We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Chapter 8

Decision Tree as an Accelerator for Support Vector Machines

Fu Chang and Chan-Cheng Liu

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/52227

1. Introduction

Support vector machine (SVM) is known to be a very powerful learning machine for pattern classification, of which optical character recognition (OCR) naturally falls as a branch. There are, however, a few hindrances in making an immediate application of SVM for the OCR purpose. First, to construct a well-performing SVM character recognizer has to deal with a large set of training samples (hundreds of thousands in the Chinese OCR, for example). There are two types of SVMs: linear and non-linear SVMs. Training a linear SVM is relatively inexpensive, while training a non-linear SVM is of the order n^p, where n is the number of training samples and $p \geq 2$. Thus, the sheer size of samples has the potential of incurring a high training cost on an OCR application. Second, a normal OCR task also deals with a large number of class types. There are, for example, thousands of character class types being handled in the Chinese OCR. There are also hundreds of them being handled in the English OCR, if touched English letters are considered as separate class types from untouched letters. Since SVM training deals with one pair of class types at a time, we need to train $l(l\text{-}1)/2$ one-against-one (1A1) classifiers (Kerr et al. [1]) or l one-against-others (1AO) classifiers (Bottou et al. [2]), where l is the number of class types. Such a gigantic collection of classifiers not only poses a problem to the training but also to the testing of SVMs. Third, SVM training also involves a number of parameters whose values affect the generalization power of classifiers. This means that searching for optimal parameter values is necessary and it constitutes another heavy load to the SVM training. The above three factors, when put together, will demand months of computing time to complete a whole round of conventional SVM trainings, including linear and non-linear SVM trainings, and also demands an unusual amount of time in conducting a conventional online OCR task.

To cope with the above problems, we propose two methods, both of which involve the use of decision tree (Breiman et al. [3]) to speed up the computation. The first method, called

decision tree support vector machine (DTSVM) (Chang et al. [4]) is developed by us to expedite SVM training. The second method, called *random forest decomposition* (RFD), generalizes a technique of ours (Liu et al. [5]) to speed up a testing process.

DTSVM decomposes a given data space with a decision tree. It then trains SVMs on each of the decomposed regions. In so doing, DTSVM can enjoy the following advantages. First, training non-linear SVMs on decomposed regions of size σ reduces the complexity from n^p to $(n/\sigma)\times\sigma^p = n\sigma^{p-1}$. Second, the decision tree may decompose the data space so that certain decomposed regions become homogeneous (i.e., they contain samples of the same class type), thereby reducing the cost of SVM training that is applied only to the remaining samples. Since DTSVM trains SVMs on regions of size σ, it leaves σ an additional parameter to the parameters $\boldsymbol{\theta}$ associated with SVMs. The third advantage of DTSVM then lies in the fact that DTSVM handles all values of $\boldsymbol{\theta}$ only on the regions of lowest σ-size, and focus on very few selected values of $\boldsymbol{\theta}$ on the regions of higher σ-sizes, thereby making further savings in the training cost.

While DTSVM speeds up SVM training, it may not help reduce the time consumed in SVM testing. To achieve the latter goal, we propose to use *multiple* trees to decompose the data space. In this method, each tree employs a subset of randomly drawn features, instead of the set of all features. The collection of these trees is called a *random forest*. The RFD method proposed by us differs from the traditional random forest method (Ho [6], Breiman [7]) in the following way. The traditional method determines the class type for each test sample $\mathbf{x}$, while RFD determines a number of class types for $\mathbf{x}$. RFD is thus a learning algorithm whose objective is to reduce the number of class types for each test sample. There are a few parameters whose values need to be determined in the RFD's learning process, including the number of trees, the common size of each tree's decomposed regions, and one more parameter to be described in Section 3. The values of these parameters will be determined under the constraint that they lead to a restricted classifier whose generalization power is not inferior to the un-restricted classifier. The generalization power of a classifier can be estimated as the accuracy rate obtained in a validation process. The RFD thus assumes that a classifier is constructed in advance. In our case, it is the DTSVM classifier.

DTSVM is very handy for constructing classifiers and for conducting other tasks, including the selection of linear or non-linear SVMs, the selection of critical features, etc. RFD, on the other hand, is handy for putting a classifier to use in an online process. The results reported in this chapter showed that DTSVM and RFD could substantially speed up SVM training and testing, respectively, and still achieved comparable test accuracy. The reason for the no loss of test accuracy is the following. DTSVM and RFD trainings, similar to SVM training, involve a search for optimal parameters, thus bringing about the best possible classifiers as an outcome.

In this chapter, we apply DTSVM and RFD methods to three data sets of very large scale: ETL8B (comprised of 152,960 samples and 956 class types), ETL9B (comprised of 607,200 samples and 3,036 class types), and ACP (comprised of 548,508 samples and 919 class types). The features to be extracted from ETL8B and ETL9B are those described in Chou et al. [8] and Chang et al. [9]; those extracted from ACP are described in Lin et al. [10].

On the three data sets, we conducted our experiments in the following manner. We first trained linear and non-linear DTSVMs on the experimental data sets with some reasonable parameter values. We then computed DTSVMs' performance scores, including training time, test speed, and test accuracy rates. Although a strict comparison between DTSVMs and global SVMs (gSVMs, i.e., SVMs that are trained on the full training data set) may not be possible, due to the extremely slow training process of gSVMs, it is possible to estimate the speedup factors achieved by DTSVMs. To show the effectiveness of DTSVM, we further compare DTSVM classifiers with the classifiers obtained by *k*-nearest neighbor (kNN) and decision tree (without the addition of SVMs) methods.

The rest of this chapter is organized as follows. Section 2 reviews the DTSVM method. In Section 3, we describe the RFD learning algorithm. In Section 4, we describe our experimental results. Section 5 contains some concluding remarks.

2. Decision Tree Support Vector Machine (DTSVM)

In this section, we describe DTSVM as a method to speed up the training of SVMs over large-scale data sets. We divide the section into two sub-sections. In the first sub-section, we describe the decision tree that is used in DTSVM, and in RFD as well, for decomposing the data space. In the second sub-section, we describe the learning algorithm of DTSVM.

An implementation of DTSVM is available at

http://ocrwks11.iis.sinica.edu.tw/~dar/Download/WebPages/DTSVM.htm,

which contains source codes, experimental data sets, and an instruction file to use the codes.

2.1 The Decision Tree as a Decomposition Scheme

The decomposition scheme adopted in the DTSVM and RFD methods is basically a CART (Breiman et al., 1984) or a binary C4.5 (Quinlan, 1986) that allows two child nodes to grow from each node that is not a leaf. To grow a binary tree, we follow a recursive process, whereby each training sample flowing to a node is sent to its left-hand or right-hand child node. At a given node E, a certain feature f of the training samples flowing to E is compared with a certain value v so that all samples with $f < v$ are sent to the left-hand child node, and the remaining samples are sent to the right-hand child node. The optimal values of (f, v) are determined as follows:

$$(f^*, v^*) = \arg\max_{(f,v)} IR(f, v),$$

where

$$IR(f, v) = I(S) - \frac{|S_{f<v}|}{|S|} I(S_{f<v}) - \frac{|S_{f\geq v}|}{|S|} I(S_{f\geq v}),$$

S is the set of all samples flowing to E; $S_{f<v}$ consists of the elements of S with $f < v$; $S_{f \geq v} = S \setminus S_{f<v}$; $| X |$ is the size of any data set X; and $I(X)$ is the impurity of X. The impurity function used in our experiments is the entropy measure, defined as

$$I(S) = -\sum_y p(S_y) \log p(S_y),$$

where $p(S_y)$ is the proportion of S's samples whose class type is y.

For both DTSVM and RFD, we do not grow a decision tree to its full scale. Instead, we stop splitting a node E when one of the following conditions is satisfied: (i) the number of samples that flow to E is smaller than a *ceiling size* σ; or (ii) when $IR(f, v) = 0$ for all f and v at E. The value of σ in the first condition is determined in a data-driven fashion, which we describe in Section 2.2. The second condition occurs mainly in the following cases. (a) All the samples that flow to E are homogeneous; or (b) a subset of them is homogeneous and the remaining samples, although differ in class type, are identical to some members of the homogeneous subset.

2.2. The DTSVM learning algorithm

After growing a tree, we train an SVM on each of its leaves, using samples that flow to each leaf as training data. A tree and all SVMs associated with its leaves constitute a DTSVM classifier. In the training phase, all the SVMs in a DTSVM classifier are trained with the same parameter values. In the validation or the testing phase, we input a given data point $\mathbf{x}$ to the tree. If $\mathbf{x}$ reaches a homogeneous leaf, we classify $\mathbf{x}$ as the common class type of those samples; otherwise, we classify it with the SVM associated with that leaf.

When a learning data set is given, we divide a given learning data set into a training and validation constituent. We then build a DTSVM classifier on the training constituent and determine its optimal parameter values with the help of the validation constituent. The parameters associated with a DTSVM classifier are: (i) σ, the ceiling size of the decision tree; and (ii) the SVM parameters. Their optimal values are determined in the following manner.

We begin by training a binary tree with an initial ceiling size σ_0, and then train SVMs on the leaves with SVM-parameters $\boldsymbol{\theta} \in \Theta$, where Θ is the set of all possible SVM-parameter values whose effects we want to evaluate. Note that we express $\boldsymbol{\theta}$ in boldface to indicate that it may consist of more than one parameter. Let $var(\sigma_0, \boldsymbol{\theta})$ be the validation accuracy rate achieved by the resultant DTSVM classifier.

Next, we construct DTSVM classifiers with larger ceiling sizes $\sigma_1, \sigma_2, \ldots$, with $\sigma_0 < \sigma_1 < \sigma_2 < \ldots$ On the leaves of these trees, we only train their associated SVMs with k top-ranked $\boldsymbol{\theta}$. To do this, we rank $\boldsymbol{\theta}$ in descending order of $var(\sigma_0, \boldsymbol{\theta})$. Let Θ_k be the set that consists of k top-ranked $\boldsymbol{\theta}$. We then implement the following sub-process, denoted as $SubProcess(\boldsymbol{\theta})$, for each $\boldsymbol{\theta} \in \Theta_k$.

1. Set $t = 0$ and get the binary tree with the ceiling size σ_0.

2. Increase t by 1. Modify the tree with ceiling size σ_{t-1} to obtain a tree with ceiling size σ_t. This is done by moving from the root towards the leaves and retaining each node whose size or whose parent's size is greater than σ_t. Then, train SVMs on the leaves with SVM-parameters $\boldsymbol{\theta}$. Let $var(\sigma_t, \boldsymbol{\theta})$ be the validation accuracy of the resultant DTSVM classifier.
3. If $var(\sigma_t, \boldsymbol{\theta}) - var(\sigma_{t-1}, \boldsymbol{\theta}) \geq 0.5\%$ and σ_t is less than the size of the training constituent, proceed to step 2.
4. If $var(\sigma_t, \boldsymbol{\theta}) - var(\sigma_{t-1}, \boldsymbol{\theta}) < 0.5\%$, then $\sigma(\boldsymbol{\theta}) = \sigma_{t-1}$; otherwise, $\sigma(\boldsymbol{\theta}) = \sigma_t$.

When we have completed $SubProcess(\boldsymbol{\theta})$ for all $\boldsymbol{\theta} \in \Theta_{[k]}$, we define

$$\boldsymbol{\theta}_{opt} = \arg\max_{\boldsymbol{\theta} \in \Theta_k} var(\sigma(\boldsymbol{\theta}), \boldsymbol{\theta}) \text{ and } \sigma_{opt} = \sigma(\boldsymbol{\theta}_{opt}).$$

We then output the DTSVM classifier with the SVM-parameter $\boldsymbol{\theta}_{opt}$ and the ceiling size σ_{opt}.

In our experiments, training linear SVMs involved only one parameter C, the cost penalty factor, whose values were taken from $\Phi = \{10^a: a = -1, 0, \ldots, 5\}$. Training non-linear SVMs involved two parameters, C and γ, where γ appears in an RBF kernel function. The values of C were also from Φ, while the values of γ were taken from $\Psi = \{10^b: b = -1, -2, \ldots, -9\}$. Furthermore, we fixed the number of top-ranked parameters at $k = 3$ for linear SVMs and at $k = 5$ for non-linear SVMs. For the sequence of ceiling sizes, we had only two such numbers: the initial ceiling size $\sigma_0 = 1{,}500$ and the next ceiling size $\sigma_1 = n+1$, where n is the number of training samples. The reason for these two numbers is as follows. On a tree with the initial ceiling size σ_0, we needed to train SVMs with all combinations of parameter values. So, we set σ_0 at a sufficiently low level to save a tremendous amount of training time. At the next stage, we immediately jumped to the root level of a tree, because in the three experimental data sets, the number of training samples per class type was not high, even though the total number of training samples was very large, so we did not want to waste time on any intermediate level between σ_0 and σ_1.

3. Random Forest Decomposition (RFD)

In this section, we address the acceleration of SVM testing, using RFD as the method to construct a multiple decomposition scheme. The implementation of the RFD method is available at the following website.

http://ocrwks11.iis.sinica.edu.tw/~dar/Download/WebPages/RFD.htm

To speed up SVM testing, we assume that all the required SVM classifiers have been constructed. Suppose that there are l class types and we want to conduct 1A1 classification, then there are $l(l-1)/2$ SVMs in total. To classify a data point $\mathbf{x}$, we first apply our multiple decomposition scheme to pull out m candidate class types for $\mathbf{x}$, where m depends on $\mathbf{x}$ and $m < l$. We then apply $m(m-1)/2$ SVMs to $\mathbf{x}$, each of which involves a pair of candidate class types. If, on the other hand, we want to conduct 1AO classification, then there are l SVMs and we apply m of them to $\mathbf{x}$.

In the above process, we use random forest (RF) as the multiple decomposition scheme. An RF is a collection of trees, each of which is trained on a separate subset of features that is drawn randomly from the set of all features. When the total number of features is F, we train all such trees on a subset of $[F/2]$ features, where $[F/2]$ is the integral part of $F/2$. Moreover, we train all these trees with a common ceiling size. At each leaf of an RF, we store the class types of the training samples that flow to this leaf, instead of the training samples themselves.

When an RF is given, let τ = the number of trees in the RF and σ = the common ceiling size of these trees. For a given data point $\mathbf{x}$, we first send $\mathbf{x}$ to all the τ trees and examine the leaves to which $\mathbf{x}$ flows. Next, we pull out the class types that are stored in at least μ leaves. We then classify $\mathbf{x}$ under the restriction that only these class types are considered as the candidate class types of $\mathbf{x}$.

The RFE training process thus involves the construction of an RF of τ trees with a common ceiling size σ, which we denote as RF(τ, σ). For each data point $\mathbf{x}$, let $M(\mathbf{x}, \tau, \sigma, \mu)$ be the collection of class types such that

$$M(\mathbf{x},\tau,\sigma,\mu) = \{l : \ l \text{ is stored in at least } \mu \text{ leaves of RF}(\tau,\sigma) \text{ to which } \mathbf{x} \text{ flows}\}.$$

To find the optimal values of τ, σ, and μ, we divide a given learning data set into a training constituent and a validation constituent. For each possible value of τ and σ, we train a random forest RF(τ, σ) on the training constituent. Then, for each possible value of μ, we compute the validation accuracy rate $var(\tau, \sigma, \mu)$ on the validation constituent, where

$$var(\tau,\sigma,\mu) = \frac{|\{\mathbf{x} \in \mathrm{V} : \text{ the SVMs restricted to } M(\mathbf{x},\tau,\sigma,\mu) \text{ correctly classifies } \mathbf{x}\}|}{|\mathrm{V}|},$$

V is the set of all validation samples, and $|\mathrm{X}|$ is the size of any data set X.

To make an exhaustive search for the highest possible value of $var(\tau, \sigma, \mu)$ proves to be very time consuming. So we propose the following two-stage search strategy. At the first stage, we fix $\mu = 1$ and search for sufficiently low τ^* and σ^* such that $var(\tau^*, \sigma^*, 1) \geq var_{baseline}$, where $var_{baseline}$ is the validation accuracy rate achieved by the unrestricted SVMs. At the second stage, we look for the largest μ^* such that $var(\tau^*, \sigma^*, \mu^*) \geq var_{baseline}$.

We fix $\mu = 1$ at the first stage based on the following observation. For any value of values of $\mathbf{x}$, τ, and σ, we have $M(\mathbf{x}, \tau, \sigma, 1) \supseteq M(\mathbf{x}, \tau, \sigma, 2) \supseteq \ldots$, and $var(\tau, \sigma, 1) \geq var(\tau, \sigma, 2) \geq \ldots$ So, if $var(\tau, \sigma, \mu) \geq var_{baseline}$ for some μ, we must have $var(\tau, \sigma, 1) \geq var_{baseline}$.

The first stage of our search strategy is detailed as follows.

1. Set $\tau = 15$ and $\sigma = 500$, namely, grow 15 trees with a common ceiling size 500.
2. If $var(\tau, \sigma, 1) \geq var_{baseline}$, stop the process. Otherwise, change the common ceiling size of the τ trees from σ to $4\times\sigma$.
3. If $var(\tau, \sigma, 1) \geq var_{baseline}$, stop the process. Otherwise, increase τ by 5; namely, grow 5 more trees with a common ceiling size σ.

4. Go to step 2.

The procedure must stop at a finite number of iteration. In the worst case, it stops when σ reaches the root level and all class types are candidate class types. The resultant τ and σ in this procedure are denoted as τ^* and σ^*. At the next stage, we look for

$$\mu^* = \arg\max_{1 \le \mu \le \tau^*} \{\mu : var(\tau^*, \sigma^*, \mu) \ge var_{baseline}\}$$

under the constraint that $\tau = \tau^*$, $\sigma = \sigma^*$, and $var(\tau^*, \sigma^*, \mu^*) \ge var_{baseline}$.

4. Experimental results

In this section, we describe the data sets in the experiments and the features extracted out the character images. We then present and discuss the experimental results.

4.1. The data sets and features

To demonstrate the effects of the proposed methods, we applied DTSVM and RFD to the data sets: ACP, ETL8B, and ETL9B. The ACP data set derived from a task of classifying textual components on machine-printed documents written in Chinese and English. In the original work described in [10], all textual components were classified into 3 types: alphanumeric (A), Chinese (C), and punctuation (P). Once classified, those components would be sent to a separate recognizer for further classification. In the current task, we expanded the 3 class types into 863 alphanumeric types and 55 punctuation types. So, there would be no separate recognizers for alphanumeric or punctuation types. Moreover, there were a lot more class types in the alphanumeric category than one might expect, because we considered touched characters as separate types from all other types. On the other hand, all Chinese components were considered as one type, since we had to first merge them into characters and send the characters into a Chinese recognizer.

When textual component was segmented from a document image, we extract the following features out of it.

Density. A 64×64 bitmap image is divided into 8×8 regions, each comprising 64 pixels. For each region, the counts of black pixels are used as a density feature. The total number of features in the density category is 64.

Cross Count. A cross count is the average number of black intervals that lie within eight consecutive scan lines that run through a bitmap in either a horizontal or vertical direction. The total number of features in the cross-count category is 16.

Aspect Ratio. For a textual component *TC* that appears in a horizontal textline *H*, we obtain the following features: 1) bit '1' for the slot indicating that *H* is a horizontal textline; 2) '0' for the slot indicating that *H* is a vertical textline; 3) the ratio between *TC*'s height and *H*'s height; 4) the ratio between *TC*'s height and *TC*'s width; 5) the ratio between *TC*'s top gap

and *H*'s height; and 6) the ratio between *TC*'s bottom gap and *H*'s height. We follow the same procedure for a textual component that appears in a vertical textline. The total number of features in the aspect-ratio category is 6.

ETL8B and ETL9B are well known data sets comprising 955 and 3,035 Chinese/Hiragana handwritten characters respectively. For all the characters contained in the two data sets, we used a feature extraction method consisting of the following basic techniques: non-linear normalization (Lee and Park [11], Yamada et al. [12]), directional feature extraction (Chou et al. [8]), and feature blurring (Liu et al. [13]). These three techniques were considered as major breakthroughs in handwritten Chinese/Hiragana character recognition (Umeda [14]). The total number of features extracted out of each character is 256.

The feature vectors extracted out of the three data sets can be found at the following website.

http://ocrwks11.iis.sinica.edu.tw/~dar/Download/WebPages/RFD.htm

When conducting both training and testing, we decompose each data set into training, validation, and test constituents at the ratio of 4:1:1. We use samples in the training constituent to train classifiers. We then use samples in the validation constituent for finding optimal parameters. Finally, we apply the classifiers trained with optimal parameters to the test constituent for computing the test accuracy rate. Table 1 contains detailed information for all the data sets and the constituents derived from them.

	ACP	ETL8B	ETL9B
Number of Class Types	919	955	3,035
Number of Features	86	256	256
Number of Samples	545,698	152,960	607,200
Training Constituent	363,789	102,292	406,824
Validation Constituent	90,944	25,812	100,188
Test Constituent	90,965	24,856	100,188

Table 1. The three data sets used in our experiments.

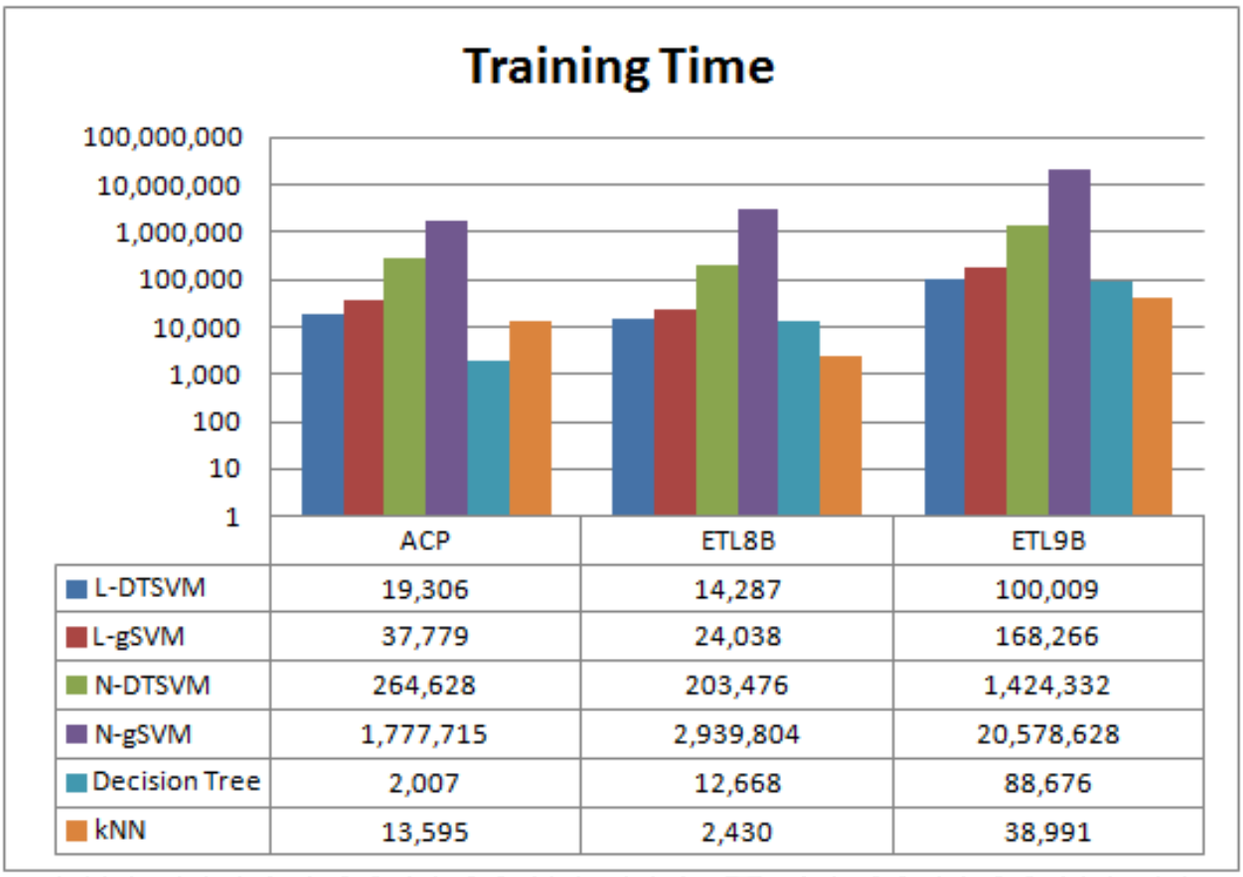
4.2. Results of DTSVM

Two types of DTSVM were studied in our experiments. They were linear DTSVM (L-DTSVM) and non-linear DTSVM (N-DTSVM). We compared them with linear gSVM (L-gSVM) and non-linear gSVM (N-gSVM). In addition to these four SVM methods, we also included decision tree and kNN for comparison. To train SVMs for L-DTSVM and L-gSVM, we employed LIBLINEAR (Fan et al. [15]). On the other hand, we used LIBSVM (Fan et al. [16]) to train SVMs for N-DTSVM and N-gSVM.

For all the SVM methods, we only conducted 1A1 classification. While 1AO is another option to take, it is too costly compared to 1A1. A 1AO-training involves samples of all class types, while a 1A1-training involves samples of only two class types. In the 1A1 training, we

needed to train $l(l\text{-}1)/2$ SVMs. In the testing, however, we performed a DAG testing process (Platt et al. [17]) that involved only l SVMs for classifying a test sample. More about DAG will be given in Section 4.3.

We display in Figure 1 the training times of the six compared methods, expressed in *seconds*. The results demonstrated that DTSVM conducted training substantially faster than gSVM. The speedup factor of L-DTSVM relative to L-gSVM was between 1.6 and 2.0, while the speedup factor of N-DTSVM relative to N-gSVM was between 6.7 and 14.4. The results also showed that the non-linear SVM methods were a lot more time-consuming than linear SVM methods. On the other hand, decision tree and kNN are fast in training.



	ACP	ETL8B	ETL9B
L-DTSVM	19,306	14,287	100,009
L-gSVM	37,779	24,038	168,266
N-DTSVM	264,628	203,476	1,424,332
N-gSVM	1,777,715	2,939,804	20,578,628
Decision Tree	2,007	12,668	88,676
kNN	13,595	2,430	38,991

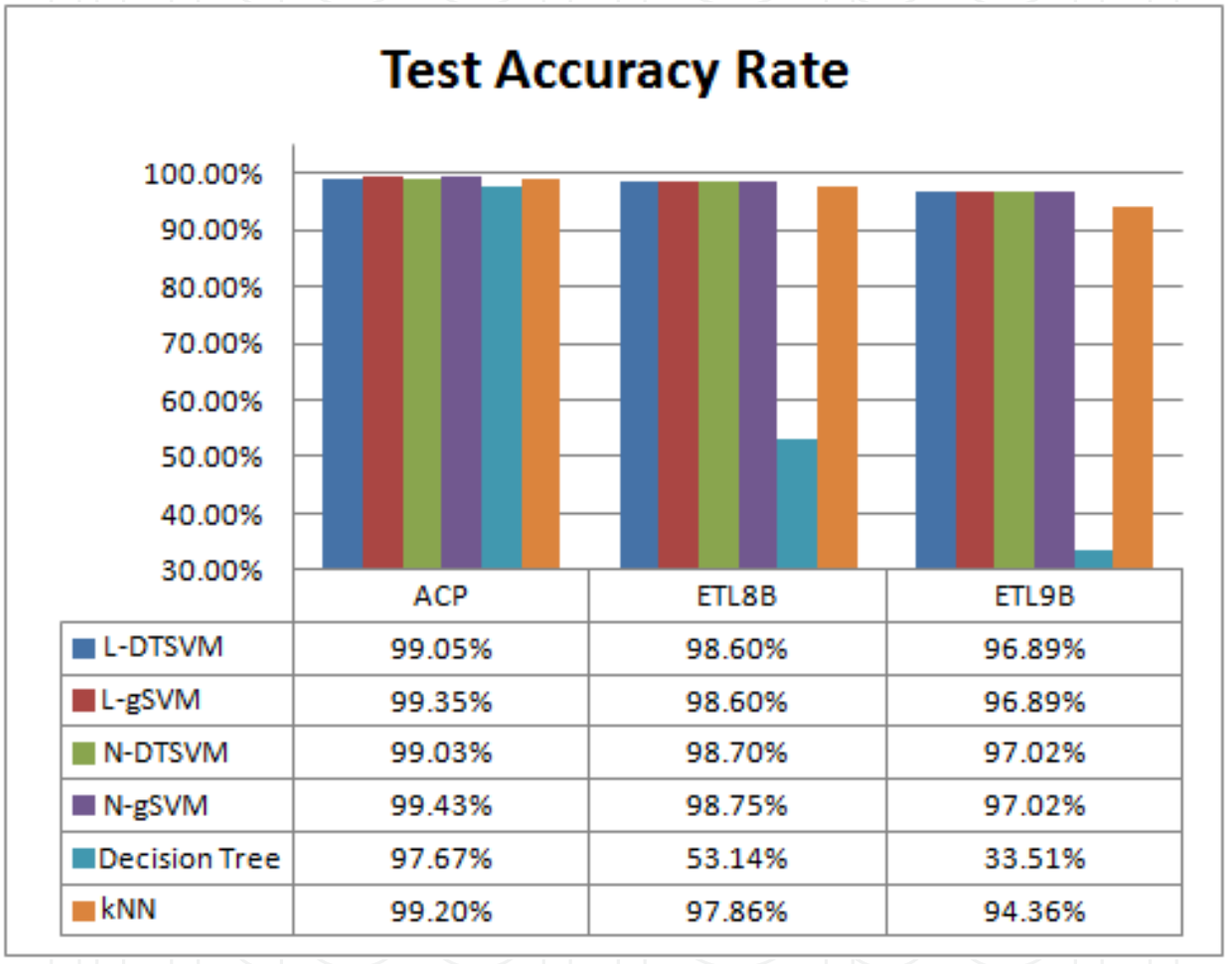
Figure 1. Training times of the six compared methods, expressed in seconds. DTSVMs outperformed gSVMs and decision tree outperformed all the six methods.

Figure 2 shows the test accuracy rates achieved by all the compared methods. All the SVM methods achieved about the same rates. Moreover, they outperformed decision tree and kNN on all the data sets. Decision tree, in particular, performed poorly on data sets ETL8B and ETL9B; kNN fell behind the SVM methods by a visible amount on ETL9B.

Figure 3 shows the test speeds of all the compared methods, expressed in *characters per second*. L-DTSVM achieved a staggering high speed on the data set ACP. The two linear SVM methods conducted testing much faster than the two non-linear SVM methods.

Decision tree, again, was faster in testing; kNN was slow, unsurprisingly, because it had to compare a test sample against all training samples.

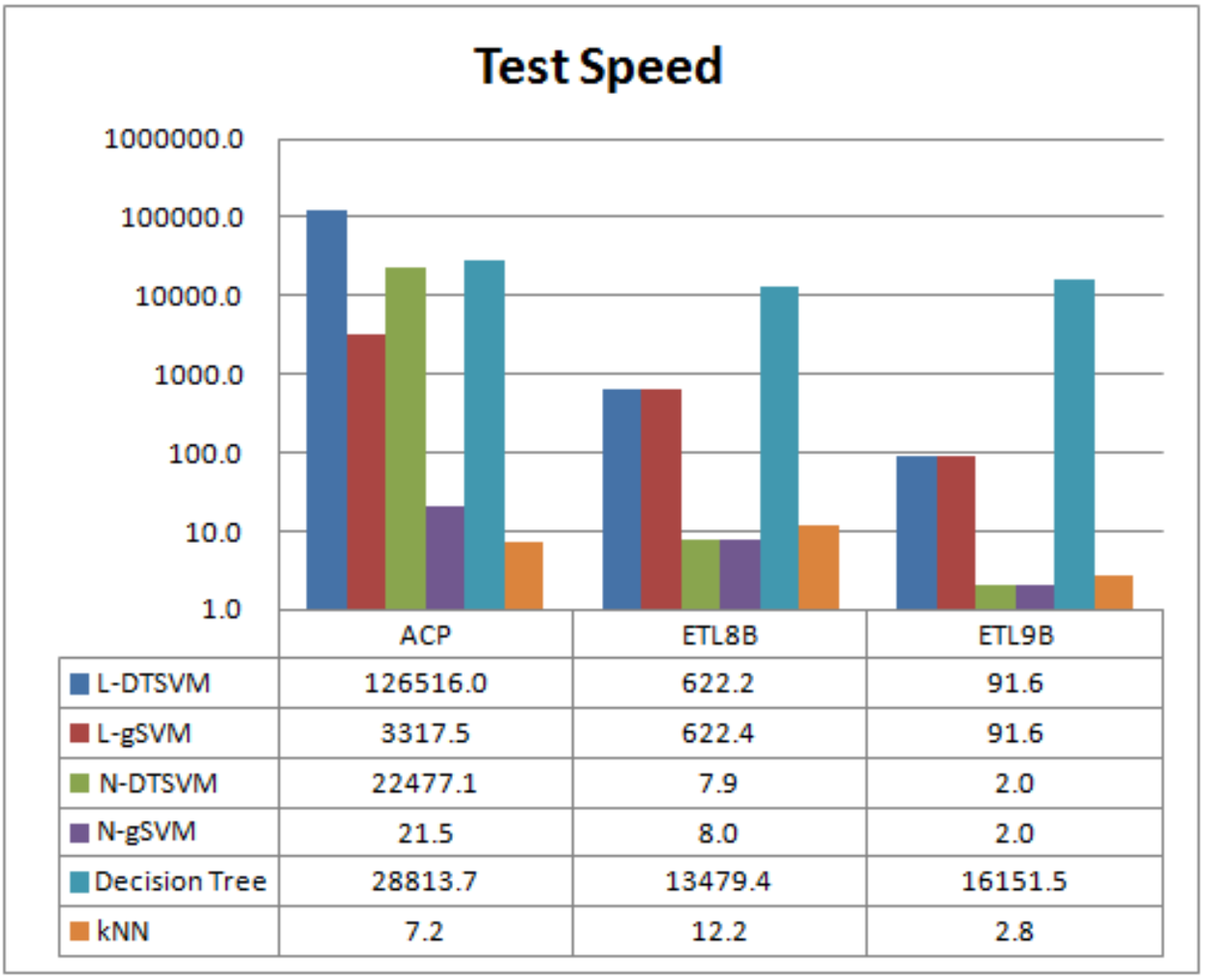
All the times or speeds reported in this chapter were measured on Quad-Core Intel Xeon E5335 CPU 2.0GHz with a 32GB RAM. In our experiments, we took advantage of parallelism to shorten the wall-clock time. However, all the times reported here are CPU times. Furthermore, we were able to train all gSVMs on ACP and ETL8B, but we did not complete the training of gSVMs on ETL9B. Instead, we estimated the total training time based on the SVM training that we had performed for DTSVMs on the root level.



	ACP	ETL8B	ETL9B
L-DTSVM	99.05%	98.60%	96.89%
L-gSVM	99.35%	98.60%	96.89%
N-DTSVM	99.03%	98.70%	97.02%
N-gSVM	99.43%	98.75%	97.02%
Decision Tree	97.67%	53.14%	33.51%
kNN	99.20%	97.86%	94.36%

Figure 2. Test accuracy rates of the six compared methods. SVMs achieved comparable accuracy rates to each other; they outperformed decision tree and kNN.

We also show in Table 2 the optimal parameter values for all compared methods, except the decision tree that involves no parameters. On the data set ACP, $\sigma^* = 1{,}500$, explaining why L-DTSVM and N-DTSVM conducted training and testing at such a high speed. On ETL8B and ETL9B, σ^* = root, implying that DTSVM classifiers were trained on the root, the same site where gSVM classifiers were trained. This explains why DTSVM and gSVM conducted testing at the same speed. However, DTSVMs consumed less time in training than gSVMs because not all local SVMs of the DTSVM classifiers were trained on the root level.



	ACP	ETL8B	ETL9B
L-DTSVM	126516.0	622.2	91.6
L-gSVM	3317.5	622.4	91.6
N-DTSVM	22477.1	7.9	2.0
N-gSVM	21.5	8.0	2.0
Decision Tree	28813.7	13479.4	16151.5
kNN	7.2	12.2	2.8

Figure 3. Test speeds of the six compared methods, expressed in characters per second. L-DTSVM outperformed all other SVM methods; decision tree outperformed all other methods, except L-DTSVM on the data set ACP.

Finally, we remark that, on the ACP data set, DTSVM training not only settled at a low ceiling size (1,500) but also resulted in a tree with some homogeneous leaves. In fact, 63.2% of the ACP training samples flowed to leaves with a single class type, the Chinese type. So in the testing phase, a large proportion of ACP test samples also flowed to homogeneous leaves, leaving no further effort for classifying them. The ETL8B and ETL9B data sets, on the other hand, comprised a large number of small-sized class types and no large-sized class type. So the DTSVM training settled at the root level on the two data sets.

4.3. Results of RFD

The RFD method is associated with an SVM training method. When applying RFD, we must have all the associated SVM classifiers constructed.

If, for example, gSVM is the training method, RFD will work with all the 1A1 classifiers (l, l'), where l and l' are any two class types. We first describe how we use these classifiers in

the DAG testing process. When a test sample **x** is given, we first tag all class types as *likely* types. We next apply a classifier (l_1, l_2) to **x**. If **x** is classified as l_1, we re-tag l_2 as unlikely and replace it by a likely type l_3. We then apply the classifier (l_1, l_3) to **x**. This process goes on until only one likely type is left, which we take as **x**'s class type.

Data Set	Method	σ^*	C^*	γ^*	k^*
ACP	L-DTSVM	1,500	1		
	L-gSVM		1		
	N-DTSVM	1,500	10^2	10^{-1}	
	N-gSVM		10	10^{-1}	
	kNN				3
ETL8B	L-DTSVM	root	10^5		
	L-gSVM		10^5		
	N-DTSVM	root	10^4	10^{-8}	
	N-gSVM		10^5	10^{-9}	
	kNN				10
ETL9B	L-DTSVM	root	10		
	L-gSVM		10		
	N-DTSVM	root	10^3	10^{-7}	
	N-gSVM		10^3	10^{-7}	
	kNN				12

Table 2. Optimal parameter values for all the methods except decision tree. Empty cells imply that the corresponding categories are not applicable.

When the RFD method is employed, we first send **x** to the corresponding RF and find the m candidate class types for **x**. We tag these class types as likely types and the remaining class types as unlikely types. We then proceed as in the DAG process until only one likely type is left.

If, on the other hand, RFD works with a DTSVM classifier, **x**'s candidate class types must fall into two subsets: one is associated with the decision tree of the DTSVM classifier and the other is with the RF derived by the RFD method. So we extract the class types from the intersection of these two subsets and tag them as the likely types. We then proceed as in the DAG process.

We show in Tables 3 and 4 the results of applying the RFD method to L-DTSVM, L-gSVM, N-DTSVM and N-gSVM classifiers. Table 3 displays the times to train the corresponding RFs and the optimal parameters associated with them. It is shown that all the RFs comprise

15 decision trees and almost all of them settled at the ceiling size 500, except the RFs for accelerating DTSVMs on the ACP data set settling at the ceiling size 2,000.

Data Set	Method	Training Time	τ^*	σ^*	μ^*
ACP	L-DTSVM	381,645	15	2,000	3
	L-gSVM	380,617	15	500	4
	N-DTSVM	381,676	15	2,000	4
	N-gSVM	381,450	15	500	9
ETL8B	L-DTSVM	190,232	15	500	3
	L-gSVM	190,232	15	500	3
	N-DTSVM	202,241	15	500	2
	N-gSVM	202,241	15	500	2
ETL9B	L-DTSVM	3,263,999	15	500	1
	L-gSVM	3,263,999	15	500	1
	N-DTSVM	3,490,394	15	500	1
	N-gSVM	3,490,394	15	500	1

Table 3. Training times and optimal parameters for the RFs associated with all the SVM methods.

Table 4 displays the testing times achieved by all the SVM methods with or without the RFD to speed up. The effects of RFD were manifest on all SVM classifiers and all data sets, except for the DTSVM classifier on the ACP data set. The reason for the exceptional case is easy to understand. DTSVM classifiers ran very fast on the ACP data set; to speed it up by another device (i.e., an RF) would not be economical, due to the fact that this device would incur its own computing cost to the process.

4.4. Summary

We summarize the results in Sections 4.2 and 4.3 as follows.

1. Among all the competing methods, we judge L-DTSVM to be the champion, since it achieved comparable test accuracy rates to all other SVM methods, required the least times to train and to test among all SVM methods, and outperformed decision tree the kNN by large. This is a rather welcomed result, since L-DTSVM conducted much faster training and testing than other SVM methods.
2. The decision tree and kNN, although were fast in training, achieved worse test accuracy rates than the SVM methods. Moreover, the kNN method was slow for testing. We thus found these two methods unsuitable for our purpose.
3. The DTSVM method proved to be very effective for speeding up SVM training and achieved comparable test accuracy rates to gSVM. This was even true when linear SVM was adopted as the learning machine.

Data Set	Classifier	With RFD	Without RFD
ACP	L-DTSVM	29249.2	126516.0
	L-gSVM	21565.9	3317.5
	N-DTSVM	16354.7	22477.1
	N-gSVM	819.5	21.5
ETL8B	L-DTSVM	2766.7	622.2
	L-gSVM	2766.7	622.4
	N-DTSVM	30.7	7.9
	N-gSVM	30.7	8.0
ETL9B	L-DTSVM	519.1	91.6
	L-gSVM	519.1	91.6
	N-DTSVM	6.6	2.0
	N-gSVM	6.6	2.0

Table 4. Testing times achieved by all the SVM methods with or without RFD to speed up.

4. The RFD method proved to be very effective to speed up SVM testing. This claim was found to be true for all but one case, in which the DTSVM method was already very fast to require any further acceleration.

5. Conclusion

Having applied the DTSVM and RFD methods to three data sets comprising machine-printed and handwritten characters, we showed that we were able to substantially reduce the time in training and testing SVMs, and still achieved comparable test accuracy. One pleasant result obtained in the experiments was that linear DTSVM classifiers performed the best among all SVM methods, in the sense that they attained better or comparable test accuracy rates and consumed the least amount of time in training and testing.

Author details

Fu Chang* and Chan-Cheng Liu
Institute of Information Science, Academia Sinica, Taipei, Taiwan

6. References

[1] Knerr S, Personnaz L, Dreyfus G (1990) Single-layer Learning Revisited: A Stepwise Procedure for Building and Training A Neural Network. In J. Fogelman,

* Corresponding Author

editor, Neurocomputing: Algorithms, Architectures and Applications. Springer-Verlag.

[2] Bottou L, Cortes C, Denker J, Drucker H, Guyon I, Jackel L, LeCun Y, Müller U, Sackinger E, Simard P, Vapnik V (1994) Comparison of Classifier Methods: A Case Study in Handwriting Digit Recognition. Int. Conf. on Pattern Recognition; pp. 77–87.

[3] Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and Regression Trees. Chapman and Hall.

[4] Chang F, Guo CY, Lin XR, Lu CJ (2010) Tree Decomposition for Large-Scale SVM Problems. Journal of Machine Learning Research; 11: 2935–2972.

[5] Liu YH, Lin CC, Lin WH, Chang F (2007) Accelerating Feature-Vector Matching Using Multiple-Tree and Sub-Vector Methods. Pattern Recognition; 40(9): 2392-2399.

[6] Ho TK (1998) The Random Subspace Method for Constructing Decision Forests. IEEE Transactions on Pattern Analysis and Machine Intelligence; 20(8): 832–844.

[7] Breiman L (2006) Random Forests. Machine Learning. 45(1): 5-32.

[8] Chou CH, Lin CC, Liu YH, Chang F (2006) A Prototype Classification Method and Its Use in A Hybrid Solution for Multiclass Pattern Recognition. Pattern Recognition; 39(4): 624-634.

[9] Chang F (2008) Techniques for Solving The Large-Scale Classification Problem in Chinese Handwriting Recognition. Arabic and Chinese Handwriting Recognition, Lecture Notes in Computer Science; 4768: 161-169.

[10] Lin XR, Guo CY, Chang F (2011) Classifying Textual Components of Bilingual Documents with Decision-Tree Support Vector Machines. Int. Conf. on Document Analysis and Recognition; PP. 498-502.

[11] Lee SW, Park JS (1994) Nonlinear Shape Normalization Methods for The Recognition of Large-Set Handwritten Characters. Pattern Recognition; 27(7): 895-902.

[12] Yamada H, Yamamoto K, Saito T (1990) A Nonlinear Normalization Method for Handprinted Kanji Character Recognition – Line Density Equalization. Pattern Recognition 23(9): 1023-1029.

[13] Liu CL, Kim IJ, Kim JH (1997) High Accuracy Handwritten Chinese Character Recognition by Improved Feature Matching Method. Int. Conf. Document Analysis and Recognition; pp. 1033-1037.

[14] Umeda M (1996) Advances in Recognition Methods for Handwritten Kanji Characters. IEICE Trans. Information and Systems; E79-D(5): 401-410.

[15] Fan RE, Chang KW, Hsieh CJ, Wang XR, Lin CJ (2008) LIBLINEAR: A Library for Large Linear Classification. Journal of Machine Learning Research; 9: 1871-1874.

[16] Fan RE, Chen PH, Lin CJ (2005) Working Set Selection Using Second Order Information for Training SVM. Journal of Machine Learning Research; 6: 1889-1918.

[17] Platt JC, Cristianini N, Shawe-Taylor J. (2000) Large Margin DAGs for Multiclass Classification. In S. A. Solla, T. K. Leen and K.-R. Müller, editors, Advances in Neural Information Processing Systems. MIT Press.

IntechOpen