Solving Large-Scale Multi-Label SVM Problems with A Tree Decomposition Approach

Fu Chang  
Institute of Information Science  
Academia Sinica  
Taipei, Taiwan  
FCHANG@IIS.SINICA.EDU.TW

Chan-Cheng Liu  
Institute of Information Science  
Academia Sinica  
Taipei, Taiwan  
CHENG@IIS.SINICA.EDU.TW

Editor:

Abstract

We propose a tree decomposition approach for solving large-scale multi-label classification problems. In this approach, we follow a convention of first transforming the problem into a number of one-against-others classification problems. Then, to solve each transformed problem, we use a decision tree to decompose the corresponding data space and train local SVMs on the decomposed regions. The resultant classifier is called decision tree support vector machine (DTSVM). The approach has the following advantages. First, when non-linear SVM is used as the learning machine, DTSVM requires much shorter training time than global SVM (gSVM) while achieving comparable test accuracy. Second, when linear SVM is used as the learning machine, DTSVM often achieves higher test accuracy than gSVM. Third, in textual applications, linear DTSVM plays the leading role among linear gSVM, non-linear gSVM, and non-linear DTSVM for the following reason. Linear DTSVM achieves higher test accuracy than linear gSVM; moreover, it achieves comparable test accuracy to, but requires much less training time than, non-linear gSVM and non-linear DTSVM.

Keywords: DTSVM, Decision Tree Support Vector Machine, Large-Scale Problem, Multi-Label Classification, Tree Decomposition

1. Introduction

Multi-label classification (MLC) deals with samples that can have multiple labels, as opposed to single-label classification (SLC), which handles samples that have only one label. Thus, in classifying a test sample, an MLC solution comprises a variable number of labels, while an SLC solution consists of a single label. MLC was initially used in text categorization (cf. Aas and Eikvil, 1999 for a broadly-cited survey), and numerous research works applied it to Reuters-21578, a collection of articles compiled from Reuters newswires in 1987. Subsequently, a much larger set of articles was compiled and published by Lewis et al., (2004). We apply the proposed method to that set, in addition to some other data sets. Other applications of MLC include music categorization (Li and Oghihara, 2003), protein...
function classification (Zhang and Zhou, 2005), and semantic scene analysis (Boutell et al., 2004).

Many methods have been proposed to solve MLC problems, most of them are modifications of existing methods for solving SLC problems. These methods include SVM (Joachims, 1998; Godbole and Sarawagi, 2004), expectation-maximization (McCallum, 1999), AdaBoost (Schapire and Singer, 2000), C4.5 (Clare and King 2001), and k-nearest neighbor (Zhang and Zhou, 2005; Luo and Zincir-Heywood, 2005). A number of methods have also been proposed to solve related problems. For example, to solve ranking problems, Elisseeff and Weston (2002) employed a margin-based approach that ranks all possible labels for a given sample; while Thabtah et al. (2004) adopted an associative classification approach to construct sets of classification rules.

Tsoumakas and Katakis (2007) and Tsoumakas et al. (2009), besides providing an excellent survey of the above methods, discussed six ways to transform an MLC problem into other problems. The first way transforms, rather unsystematically, all multi-label samples in the training data set into single-label samples. The second way simply removes all multi-label samples from the training data set. The third way transforms an MLC problem with $L$ labels into $2^L$ SLC problems, each of which is used to decide whether to assign a subset of labels to a given test sample, as Mencía and Fürnkranz (2008). The fourth way transforms an MLC problem with $L$ labels into $L$ SLC problems, each of which decides whether to assign one label to a given test sample. The first two ways can lose precious information in the training, while the third way requires an enormous amount of training time. The fourth way is reasonable and has motivated many research works, including Joachims (1998); Goncalves and Quaresma (2003); Lauser and Hotho (2003); Li and Ogihara (2003); and Boutell et al. (2004). The fifth way duplicates a sample with $l$ labels to form $l$ single-label samples, and then builds a classifier that assigns probabilities to all the labels for each test sample. The sixth way assumes that labels are correlated densely and some priori knowledge is given; under this assumption, a modified primal formulation is proposed to reduce the complexity from exponential to linear (Hariharan et al., 2010).

The method described in this paper adopts the fourth transformation method. Thus, for an MLC problem with $L$ labels, we derive $L$ single-label data sets from a multi-label data set. Each single-label data set is associated with a label, say $y$, whose positive samples are derived from multi-label samples that have label $y$; and whose negative samples are derived from those samples that do not have label $y$. If we formulate an SVM problem on each single-label data set, we then have $L$ single-label SVM problems. To solve these problems, a straightforward approach is to simply train $L$ classifiers on the $L$ data sets. When dealing with large-scale data sets, an effective tool, such as LIBLINEAR (Fan et al., 2008), can be used for linear training. However, an effective tool for non-linear training has yet to be constructed.

The tree decomposition method has been applied to single-label SVM problems with successful outcomes (Chang et al., 2010). The standard way to solve such problems involves building global SVMs (gSVMs) on the full training data set. However, the tree decomposition approach first builds a binary decision tree (Breiman et al., 1984; Quinlan, 1986) to decompose the data space and then trains local SVMs (lSVMs) on the decomposed regions, namely the tree leaves. The resultant classifier, comprised of the binary tree and all the lSVMs built on the tree leaves, is called a decision-tree SVM (DTSVM). By applying this
method to various single-label data sets, it is possible to train non-linear DTSVMs at a much faster rate and achieve a comparable test performance to that of non-linear gSVMs.

To solve multi-label SVM problems, we adopt the tree decomposition approach. There are, however, two crucial differences between the MLC version and the SLC version of the tree decomposition approach. When solving an SLC problem, we obtain one DTSVM from the training process. In contrast, when solving an MLC problem, we first transform the problem into a number of SLC problems. We then apply the tree decomposition technique to each SLC problem and obtain a DTSVM for each label. More specifically, in an SLC problem, there is a single DTSVM in the solution, but it may have multiple (one-against-one or one-against-others) lSVMs in each decomposed region. In an MLC problem, there are multiple DTSVMs in the solution, while each DTSVM has a single lSVM in each decomposed region. This marks the first difference between the SLC and MLC versions.

To describe the second difference between the two versions, we have to say more about the learning process for DTSVMs. When training a DTSVM, we need to determine the sizes of decomposed regions, which are controlled by a parameter, $\sigma$. The optimal value of $\sigma$ is determined in an iterative manner. Initially, we set $\sigma = \sigma_0$ and build lSVMs on the tree leaves that are smaller than $\sigma_0$. In each of the subsequent iterations, we quadruple the size of $\sigma$ and again build lSVMs on the tree leaves that are smaller than $\sigma$, until $\sigma$ exceeds the size of all the training samples. In SLC training, the process can be terminated earlier if the validation performance obtained in the current iteration does not improve significantly on that obtained in the previous iteration.

We perform MLC training rather differently for the following reason. In our experiments, we observed that the single-label data sets derived from certain multi-label data sets contained very few positive samples. Consequently, we need to use a performance measure to strike a balance between positive and negative samples. When such a measure is adopted, we found that the DTSVM performance obtained for different values of $\sigma$ did not change in a monotonically increasing fashion. Thus, if we adopted the stop criterion used in SLC learning, we could terminate the learning process in an early iteration and result in a local maximum in the validation performance. To avoid this problem, we require that each MLC learning process should continue until the maximum possible level of $\sigma$ is reached. We then select the value of $\sigma$ that yields the maximum validation performance.

At the first glance, the new learning algorithm does not seem to be very efficient because it must try all possible values of $\sigma$. However, this requirement provides two gains. First, we achieve the best possible validation performance, thereby achieving a good test performance as a consequence. Second, we save time in searching for the optimal parameter values of lSVMs. The second gain derives from the fact that we only try all possible lSVM parameter values at the lowest value of $\sigma$. For larger values of $\sigma$, we only try a few top-ranked parameter values derived from the lowest value of $\sigma$. The time saving is enormous for non-linear training, since training non-linear lSVMs on a low value of $\sigma$ is much faster than on a higher value.

We now summarize the insights gained through our experiments, which were conducted on five large-scale data sets. The sample sizes were between 28K and 881K and the number of labels was between 22 and 103. We conducted both linear and non-linear training (i.e., we trained linear and non-linear SVMs). In addition, we measured the performance in terms of macro-F1 and micro-F1 scores, which we define later. For non-linear training,
DTSVMs required a much smaller amount of training time than gSVMs. For example, DTSVMs were 27 times faster than gSVMs on a data set comprising 19,068 training samples, and 474 times faster on a data set comprising 28,119 training samples (cf. Figure 7). DTSVMs also achieved higher or comparable test performances than gSVMs. For linear training, DTSVMs often achieved significantly higher test performances than gSVMs, although DTSVMs were built at a somewhat higher computational expense. The better test performance achieved by both linear and non-linear DTSVMs implies that all the data sets in our experiments are either non-linear or at least piecewise linear, rather than globally linear. We believe this result is interesting and warrants further investigation.

The remainder of this paper is organized as follows. In Section 2, we describe the learning algorithm for DTSVMs. In Section 3, we discuss the experimental results derived from DTSVMs and gSVMs. Section 4 contains some concluding remarks.

2. The Learning Process

Assume that we are given a set of multi-label training samples $X = \{(x_i, Y_i) : i = 1, 2, \ldots, n\}$, where $x_i$ is a $d$-dimensional vector and $Y_i$ is the set of labels associated with $x_i$. Let $L$ be the total number of labels. The $L$ single-label data sets derived from $X$ are $X_y$’s, $y = 1, 2, \ldots, L$. Each $X_y$ comprises $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where $y_i = 1$ if $y \in Y_i$; otherwise, $y_i = -1$. We construct a DTSVM on each $X_y$. In this section, we explain the learning process under our approach. The training of a binary decision tree is described in Section 2.1, and algorithm for building lSVMs on the tree leaves is presented in Section 2.2. The implementation of the learning process is available at [http://ocrwks11.iis.sinica.edu.tw/~dar/Download/WebPages/MLDT-SVM.htm](http://ocrwks11.iis.sinica.edu.tw/~dar/Download/WebPages/MLDT-SVM.htm), along with the source code, execution file, readme file, and data sets used in our experiments.

2.1 Training a Binary Decision Tree

To train a binary tree, we start with the root, which takes all the training samples as input. We then decide whether to send each sample to the left-hand or right-hand child node. The same procedure is repeated for each of the root’s child nodes in a recursive manner.

At a given node $E$, we pick a feature $f_E$ and a split point $v_E$ so that all elements of $E$ with $f_E < v_E$ are sent to the left-hand child node, and the remaining elements are sent to the right-hand child node. The values of $f_E$ and $v_E$ are determined as follows.

For each feature $f$, we define the split point $v_f$ associated with $f$ as the value $v$ that maximizes the information gain $IG_E(f, v)$. We then choose feature $f_E$ as the feature $f$ that maximizes $IG_E(f, v_f)$ and $v_E$ as the split point associated with $f_E$.

Intuitively, the information gain $IG_E$ measures how much uncertainty is removed by sending elements of $E$ to the two child nodes of $E$. To quantify the uncertainty of a set of labeled samples, $S$, we use the following measure:

$$U(S) = -\sum p(S_y) \log p(S_y)$$

where $p(S_y)$ is the proportion of $S$’s samples labeled $y$. The information gain is then defined as
\[ IG_E(f, v) = U(E) - \frac{|E_L|}{|E|} U(E_L) - \frac{|E_R|}{|E|} U(E_R) \]

where \( E_L \) consists of the elements of \( E \) with \( f < v \); \( E_R \) consists of the remaining elements of \( E \); and \(|S|\) is the size of any set \( S \).

We stop splitting the node \( E \) when \( IG_E(f, v) = 0 \) for all \( f \) and \( v \) at \( E \). This condition often occurs in the following situations: (1) when all the samples flowing to \( E \) are homogeneous, i.e., all samples have the same label; or (2) when a subset of samples is homogeneous and the remaining samples, although carrying different labels, are identical to some members of the homogeneous subset.

2.2 Training lSVMs on Tree Leaves

After growing a tree, we train lSVMs on its leaves using the samples that flow to each leaf as training data. However, if a leaf contains homogeneous samples, we do not build an lSVM on it. In the testing phase, when a test sample flows to a homogeneous leaf, the sample is automatically assigned the common label of that leaf. The resulting tree and all the lSVMs built on the leaves constitute a DTSVM.

The parameters associated with a DTSVM are: (i) \( \sigma \), the ceiling size of the decision tree; and (ii) the SVM parameters. For a given value of \( \sigma \), all the lSVMs in the DTSVM take the same SVM parameter values. The optimal values of the above parameters are determined as follows. Assuming that a training data set and a validation data set are given, we build DTSVMs on the training data set with various parameter values and determine the optimal parameter values with the help of the validation data set. The training process proceeds as follows.

In the initial stage, we train a binary tree with an initial ceiling size \( \sigma_0 \), and then train all the lSVMs with the same SVM parameters \( \theta \). We express \( \theta \) in boldface to indicate that it may consist of more than one parameter. Let \( v(\sigma_0, \theta) \) be the validation performance of the resultant DTSVM measured on the validation data set. In our experiments, we set \( \sigma_0 = 1,500 \).

In the subsequent stages, we construct DTSVMs with a larger ceiling size, but we only train their lSVMs with the top-ranked \( \theta \). To do this, we rank \( \theta \) in descending order of \( v(\sigma_0, \theta) \). Let \( \Theta_{[k]} \) be the set containing the \( k \) top-ranked \( \theta \). In our experiments, we set \( k = 5 \) for non-linear training and \( k = 3 \) for linear training.

More specifically, in stage \( t \), we set \( \sigma_t = 4\sigma_{t-1} \) for \( t = 1, 2, \ldots \). We modify the tree with ceiling size \( \sigma_{t-1} \) to obtain a tree with ceiling size \( \sigma_t \). This is done by moving from the root towards the leaves and retaining each node whose parent’s size is greater than \( \sigma_t \). Then, we train new DTSVMs on the modified tree with \( \theta \) chosen from \( \Theta_{[k]} \) and calculate the validation performance \( v(\sigma_t, \theta) \) for \( \theta \in \Theta_{[k]} \). We terminate the iterative process when \( \sigma_t \) exceeds the size of the training samples.

When the process is terminated, we select the optimal values for the ceiling size and the SVM parameters to be the \( \sigma_t \) and the \( \theta \) in \( \Theta_t \) that maximize \( v(\sigma_t, \theta) \), where \( \Theta_0 = \Theta \) and \( \Theta_t = \Theta_{[k]} \) for \( t > 0 \). If multiple sets of parameters achieve the maximum validation performance, we choose the set derived from the earliest iteration. For simplicity, we present the initial and subsequent stages in terms of the following pseudo-codes.
3. Experimental Results

We conducted our experiments on five data sets: "tmc2007" (Srivastava and Zane-Ulman, 2005), "Mediamill" (Snoek et al., 2006), and three others derived from "RCV1" (Lewis et al., 2004) by restricting the sample sizes to 100K, 250K, and 881K (the full size), respectively. "Tmc2007" and "RCV1" originated from textual applications, while "Mediamill" originated from a multimedia application.

First, we randomly divided each data set into six parts of approximately equal size. We then used four of the parts as the training component, another part as the validation component, and the remaining part as the testing component. We built classifiers, i.e., gSVMs and DTSVMs, on the training component, and computed their validation performance on the validation component. We were then able to determine the optimal values of parameters. Finally, using the test component we computed the test performance of the classifiers with the optimal parameters. Table 1 contains the information for each data set, including the number of labels, the number of features, the size of the data set, and the sizes of the training, validation, and testing components.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>L</th>
<th>F</th>
<th>D</th>
<th>$T_r$</th>
<th>V</th>
<th>$T_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMC2007</td>
<td>22</td>
<td>30,438</td>
<td>28,596</td>
<td>19,068</td>
<td>4,765</td>
<td>4,763</td>
</tr>
<tr>
<td>MEDIAMILL</td>
<td>101</td>
<td>120</td>
<td>42,177</td>
<td>37,119</td>
<td>7,029</td>
<td>7,029</td>
</tr>
<tr>
<td>RCV1-100K</td>
<td>103</td>
<td>47,236</td>
<td>100,000</td>
<td>66,666</td>
<td>16,667</td>
<td>16,667</td>
</tr>
<tr>
<td>RCV1-250K</td>
<td>103</td>
<td>47,236</td>
<td>250,000</td>
<td>166,666</td>
<td>41,667</td>
<td>41,667</td>
</tr>
<tr>
<td>RCV1-FULL</td>
<td>103</td>
<td>47,236</td>
<td>881,264</td>
<td>587,508</td>
<td>146,878</td>
<td>146,878</td>
</tr>
</tbody>
</table>

Table 1: The data sets used in our experiments; L = the number of labels, F = the number of features, D = the size of the data set, $T_r$ = the size of the training component, V = the size of the validation component, and $T_s$ = the size of the testing component.

3.1 Methods Compared

In the experiments, we compared two methods for solving MLC problems. The first method builds as many gSVMs as the number of labels; and the second method builds as many as DTSVMs as the number of labels.
When constructing gSVMs and DTSVMs, we conducted both linear and non-linear training. For linear training, we employed LIBLINEAR (Fan et al., 2008). The parameter involved in linear training was the cost factor $C$, whose values were taken from $\Phi = \{10^a : a = -1, 0, \ldots, 5\}$. Hence, there were seven values for $C$. For non-linear training, we employed LIBSVM (Fan et al., 2005). In addition, we used an RBF kernel function to express the similarity between vectors. As a result, there were two parameters involved in non-linear training: the cost factor $C$ and the $\gamma$ parameter in the RBF function. The values of $C$ were taken from $\Phi$; and the values of $\gamma$ were taken from $\Psi = \{10^b : b = -4, -3, \ldots, 4\}$. Thus, there were 63 pairs of values for $(C, \gamma)$.

In the MLC framework, there are as many single-label classifiers as the number of parameters. This raises the following question. When searching for the optimal values of SVM parameters, how large should the search range be? Assuming that the total number of labels is $L$, then the maximum size for the search range is $\Phi^L$ for linear training, and $\Phi^L \Psi^L$ for non-linear training. This enormous search range incurs a high computational overhead and may result in overfitting in the end. Therefore, following the usual practice, we require that all single-label classifiers should be built using the same value of $C$ for linear training, and the same values of $(C, \gamma)$ for non-linear training. This limits the size of the search range to $\Phi$ for linear training, and $\Phi \Psi$ for non-linear training.

3.2 Performance Measures

For both methods compared in our experiments, there is a single-label classifier $C_y$ for each $y$ that assigns 1 or -1 to each sample $x$. The labels assigned to $x$ will be all the $y$’s for which $C_y$ assigns 1 to $x$. To assess the performance of the label assignment, we employ two types of measures: macro-F1 ($macro$) and micro-F1 ($micro$) scores (Tague, 1981). The $macro$ score is defined from the viewpoint of labels. So for each label $y$, we first define the following two quantities.

\[
Recall(y) = \frac{\text{the number of samples to which } y \text{ is correctly assigned}}{\text{the number of samples that possess label } y}
\]

\[
Precision(y) = \frac{\text{the number of samples to which } y \text{ is correctly assigned}}{\text{the number of samples that are assigned label } y}
\]

The $macro$ score is then defined as follows.

\[
Macro = \frac{1}{L} \sum_{y=1}^{L} \frac{2 \times Precision(y) \times Recall(y)}{Precision(y) + Recall(y)}
\]

The $micro$ score is defined from the viewpoint of samples. Thus for each multi-label sample $x$, we first define the following two quantities.

\[
Recall(x) = \frac{\text{the number of labels correctly assigned to } x}{\text{the number of labels possessed by } x}
\]

\[
Precision(x) = \frac{\text{the number of labels correctly assigned to } x}{\text{the number of labels assigned to } x}
\]

The $micro$ score is then defined as follows.
Micro = \frac{1}{n} \sum_{i=1}^{n} \frac{2 \times \text{Precision}(x_i) \times \text{Recall}(x_i)}{\text{Precision}(x_i) + \text{Recall}(x_i)}

We note that if \( x \) is a test sample and no label is assigned to it, then \( \text{Recall}(x) \) is zero and the contribution of \( x \) to micro is also zero. Thus, if our goal is to derive the highest micro score, we do not lose anything by assigning a label, say \( z \), to \( x \) for the following reason. If \( z \) is incorrectly assigned to \( x \), \( \text{Recall}(x) \) is still zero; however, if \( z \) is correctly assigned to \( x \), \( \text{Recall}(x) \) becomes positive and the contribution of \( x \) to micro also becomes positive.

For the first method studied in our experiments, each \( C_y \) is a gSVM. If all \( C_y \) assign -1 to \( x \), we choose \( z \) as the label for which \( C_z \) achieves the highest objective value among all \( C_y \)'s. This choice maximizes \( x \)'s probability of contributing positively to the micro score. For the second method, \( C_y \) is a DTSVM. In this case, if all the leaves that \( x \) flows to are homogeneous, we assign an arbitrary \( z \) to \( x \); otherwise, we choose \( z \) as the label for which the associated ISVM achieves the highest objective value.

Similarly, if our objective is to derive the highest macro score, we assign label \( y \) to an arbitrary test sample provided that it has not been assigned to any other test sample. However, this is highly unlikely because, in principle, the number of test samples is infinite and the probability that a label has not been assigned to any of them is virtually zero.

Finally, we consider how the performance measure affects our choice of the optimal ceiling size for each DTSVM\( y \), where DTSVM\( y \) is the classifier associated with label \( y \). If our goal is to optimize the macro score, then it is clear that we should choose the optimal ceiling size as the size at which DTSVM\( y \) achieves the highest value of the following measure:

\[
m(y) = \frac{2 \times \text{Precision}(y) \times \text{Recall}(y)}{\text{Precision}(y) + \text{Recall}(y)}
\]

Meanwhile, to optimize the micro score, in principle, we must assess the interactions between all DTSVM\( y \). However, we want to avoid such complexity and simply evaluate how a ceiling size for DTSVM\( y \) affects samples with or without \( y \). For this reason, we still choose the optimal ceiling size as the size at which DTSVM\( y \) achieves the highest value of \( m(y) \).

As a consequence, when we optimize a different performance measure, we choose the optimal ceiling size for DTSVM\( y \) in the same way but we choose the optimal values of the SVM parameters in a different way.

### 3.3 Performance Comparison under the Macro-F1 Measure

Figures 1-3 show the results of applying linear gSVMs (L-gSVMs), linear DTLsVMs (L-DTSVMs), non-linear gSVMs (N-gSVMs), and non-linear DTSVMs (N-DTSVMs) to the five data sets, using the macro score as the performance measure. Figure 1 shows the macro scores obtained by the above classifiers, while Figure 2 and Figure 3 show their training times and testing times respectively. We did not apply N-gSVMs and N-DTSVMs to the last two RCV data sets due to the large amount of time required to build them. The training times reported in Figure 2 include the time required to build the classifiers and search for optimal parameters, but not the time to input or output data. All the computations were performed on a Quad-Core Intel Xeon X3550 processor with a 2.33GHz CPU and 32GB
RAM. To overcome the huge training time of N-gSVMs, we employed multiple computers to train N-gSVMs with different SVM parameters c. The training time is then derived as the sum of all the time consumptions.

We now summarize the results shown in Figures 1-3.

1. N-DTSVMs achieved comparable macro test scores to N-gSVMs. On the other hand, N-DTSVMs required much less time than N-gSVMs for training.

2. L-DTSVMs often achieved better macro test scores to L-gSVMs, although L-DTSVMs required more time than L-gSVMs for training. On the other hand, L-DTSVMs required a much less time than N-DTSVMs for training.

3. L-DTSVMs required a bit more time than L-gSVMs for testing, but N-DTSVMs needed less time than N-gSVMs for testing.

4. On ”Mediamill”, the non-linear classifiers (i.e., N-DTSVMs and N-gSVMs) achieved much higher macro test scores than linear classifiers (i.e., L-DTSVMs and L-gSVMs). This makes N-DTSVMs leading performers on this data set, since they performed comparably to, but required much shorter training time than N-gSVMs. The macro test scores are in general lower than the micro test scores, reflecting the fact that the macro score is a more strict measure than the micro score.

5. On the remaining data sets, L-DTSVMs were the leading performers, since they performed comparably to, but required much less training time than, the non-linear classifiers. Moreover, L-DTSVMs outperformed L-gSVMs.

As described in Section 2.2, the optimal size of $\sigma$ for each DTSVM is determined by a validation process. It would be interesting to know how this size varies from one DTSVM to another. We analyzed the information gathered from applying L-DTSVMs to ”RCV1-Full”. Figure 4 shows that 19 L-DTSVMs were built on tree leaves with a ceiling size of 1,500, 34 L-DTSVMs were built on tree leaves with a ceiling size of 6,000, etc. Note that only four L-DTSVMs were built on the full set of training data.

In all of our multiple-label data sets, the number of samples that possess a certain label is much smaller than the number of samples that do not possess that label. As a result, the single-label data sets derived from a multi-label data set have a very small amount of positive samples. Thus, it would be interesting to know how the minority ratio, defined as the ratio of minority (positive) samples to majority (negative) samples, impacts the performance of gSVMs or DTSVMs. Once again, we analyzed the information gathered from ”RCV1-Full”. Figure 5 shows, for example, that L-gSVMs achieved an average macro score of 38.7% on the single-label data sets whose minority ratios were between 0.00001 and 0.0001; while L-DTSVMs achieved an average macro score of 43.3% on the same data sets. From the results, we conclude that L-DTSVMs obtained higher average macro scores than L-gSVMs for all ranges of the minority ratio. Moreover, the performance gains of L-DTSVMs over L-gSVMs were higher than 4% on some ranges.

### 3.4 Performance Comparison under the Micro-F1 Measure

Figures 6-8 show the results of applying gSVMs and DTSVMs to the five data sets, using the micro score as the performance measure. Specifically, Figure 6 shows the micro scores
Figure 1: The macro scores obtained by gSVMs and DTSVMs.

Figure 2: The training times required by gSVMs and DTSVMs when the macro score is used as the performance measure.
Figure 3: The testing times required by gSVMs and DTSVMs when the macro score is used as the performance measure.

Figure 4: Ceiling-size variation of L-DTSVMs on "RCV1-Full" when the macro score is used as the performance measure.

Figure 5: The average macro scores achieved by L-gSVMs and L-DTSVMs on single-label data sets whose minority ratios are within a certain range.
obtained by the classifiers, while Figure 7 and Figure 8 show their training times and testing times respectively. Once again, we did not apply N-gSVMs and N-DTSVMs to the last two RCV data sets due to the large amount of time required to build them. Figure 9 shows the variation in the ceiling size for L-DTSVMs on "RCV1-Full"; and Figure 10 shows the average micro scores obtained by L-gSVMs and L-DTSVMs on various collections of samples from "RCV1-Full" with a certain number of labels.

We now summarize the results shown in Figures 6-10.

i N-DTSVMs achieved comparable micro test scores to N-gSVMs. On the other hand, N-DTSVMs required much less time than N-gSVMs for training.

ii L-DTSVMs often achieved better micro test scores to L-gSVMs, although L-DTSVMs required more time than L-gSVMs for training. On the other hand, L-DTSVMs required much less time than N-DTSVMs for training.

iii L-DTSVMs required a bit more time than L-gSVMs for testing, while N-DTSVMs needed less time than N-gSVMs for testing.

iv On "Mediamill", the non-linear classifiers achieved much higher micro test scores than linear classifiers. This makes N-DTSVMs leading performers on this data set, since they performed comparably to, but required much shorter training time than, N-gSVMs.

v On the remaining data sets, L-DTSVMs were the leading performers, since they performed comparably to, but required much less training time than, the non-linear classifiers. Moreover, L-DTSVMs outperformed L-gSVMs.

vi When applying L-DTSVMs to "RCV1-Full," we had 12 L-DTSVMs built on the full set of training data and several L-DTSVMs built on much smaller leaves.

vii When applied to "RCV1-Full," L-DTSVMs achieved higher average micro scores than L-gSVMs on all the collections of samples that possessed a certain number of labels. Moreover, the performance gain of L-DTSVMs over L-gSVMs was more than 4% for the collection of samples that possessed 7 or 8 labels.

4. Conclusion

We have shown that DTSVMs can provide an efficient and effective solution for multi-label classification problems. The efficiency and effectiveness of DTSVMs derive from two factors: building lSVMs on regions decomposed by decision trees, and searching for the appropriate size for decomposed regions. As a consequence, N-DTSVMs achieved a comparable performance to that of N-gSVMs, but required much less time for training. In addition, L-DTSVMs often achieved better performances than L-gSVMs. Our experimental studies also revealed that, on the four data sets derived from textual applications, L-DTSVMs were the leading performers, since they performed comparably to, but required much less training time than, non-linear classifiers. On the "Mediamill" data set derived from a multimedia application, N-DTSVMs were the leading performers, since N-DTSVMs performed comparably to, but required much shorter training time than, N-gSVMs, and N-DTSVMs outperformed linear classifiers.
Figure 6: The micro scores obtained by gSVMs and DTSVMs.

Figure 7: The training times required by gSVMs and DTSVMs when the micro score is used as the performance measure.
Figure 8: The testing times required by gSVMs and DTSMVs when the micro score is used as the performance measure.

![Testing Time (Sec.)](image)

<table>
<thead>
<tr>
<th>Method</th>
<th>Trnc2007</th>
<th>Mellam2008</th>
<th>RCV1-100k</th>
<th>RCV1-250k</th>
<th>RCV1-Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-gSVMs</td>
<td>0.09</td>
<td>0.87</td>
<td>2.19</td>
<td>4.67</td>
<td>12.18</td>
</tr>
<tr>
<td>L-DTSMVs</td>
<td>0.20</td>
<td>1.38</td>
<td>2.77</td>
<td>10.49</td>
<td>59.48</td>
</tr>
<tr>
<td>N-gSVMs</td>
<td>498.06</td>
<td>2,180.48</td>
<td>4,212.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N-DTSMVs</td>
<td>544.78</td>
<td>2,079.87</td>
<td>3,545.07</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 9: Ceiling size variation of L-DTSMVs on "RCV1-Full" when the micro score is used as the performance measure.

![Ceiling Size Variation (RCV1-Full)](image)

<table>
<thead>
<tr>
<th>Number of L-DTSMVs</th>
<th>1,200</th>
<th>2,000</th>
<th>4,000</th>
<th>8,000</th>
<th>16,000</th>
<th>32,000</th>
<th>64,000</th>
<th>128,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>28</td>
<td>50</td>
<td>28</td>
<td>10</td>
<td>6</td>
<td>13</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 10: The average micro scores achieved by L-gSVMs and L-DTSMVs on collections of samples with a certain number of labels.

![Micro Score vs. Label Variation (RCV1-Full)](image)

<table>
<thead>
<tr>
<th>Label Variation</th>
<th>1*2</th>
<th>5*4</th>
<th>5*5</th>
<th>5*6</th>
<th>7*8</th>
<th>9*10</th>
<th>11*12</th>
<th>13*14</th>
<th>15*16</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-gSVMs</td>
<td>85.3%</td>
<td>79.8%</td>
<td>68.8%</td>
<td>65.7%</td>
<td>63.4%</td>
<td>60.1%</td>
<td>47.9%</td>
<td>22.2%</td>
<td></td>
</tr>
<tr>
<td>L-DTSMVs</td>
<td>86.6%</td>
<td>81.6%</td>
<td>72.0%</td>
<td>69.9%</td>
<td>67.8%</td>
<td>62.3%</td>
<td>49.9%</td>
<td>22.2%</td>
<td></td>
</tr>
</tbody>
</table>
References


B. Hariharan, L. Zelnik-Manor, S. V. N. Vishwanathan, and M. Varma. Large scale max-
margin multi-label classification with priors. In Proceedings of the International Confer-
ence on Machine Learning, 2010.


F. Chang, C.-Y. Guo, X.-R. Lin, and C.-J. Lu. Tree Decomposition for Large-Scale SVM 

A. Clare and R. D. King. Knowledge discovery in multi-label phenotype data. In Pro-
cedings of the 5th European Conference on Principles of Data Mining and Knowledge 

A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. Advances in 

E. L. Mencía and J. Fürnkranz. Efficient pairwise multilabel classification for large-scale 
Learning and Knowledge Discovery in Databases, Pages 50-65, 2008.


S. Godbole and S. Sarawagi. Discriminative methods for multi-labeled classification. In 
Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 

T. Goncalves and P. Quaresma. A preliminary approach to the multilabel classification 
problem of Portuguese juridical documents. In Proceedings of the 11th Portuguese Con-

T. Joachims. Text categorization with support vector machines: learning with many rele-
vant features. In Proceedings of ECML-98, 10th European Conference on Machine Learn-
ing, Pages 137-142, 1998.

B. Lauser and A. Hotho. Automatic multi-label subject indexing in a multilingual en-
vironment. In Proceedings of the 7th European Conference in Research and Advanced 

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text 


