions, which are represented as a leaf (terminal node) in the tree’s structure. The leaves can be represented by

\[
\mathcal{L}_i = \left\{ \bigcap_{j=1}^{d} r_{ij}, l_{ij} \leq r_{ij} < h_{ij} \right\} \tag{13}
\]

where \( \mathcal{L}_i \) is the \( i-th \) leaf in the decision tree, \( d \) is the dimension number of the training set, \( r_{ij} \) is a region of input space determined by
boundaries \([l_{ij}, h_{ij}], l_{ij}, h_{ij} \in \mathbb{R}\) are the cutting points found by an induction tree algorithm.

Figure 4 shows a two-dimensional example, Figure 4: Leaf \(L_i\) with \(d = 2\).

The regions detected by a decision tree have their boundaries defined by separating hyperplanes which are orthogonal to axes [9]. These regions are the leaves of decision tree and they can be interpreted as orthogonal disjoints or hyper boxes.

**Definition 1** Two leaves \(L_o\) and \(L_p\) are candidate to be neighbors if their boundaries are changed by

\[
L_o = \left\{ \bigcap_{j=1}^d r_{oj}, \ l_{oj} \leq r_{oj} \leq h_{oj} \right\}
\]

\[
L_p = \left\{ \bigcap_{j=1}^d r_{pj}, \ l_{pj} \leq r_{pj} \leq h_{pj} \right\}
\]

**Remark 1.** Most induction tree algorithms split input space using a rule of the form \(x_i < C\). In the Definition 1, the boundaries of leaves \(L_o\) and \(L_p\) are changed to give them a connection chance.

**Definition 2** Two leaves \(L_o\) and \(L_p\) are neighbors if \(L_o \cup L_p\) is a connected space.

**Property.** Leaves \(L_o\) and \(L_p\) are neighbors if exists \(m\) with \(1 \leq m \leq d\), and the follow are satisfied

\[
h_{om} = l_{pm} \text{ or } l_{om} = h_{pm} \tag{15}
\]

\[
l_{pn} \leq l_{on} \leq h_{pn} \text{ or } l_{pn} \leq h_{on} \leq h_{pn} \tag{16}
\]

where \(1 \leq n \leq d, n \neq m\)

If two leaves \(L_o\) and \(L_p\) are neighbors, then there exists at least one point \(x_n\) that satisfies

\[
x_n \in \{r_{ok} \cap r_{pk}\}, \text{ for } k = 1, \ldots, d \tag{17}
\]

This means that \(L_o\) and \(L_p\) must necessarily share a boundary or separating hyperplane which is orthogonal to an axis of features, for example \(m\). The possible options are that the lower(upper) bound of \(L_p\) coincide with the upper(lower) bound of \(L_o\), as in (15).

**Remark 2.** The condition (15) is necessary, but not sufficient for two leaves to be neighbors. They need to share boundaries at the other dimensions. In order to fulfill (17), (16) must also be satisfied.

We give an example to explain how to find the neighbors, see Figure 5. Here \(h_{1,1} = l_{5,1}\) (the separating hyperplane orthogonal to dimension one is shared by leaves \(L_1\) and \(L_5\)), but \(L_1\) and \(L_5\) are not neighbors. Similarly, this occurs with the pairs \((L_1, L_6), (L_1, L_7), (L_2, L_7), (L_3, L_5), (L_4, L_6), (L_4, L_8)\) and \((L_5, L_3)\). In Figure 6, the neighbors of \(L_1\) are \(L_2, L_3\) and \(L_4\); the neighbors of \(L_2\) are \(L_1, L_3, L_4, L_5\) and \(L_6\).
The algorithm proposed in this paper to compute all the neighbors of a leaf includes two steps.

The first step is to determine the boundaries of each leaf, see Algorithm 2. All these boundaries are stored in a matrix $M \in \mathbb{R}^{N_L \times 2d}$, $N_L$ is the number of leaves in the induced decision tree. The $i^{\text{th}}$ row in $M$ contains the boundaries of leaf $i$. The first $d$ columns of $T$ contain the values $l_{ij}$, whereas the columns $d + 1$ upto $2d$ contain the values $h_{ij}$. The values $l_{ij}$ and $h_{ij}$ of a leaf are implemented as a vector, called $B$ which is initialized with the minimum and maximum value representation of float point variables at root node.

4. Experiment results

In this paper our algorithm is compared with SMO [20], LibSVM [6] and Pegasos [29]. SMO was developed in 1998, it is the most representative method for training SVM. SMO is currently used to compare novel algorithms to train SVM. It uses the extreme case of decomposition approach, it modifies only a subset (known as the working set) of Lagrange multipliers per iteration, the size...
Algorithm 2: Computation of leaves’ boundaries

1 Input:
2 \( T \) An induced decision tree.
3 Output:
4 \( M \) A matrix with the boundaries of all the leaves of \( T \).
5 Begin
6 Initialize \( B \) as explained in the paper;
7 call BuildBoundaries(\( B, M \)) on root node;
8 return \( M \);
9 End

Procedure BuildBoundaries(\( B, M \))
10 if current node is a leaf then
11 Insert vector \( B \) as last row into matrix \( M \);
12 else
13 Create \( B_L \) copying boundaries from \( B \);
14 \( B_L \): Change \( h_{ij} \) using attribute index and and split point value of current node;
15 call BuildBoundaries(\( B_L, M \)) on the left son of current node;
16 Create \( B_R \) copying boundaries from \( B \);
17 \( B_R \): Change \( l_{ij} \) using attribute index and and split point value of current node;
18 call BuildBoundaries(\( B_R, M \)) on the right son of current node;
19 end if
20 return

Algorithm 3: Neighbors of a leaf

1 Input:
2 \( M \) matrix of leaf’s boundaries.
3 \( \mathcal{L}_p \) A leaf
4 Output:
5 \( N \) The neighbors of leaf \( \mathcal{L}_p \)
6 Begin
7 for each attribute \( m \)
8 Get all leaves that satisfy (15);
9 for each leaf \( \mathcal{L}_o \) gotten in the previous step
10 if \( \mathcal{L}_o \) satisfies (16) then
11 Add to list \( N \);
12 end if
13 end for
14 end for
15 Remove repeated elements in \( N \);
16 return \( N \);
17 End
of working set is two. This leads to a small sub-problem to be minimized in each iteration. Because only two variables are involved in the optimization problem, it is not necessary to use any optimization software. In general, an advantage of decomposition methods is that they can deal with large data sets, because only a portion of training set is loaded in memory. A disadvantage of decomposition is that the convergence can be slow because selecting elements that form the working set is costly.

LibSVM is a library updated in 2011 and used to train SVM with large data sets, it uses an method that implements an SMO-type algorithm. The main improvement of LibSVM over SMO is that the former uses a Hessian matrix (second order information) to select the elements of working set by detecting the points that most violate the Karush-Kuhn-Tucker conditions. Currently LibSVM one of the fastest methods for training SVM, it outperforms PSVM [17] and LS SVM [24].

Pegasos [29] is an algorithm developed in 2007 which alternates between stochastic gradient descent steps and projection steps. The algorithm has a small training times for large data sets. Pegasos achieves state-of-the-art results if Linear kernel is used, in the case of non linear kernel, Pegasos finds an approximated solution of the QP problem.

All original data are applied to SMO and LibSVM training. The experiments are run on a computer with the following features: Core i7 2.2 GHz processor, 8.0 GB RAM, Windows 7 ultimate operating system. The algorithms are implemented in the Java language. The maximum amount of random access memory given to the Java virtual machine is set to 2.0 GB. The results obtained in this paper correspond to 100 runs of each experiment. For each experiment, the training data are chosen randomly from 70% of the data set, the rest data are used for testing. The kernel used in all experiments is a radial basis function

\[ f(x, z) = \exp\left(-\frac{(x - z)^T(x - z)}{2\gamma^2}\right) \] (18)

where \( \gamma \) was selected using the grid search method.

4.1. Datasets

In this paper we use 18 data sets to compare our method with the other ones. Nine data sets are public available, they are Haberman’s survival, Ionosphere, Breast cancer, Diabetes, Four-class, IJCNN-1, Bank marketing, Cod-RNA, and Skin Segmentation. Three data sets are modified for binary classification problems, they are Checkerboard [11] (see Figure 7), Iris, and Waveform. One data set is created artificially, it is Rotated cross (see Figure 8).

The Table 2 shows a summary of the data sets used in the experiments. The Size is the number of examples in data set; Dim is the number of features; \( |y_i = +1| \) and \( |y_i = -1| \) is the number of data with label +1 and −1 respectively.

4.2. Parameters

The parameters of our algorithms are the minimum number of objects in each leaf of
Figure 8: Rotated-cross data set

In order to achieve the best classification accuracy, the parameters of our algorithm need to be tuned. Table 3 shows the values used for each data set in the experiments. These values are selected by a grid searching method, which is widely used in the literature [3][23][24].

In Table 3, the regularization parameter for the QPP and the gamma for the kernel are represented by $C$ and $\gamma$ respectively.

4.3. Results

The performances of SMO, LibSVM and our algorithm DTSVM are shown in the Table 4.

It can be seen that the training time is improved in practically all cases with our method DTSVM. For small data sets, the time saving is not considerable. However, when the training sets are large, the computation burden is very low. The accuracies achieved by our algorithm are slightly degraded, but they are still acceptable. These are caused by the fact that some SV are not selected during the pre-processing step. The parameter $C$ in our method is larger than LibSVM and SMO, because the training data of algorithm have more mixed samples. It is necessary to penalize the selected samples to obtain a good accuracy level. The size of training sets are reduced in about 80% to 90% for most cases. This reduction makes the training procedure faster for the large data set.

The figure 9 shows the training times, these correspond to data sets larger than 1,000. It can be seen that our DTSVM easily outperforms LibSVM and SMO. Pegasos has almost the same training time than our method. In some cases the training time using DTSVM is no better than Pegasos, however the accuracy obtained with our method is in general higher than the obtained with Pegasos, this can be seen in the figure 10.

Table 2: Datasets for experiments

| Dataset            | Size | Dim | $|y_i = +1|$ | $|y_i = -1|$ |
|--------------------|------|-----|------------|------------|
| Iris-setosa        | 100  | 4   | 50         | 50         |
| Iris-versicolor    | 100  | 4   | 50         | 50         |
| Iris-virginica     | 100  | 4   | 50         | 50         |
| Haberman’s Survival| 306  | 3   | 225        | 81         |
| Ionosphere         | 351  | 34  | 126        | 225        |
| Breast-Cancer      | 683  | 10  | 444        | 239        |
| Diabetes           | 768  | 8   | 500        | 268        |
| Four-class         | 862  | 2   | 307        | 555        |
| Waveform-0         | 3,308| 40  | 1,653      | 1,655      |
| Waveform-1         | 3,347| 40  | 1,692      | 1,655      |
| Waveform-2         | 3,347| 40  | 1,692      | 1,653      |
| ijcnn1             | 35,000| 22  | 3,415      | 31,585     |
| Bank marketing     | 45,211| 16  | 39,922     | 5,289      |
| Cod-rna            | 59,535| 8   | 19,845     | 36,690     |
| Cross rotated      | 90,000| 2   | 50,000     | 40,000     |
| Checkerboard100K   | 100,000| 2   | 50,000     | 50,000     |
| Skin Segmentation  | 245,057| 3   | 50,859     | 194,198    |
Table 3: Value of parameters for the experiments

<table>
<thead>
<tr>
<th>Method</th>
<th>Dataset</th>
<th>C</th>
<th>( \gamma )</th>
<th>( \delta )</th>
<th>Min obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>LibSVM</td>
<td>Iris-setosa</td>
<td>3.00</td>
<td>3.00</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>2.00</td>
<td>1</td>
<td>0.10</td>
<td>2</td>
</tr>
<tr>
<td>LibSVM</td>
<td>Iris-versicolor</td>
<td>1.00</td>
<td>3.00</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>3.00</td>
<td>1</td>
<td>0.10</td>
<td>2</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>3.00</td>
<td>1</td>
<td>0.15</td>
<td>2</td>
</tr>
<tr>
<td>LibSVM</td>
<td>Haberman’s survival</td>
<td>3.00</td>
<td>2.00</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>2.00</td>
<td>2.00</td>
<td>0.30</td>
<td>15</td>
</tr>
<tr>
<td>LibSVM</td>
<td>Ionosphere</td>
<td>2.50</td>
<td>1.55</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>3.00</td>
<td>2.00</td>
<td>0.50</td>
<td>2</td>
</tr>
<tr>
<td>LibSVM</td>
<td>Breast-cancer</td>
<td>1.00</td>
<td>1/( n )</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>1.50</td>
<td>1/( n )</td>
<td>0.50</td>
<td>10</td>
</tr>
<tr>
<td>LibSVM</td>
<td>Diabetes</td>
<td>1.00</td>
<td>1/( n )</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>2.50</td>
<td>2.00</td>
<td>0.15</td>
<td>5</td>
</tr>
<tr>
<td>LibSVM</td>
<td>Four-class</td>
<td>2.00</td>
<td>3.00</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>20.00</td>
<td>3.50</td>
<td>0.15</td>
<td>2</td>
</tr>
<tr>
<td>LibSVM</td>
<td>Waveform-0</td>
<td>1.00</td>
<td>1/( n )</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>2.50</td>
<td>1/( n )</td>
<td>0.35</td>
<td>35</td>
</tr>
<tr>
<td>LibSVM</td>
<td>Waveform-1</td>
<td>1.00</td>
<td>1/( n )</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>2.50</td>
<td>1/( n )</td>
<td>0.35</td>
<td>35</td>
</tr>
<tr>
<td>LibSVM</td>
<td>Waveform-2</td>
<td>1.00</td>
<td>1/( n )</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>2.50</td>
<td>1/( n )</td>
<td>0.35</td>
<td>35</td>
</tr>
<tr>
<td>LibSVM</td>
<td>ijcnn1</td>
<td>3.00</td>
<td>1/( n )</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>5.00</td>
<td>1/( n )</td>
<td>0.10</td>
<td>30</td>
</tr>
<tr>
<td>LibSVM</td>
<td>Bank marketing</td>
<td>3.50</td>
<td>2.3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>4.30</td>
<td>1/( n )</td>
<td>0.05</td>
<td>40</td>
</tr>
<tr>
<td>LibSVM</td>
<td>cod-ma</td>
<td>4.50</td>
<td>3.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>10.50</td>
<td>3.5</td>
<td>0.35</td>
<td>100</td>
</tr>
<tr>
<td>LibSVM</td>
<td>Rotated Cross</td>
<td>2.00</td>
<td>1/( n )</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>3.00</td>
<td>1/( n )</td>
<td>0.15</td>
<td>20</td>
</tr>
<tr>
<td>LibSVM</td>
<td>Checkerboard100K</td>
<td>2.00</td>
<td>3.00</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>20.00</td>
<td>3.50</td>
<td>0.35</td>
<td>2</td>
</tr>
<tr>
<td>LibSVM</td>
<td>Rotated Cross</td>
<td>1.00</td>
<td>1/( n )</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DTSVM</td>
<td></td>
<td>2.00</td>
<td>1/( n )</td>
<td>0.15</td>
<td>25</td>
</tr>
</tbody>
</table>

5. Conclusions

This paper proposes a novel method for SVM classification, DTSVM. By using decision trees (DT) and Fisher linear discriminant (FLD), DTSVM overcomes the problems of slow training of SVM and low accuracy of many DT and FLD based SVMs. The key point of our method is to find the low entropy regions and opposite class regions which are closed to decision boundaries. Experimental results demonstrate that our approach has good classification accuracy while the training is significantly faster than other SVM classifiers. We compared DTSVM with other three methods of the state of the art, they are SMO, LibSVM library and Pegasos. Our method outperforms all these methods in training time and/or accuracy.

For small data sets, the accuracy of DTSVM is maintained slightly lower than the classical SVM classifiers which use the whole data set. However, in the case of larger data sets, the classification accuracy are almost the same or even better than the other SVMs, because we do not use soft margin method to deal with the misleading points.

[1] M. Berg, O. Cheong, M. Kreveld,
Figure 10: Comparative of accuracies. The LibSVM and SMO run out of memory for the bank marketing data set.


