Alcom-FT Technical Report Series
ALCOMFT-TR-01-92

Random sampling techniques for training support vector machines*

José L. Balcázar

Departament de Llenguatges i Sistemes Informatics,
Univ. Politecnica de Catalunya
Campus Nord, Jordi Girona Salgado 1-3, 08034 Barcelona, Spain

Yang Dai and Osamu Watanabe

Dept. of Mathematical and Computing Sciences,
Tokyo Institute of Technology
Meguro-ku Ookayama, Tokyo 152-8552, Japan

May 21, 2001

Abstract

Random sampling techniques have been developed in for combinatorial optimization problems. In this note, we report two applications of one of these techniques for training support vector machines that solve two-group classification problems by using hyperplane classifiers. Through this research, we are aiming (i) to design efficient and theoretically guaranteed support vector machine training algorithms, and (ii) to develop systematic and efficient methods for finding "outliners", i.e., examples having an inherent error.

1. Introduction

This paper proposes new training algorithms of *support vector machines* (SVMs in short) for two-group classification problems. We use one of the random sampling techniques that have been developed and used for combinatorial optimization problems; see, e.g., [Cla88, AS93, GW00]. Through this research, we are aiming (i) to design efficient and theoretically guaranteed support vector machine training algorithms, and (ii) to develop systematic and efficient methods for finding "outliers", i.e., examples having an inherent error. Our two proposed algorithms, though not perfect, are a good step towards the first goal (i). We show that our algorithms terminate within a reasonable number of training steps. Unfortunately, though, the first algorithm may not work if there are many errors, and for the second algorithm, we need some hypothesis for

*The first author is supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT), EU EP27150 (Neurocolt II), Spanish Government PB98-0937-C04 (FRESCO), and CIRIT 1997SGR-00366. The third author is supported by a Grant-in-Aid for Scientific Research on Priority Areas "Discovery Science" from the Ministry of Education, Science, Sports and Culture of Japan.

proving its efficiency. For the second goal (ii), we propose, though only briefly, some approach based on these random sampling techniques.

Since the present form of SVM was proposed [CV95], SVMs have been used in various application areas, and their classification power has been investigated in depth from both experimental and theoretical points of view. Also many algorithms and implementation techniques have been developed for training SVMs efficiently; see, e.g., [SS98, BMS00]. This is because quadratic programming (QP in short) problems need to be solved for training SVMs (as in the original form) and such a QP problem is, though polynomial-time solvable, not so easy. Among speed-up techniques, those called "subset selection" [SS98] have been used as effective heuristics from the early stage of the SVM research. Roughly speaking, a *subset selection* is a technique to speed-up SVM training by dividing the original QP problem into small pieces, thereby reducing the size of each QP problem. Well known subset selection techniques are chunking, decomposition, and sequential minimal optimization (SMO in short). (See [CV95, Pla99, CST00] for the detail.) In particular, SMO has become popular because it outperforms the others in several experiments. Though the performance of these subset selection techniques has been extensively examined, no theoretical guarantee has been given on the efficiency of algorithms based on these techniques. (As far as the authors know, the only positive theoretical results are the convergence (i.e., termination) of some of such algorithms [OFG97, Lin01, KG01].)

In this paper, we propose two subset selection type algorithms based on a randomized sampling technique developed in the combinatorial optimization community. Both of them solve the SVM training problem by solving iteratively small QP problems for randomly chosen examples. Unfortunately, neither of them is perfect. But we think that they are good starting points towards efficient and theoretically guaranteed algorithms.

The first algorithm indeed is a simple/direct application of the random sampling technique. We can prove some reasonable[1] upper bound on the number of iterations, i.e., the number of small QP problems to be solved. (Unfortunately, though, the algorithm could become slower than the naive one for data with many errors.) On the other hand, the second algorithm can accomplish each iteration within reasonable amount time, and it halts within reasonable amount of iterations. (Unfortunately, though, we need some hypothesis to obatain an upper bound for the number of iterations.)

In order to simplify our discussion, we focus on training SVMs as a primal-form; that is, we discuss with the problem of obtaining a maximal-margin hyperplane classifier. It is, however, possible to extend the first algorithm to work in a dual-form, where we can use the kernel technique.

2. Support Vector Machine and Random Sampling Algorithm

Here we explain basic notions on SVM and random sampling techniques. Due to the space limit, we only explain those necessary for our discussion. For SVM, see, e.g., a good textbook [CST00], and for random sampling techniques, see, e.g., an excellent survey [GW00].

For support vector machine formulations, we will consider, in this paper, only the binary classification by a hyperplane of the example space; in other words, we regard training SVM for

[1]By "reasonable bound", we mean some low polynomial bound w.r.t. n, m, and ℓ, where n, m, and ℓ are respectively the number of attributes, the number of examples, and the number of errorneous examples.

a given set of labeled examples as the problem of computing a hyperplane separating positive and negative examples with the largest margin.

Suppose that we are given a set of m examples $\boldsymbol{x}_i$, $1 \leq i \leq m$, in some n dimension space, say $\mathbb{R}^n$. Each example $\boldsymbol{x}_i$ is labeled by $y_i \in \{1, -1\}$ denoting the classification of the example. The *SVM training problem* we will discuss in this paper is essentially to solve the following optimization problem. (Here we follow [BB00] and use their formulation. But the above problem can be restated by using a single threshold parameter as given in the original paper [CV95]. Note also that this is the separable case. The nonseparable case, which is our main topic, will be discussed in the next section.)

$$\begin{array}{ll} \underline{\text{Max Margin (P1)}} & \\ \text{min.} & \frac{1}{2}\|\boldsymbol{w}\|^2 - (\theta_+ - \theta_-) \quad \text{w.r.t. } \boldsymbol{w} = (w_1, ..., w_n),\ \theta_+, \text{ and } \theta_-, \\ \text{s.t.} & \boldsymbol{w} \cdot \boldsymbol{x}_i \ \geq \ \theta_+ \quad \text{if } y_i = 1, \quad \text{and} \quad \boldsymbol{w} \cdot \boldsymbol{x}_i \ \leq \ \theta_- \quad \text{if } y_i = -1. \end{array}$$

(*Remarks on Notations.* Throughout this note, we use X to denote the set of examples, and let n and m denote the dimension of the example space and the number of examples respectively. Also we use i for indexing examples (and their labels), and $\boldsymbol{x}_i$ and y_i to denote the ith example and its label respectively. The range of i is always $\{1, ..., m\}$.)

By the *solution* of (P1), we mean the hyperplane that achieves the minimum cost. We sometimes consider a partial problem of (P1) that minimizes a target cost under some subset of constrains. A solution to such a partial problem of (P1) is called a *local solution* of (P1) for the subset of constraints.

We can solve this optimization problem by using a standard general quadratic programming algorithm. In most applications, however, the number m of examples is much larger than the dimension n (in other words, much more constraints than variables). This is the situation where randomized sampling techniques are effective.

We first explain intuitively our[2] random sampling algorithm for solving the problem (P1). The idea is simple. Pick up a certain number of examples from X and solve (P1) under the set of constraints corresponding to these examples. We choose examples randomly according to their "weights", where initially all examples are given the same weight. Clearly, the obtained local solution is, in general, not the global solution, and it does not satisfy some constraints; in other words, some examples are misclassified by the local solution. Then double the "weight" of such misclassified examples, and then pick up some examples again randomly according to their weights. If we iterate this process several rounds, the weight of "important examples", which are support vectors in our case, would get increased, and hence, they are likely to be chosen. Note that once all support vectors are chosen at some round, then the local solution of this round is the real one, and the algorithm terminates at this point. By using the *Sampling Lemma*, we can prove that the algorithm terminates in $O(n \log m)$ rounds on average. We will give this bound after explaining necessary notions and notations and stating our algorithm.

Randomized sampling techniques, particularly, the Sampling Lemma, is applicable for many "LP-type" problems. Here we use $(\mathcal{D}, \phi)$ to denote an abstract LP-type problem, where $\mathcal{D}$ is a set of elements and ϕ is a function mapping any $\mathcal{R} \subseteq \mathcal{D}$ to some value space. In the case of

[2] This algorithm is not new at all. It is obtained from a more general algorithm given in [GW00].

procedure OptMargin
  set weight $u(\boldsymbol{x}_i)$ to be 1 for all examples in X;
  $r \leftarrow 6\delta^2$;   % $\delta = n+1$.
  repeat
    $R \leftarrow$ choose r examples from X randomly according to u;
    $(\boldsymbol{w}^*, \theta_+^*, \theta_-^*)$ is a solution of (P1) for R;
    $V \leftarrow$ the set of violators in X of the solution;
    if $u(V) \le u(X)/(3\delta)$ **then** double the weight $u(\boldsymbol{x}_i)$ for all $\boldsymbol{x}_i \in V$;
  until $V = \emptyset$;
  return the last solution;
end-procedure.

Figure 1: Randomized SVM Training Algorithm

our problem (P1), for example, we can regard $\mathcal{D}$ as X and define ϕ as a mapping from a given subset R of X to the local solution of (P1) for the subset of constraints corresponding to R. As a LP-type problem, we require $(\mathcal{D}, \phi)$ to satisfy certain conditions. Here we omit the explanation and simply mention that our example case clearly satisfies these conditions.

For any $\mathcal{R} \subseteq \mathcal{D}$, a *basis* of $\mathcal{R}$ is an inclusion-minimal subset $\mathcal{B}$ of $\mathcal{R}$ such that $\phi(\mathcal{B}) = \phi(\mathcal{R})$. The *combinatorial dimension* of $(\mathcal{D}, \phi)$ is the size of the largest basis of $\mathcal{D}$. We will use δ to denote the combinatorial dimension. For the problem (P1), the largest basis is the set of all support vectors; hence, the combinatorial dimension of (P1) is $n+1$. Consider any subset $\mathcal{R}$ of $\mathcal{D}$. A *violator* of $\mathcal{R}$ is an element e of $\mathcal{D}$ such that $\phi(\mathcal{R} \cup \{e\}) \neq \phi(\mathcal{R})$. An element e of $\mathcal{R}$ is *extreme* (or, simply called an *extremer*) if $\phi(\mathcal{R} - \{e\}) \neq \phi(\mathcal{R})$. Consider our case. For any subset R of X, let $(\boldsymbol{w}^*, \theta_+^*, \theta_-^*)$ be a local solution of (P1) obtained for R. Then $\boldsymbol{x}_i \in X$ is a *violator* of R (or, more directly, a *violator* of $(\boldsymbol{w}^*, \theta_+^*, \theta_-^*)$) if the constraint corresponding to $\boldsymbol{x}_i$ is not satisfied with $(\boldsymbol{w}^*, \theta_+^*, \theta_-^*)$.

Now we state our algorithm as Figure 1. In the algorithm, we use u to denote a weight scheme that assigns some integer weight $u(\boldsymbol{x}_i)$ to each $\boldsymbol{x}_i \in X$. For this weight scheme u, consider a multiple set U containing each example $\boldsymbol{x}_i$ exactly $u(\boldsymbol{x}_i)$ times. Note that U has $u(X)$ $(= \sum_i u(\boldsymbol{x}_i))$ elements. Then by "choose r examples randomly from X according to u", we mean to select a set of examples randomly from all $\binom{u(X)}{r}$ subsets of U with equal probability.

For analyzying the efficiency of this algorithm, we use the Sampling Lemma that is stated as follows.

Lemma 2.1. Let $(\mathcal{D}, \phi)$ be any LP-type problem. Assume some weight scheme u on $\mathcal{D}$ that gives an integer weight to each element of $\mathcal{D}$. Let $u(\mathcal{D})$ denote the total weight. For a given r, $0 \le r < u(\mathcal{D})$, we consider the situation where r elements of $\mathcal{D}$ are chosen randomly according to their weights. Let $\mathcal{R}$ denote the set of chosen elements, and let $v_{\mathcal{R}}$ be the weight of violators of $\mathcal{R}$. Then we have the following bound.

$$\mathrm{Exp}(v_{\mathcal{R}}) \ \le \ \frac{u(\mathcal{D}) - r}{r+1} \cdot \delta. \tag{1}$$

Using this lemma, we can prove the following bound.

Theorem 2.2. The average number of iterations executed in the OptMargin algorithm is bounded by $6\delta \ln m = O(n \ln m)$. (Recall $|X| = m$ and $\delta = n+1$.)

Thus, while our randomized OptMargin algorithm needs to solve (P1) for about $6n \ln m$ times on average, the number of constraints needed to consider at each time is about $6n^2$. Hence, if n is much smaller than m, then this algorithm is faster than solving (P1) directly. For example, the fastest quadratic programming solver up to date needs *roughly* $O(mn^2)$ time. Hence, if n is smaller than $m^{1/3}$, then we can get (at least asymptotic) speed-up. (Of course, one does not have to use such a general purpose solver, but even for an algorithm designed specifically for solving (P1), it is better if the number of constrains is smaller.)

3. The Nonseparable Case with Outliers and the First Algorithm

For the separable case, the randomized sampling approach seems to help us by reducing the size of the optimization problem we need to solve for training SVM. On the other hand, the important feature of SVM is that it is also applicable for the nonseparable case. More precisely speaking, the nonseparable case includes two subcases: (i) the case where the hyperplane classifier is too weak for classifying given examples, and (ii) the case where there are some erroneous examples, namely outliers. The first subcase is solved by the SVM approach by mapping examples into a much higher dimension space. The second subcase is solved by relaxing constraints by introducing slack variables or "soft margin error". In this paper, we will discuss a way to handle the second subcase; that is, the nonseparable case with outliers.

First we generalize the problem (P1) and state the soft margin hyperplane separation problem.

Max Soft Margin (P2)

$$\begin{array}{ll} \text{min.} & \frac{1}{2}\|\boldsymbol{w}\|^2 - (\theta_+ - \theta_-) + D \cdot \sum_i \xi_i, \quad \text{w.r.t. } \boldsymbol{w} = (w_1, ..., w_n),\ \theta_+,\ \theta_-, \text{ and } \xi_1, ..., \xi_m \\ \text{s.t.} & \boldsymbol{w} \cdot \boldsymbol{x}_i \ \geq\ \theta_+ - \xi_i \text{ if } y_i = 1, \quad \boldsymbol{w} \cdot \boldsymbol{x}_i \ \leq\ \theta_- + \xi_i \text{ if } y_i = -1, \quad \text{and} \quad \xi_i \ \geq\ 0. \end{array}$$

Here $D < 1$ is a parameter that determines the degree of influence from outliers. Note that D should be fixed in advance; that is, D is a constant throughout the training process. (Note that a more generalized SVM formulation, one can change D and furthermore use different D for each example. We left such a generalization for our future work.)

At this point, we can formally define the notion of outliers we are considering in this paper. For a given set X of examples, suppose we solve the problem (P2) and obtain the optimal hyperplane. Then an example in X is called an *outlier* if it is misclassified with this hyperplane. Throughout this paper, we use ℓ to denote the number of outliers. Notice that this definition of outlier is quite relative; that is, relative to the hypothesis class and relative to the soft margin parameter D.

The problem (P2) is again a quadradic programming with linear constraints; thus, it is possible to use our random sampling technique. More specifically, by choosing δ appropriately, we can use the algorithm of Figure 1 here. This is our first algorithm solving the SVM training problem with errorneous examples.

Here we should be a bit careful for determining the parameter δ, the combinatorial dimension. Although we have increased the number of variables by introducing $\xi_1, ..., \xi_m$, the number of examples that determine the solution hyperplane remains $n+1$ as before. In our words, the combinatorial dimension of (P2) seems still $n+1$. But we should be a bit careful for arguing this way. In order to see (P1) as a LP-type problem, we gave some interpretation to $\mathcal{D}$ and ϕ. That is, we use X for $\mathcal{D}$, and define ϕ as a mapping from a given subset R of X to the local "solution" of (P1) for R. For the problem (P2), we can also use similar interpretaton; but we should be careful about the meaning of "solution". If we regard "solution" as the obtained optimal hyperplane and define ϕ accordingly, then (P2) does not satisfy some of the conditions for LP-type problems. What we have to use for the "solution" is the value that is minimized in the problem (P2). In this case, any example that increases this value is a violator; hence, an example is a violator if and only if it is misclassified the currently obtained hyperplane. From this observation, we can argue that the combinatorial dimension of (P2) is $n+\ell+1$. That is, the combinatorial dimension is increased by the number ℓ of outliers.

Now from the argument in the previous section, our algorithm samples roughly $O((n+\ell)^2)$ examples at each round, and terminates in $O((n+\ell)\ln m)$ rounds on average. Therefore, if ℓ is large, this algorithm does not work well. On the other hand, if ℓ is small, then we can still use this algorithm to reduce the total computation time. In fact, in some situation, we may be able to remove "trivial" errors by some precomputation, thereby reducing the number of errorneous examples. Then this algorithm may be appropriate.

4. A Geometrical View

Usually optimization problems can be viewed in several different ways, which can help us to introduce new algorithms. Here we introduce new algorithms by reformulating the problem (P2) in a different way. We will make use of an intuitive geometric interpretation to (P2) that has been given by Bennett and Bredensteiner [BB00].

Bennett and Bredensteiner [BB00] proved that (P2) is equivalent to the following problem (P3); more precisely, (P3) is the Wolfe dual of (P2).

<u>Reduced Convex Hull (P3)</u>

$$\begin{array}{lll} \text{min.} & \dfrac{1}{2}\left\|\sum_i y_i s_i \boldsymbol{x}_i\right\|^2 & \text{w.r.t. } s_1, \ldots, s_m \\ \text{s.t.} & \sum_{i:y_i=1} s_i = 1, & \sum_{i:y_i=-1} s_i = 1, \\ & 0 \le s_i \le D. & \end{array}$$

<u>Convex Hull of Composed Examples (P4)</u>

$$\begin{array}{lll} \text{min.} & \dfrac{1}{2}\left\|\sum_I y_I s_I \boldsymbol{z}_I\right\|^2 & \text{w.r.t. } s_1, \ldots, s_M \\ \text{s.t.} & \sum_{I:y_I=1} s_I = 1, & \sum_{I:y_I=-1} s_I = 1, \\ & 0 \le s_I \le 1. & \end{array}$$

Note that $||\sum_i y_i s_i \boldsymbol{x}_i||^2 = ||\sum_{i:y_i=1} s_i \boldsymbol{x}_i - \sum_{i:y_i=-1} s_i \boldsymbol{x}_i||^2$. That is, the value minimized in (P3) is the distance between two points in the convex hull of positive and negative examples. In the separable case, it is the distance between two closest points in two convex hulls. On the other hand, in the nonseparable case, we give some restriction to the influence of each example; each example cannot contribute to the closest point more than D.

As mentioned in [BB00], the meaning of D is intuitively explained by considering its inverse $k = 1/D$. (Here we assume that $1/D$ is an integer. Throughout this note, we use k to denote this constant.) Instead of the original convex hulls, we consider the convex hulls of points composed from k examples. Then resulting convex hulls are reduced ones and they may be separable by some hyperplane; in the extreme case where $k = m_+$ (where m_+ is the number of positive examples), the reduced convex hull for positive examples consists of only one point.

More formally, we can reformulate (P3) as follows. Let Z be the set of *composed examples* $\boldsymbol{z}_I$ that is defined by $\boldsymbol{z}_I = (\boldsymbol{x}_{i_1} + \boldsymbol{x}_{i_2} + \cdots + \boldsymbol{x}_{i_k})/k$, with some k distinct elements $\boldsymbol{x}_{i_1}$, $\boldsymbol{x}_{i_2}$, ..., $\boldsymbol{x}_{i_k}$ of X with the same label (i.e., $y_{i_1} = y_{i_2} = \cdots = y_{i_k}$). The label y_I of the composed example $\boldsymbol{z}_I$ inherits its members'. Throughout this note, we use I for indexing elements of Z and their labels. The range of I is $\{1, ..., M\}$, where $M \stackrel{\text{def}}{=} |Z|$. Note that $M \le \binom{m}{k}$. For each $\boldsymbol{z}_I$, we use z_I to denote the set of original examples from which $\boldsymbol{z}_I$ is composed. Then (P3) is equivalent to the problem (P4) stated above.

Finally we consider the Wolfe primal of this problem. Then we came back to our favorite formulation!

Max Margin for Composed Examples (P5)

$$
\begin{array}{ll}
\text{min.} & \frac{1}{2}\|\boldsymbol{w}\|^2 - (\eta_+ - \eta_-) \quad \text{w.r.t. } \boldsymbol{w} = (w_1, ..., w_n),\ \eta_+, \text{ and } \eta_-, \\
\text{s.t.} & \boldsymbol{w} \cdot \boldsymbol{z}_I \ \ge\ \eta_+ \ \text{ if } y_I = 1, \ \text{ and } \ \boldsymbol{w} \cdot \boldsymbol{z}_I \ \le\ \eta_- \ \text{ if } y_I = -1.
\end{array}
$$

Note that the combinatorial dimension of (P5) is $n+1$, the same as that of (P1). The difference is that we have now $M = O(m^k)$ constraints, which is quite large. But this situation is suitable for the sampling technique. Suppose that we use our algorithm based on the randomized sampling technique (OptMargin of Figure 1) for solving (P5). Since the combinatorial dimension is the same, we can use $r = 6(n+1)^2$ as before. On the other hand, from our analysis, the expected number of iterations is $O(n \ln M) = O(kn \ln m)$. That is, we need to solve QP problems with $n+2$ variables and $O(n^2)$ constraints for $O(kn \ln m)$ times, which is not bad at all!

Unfortunately, however, there are serious problems. The algorithm needs, at least as it is, a large amount of time and space for "book keeping" computation. First of all, we have to keep weights of all M composed examples in Z. Secondly, for finding violators and for modifying weights, we have to go through Z, which takes at least $O(M)$ steps. Also it is not so easy to choose composed examples randomly according to their weights.

5. A Modified Random Sampling Algorithm

As we have seen in Section 4, we cannot simply use the algorithm OptMargin for (P5). It takes too much time and space to maintain the weight of all composed examples and to generate them according to their weights. Here we propose a way to get around this problem by giving weight to original examples; this is our second algorithm.

Before stating our algorithm and its analysis, let us first examine solutions to the problems (P2) and (P5). For a given example set X, let Z be the set of composed examples. Let $(\boldsymbol{w}^*, \theta_+^*, \theta_-^*)$ and $(\boldsymbol{w}^*, \eta_+^*, \eta_-^*)$ be the solutions of (P2) for X and (P5) for Z respectively. Note that two solutions share the same $\boldsymbol{w}^*$; this is because (P2) and (P5) are essentially equivalent problems [BB00]. Let $X_{\text{err},+}$ and $X_{\text{err},-}$ denote the sets of positive/negative outliers. That is, $\boldsymbol{x}_i$ belongs to $X_{\text{err},+}$ (resp., $X_{\text{err},-}$) if and only if $y_i = 1$ and $\boldsymbol{w}^* \cdot \boldsymbol{x}_i < \theta_+^*$ (resp., $y_i = -1$ and $\boldsymbol{w}^* \cdot \boldsymbol{x}_i > \theta_-^*$). We use ℓ_+ and ℓ_- to denote the number of positive/negative outliers. Recall that we are assuming that our constant k is larger than both ℓ_+ and ℓ_-. Let $X_{\text{err}} = X_{\text{err},+} \cup X_{\text{err},-}$.

The problem (P5) is regarded as the LP-type problem $(\mathcal{D}, \phi)$, where the correspondence is the same as (P1) except that Z is used as $\mathcal{D}$ here. Let Z_0 be the basis of Z. (In order to simplify our discussion, we assume nondegeneracy throughout the following discussion.) Note that every element of the basis is extreme in Z. Hence, we call elements of Z_0 *final extremers*.

By definition, the solution of (P5) for Z is defined by the constraints corresponding to these final extremers.

By analyzing the Karush-Kuhn-Tucker (in short, KKT) condition for (P2), we can show the following facts. (Though the lemma is stated only for the positive case, i.e., the case $y_I = 1$, the corresponding properties hold for the negative case $y_I = -1$.)

Lemma 5.1. Let $\boldsymbol{z}_I$ be any positive final extremer, i.e., an element of Z_0 such that $y_I = 1$. Then the following properties hold: (a) $\boldsymbol{w}^* \cdot \boldsymbol{z}_I = \eta_+^*$. (b) $X_{\mathrm{err},+} \subseteq z_I$. (c) For every $\boldsymbol{x}_i \in z_I$, if $\boldsymbol{x}_i \notin X_{\mathrm{err},+}$, then we have $\boldsymbol{w}^* \cdot \boldsymbol{x}_i = \theta_+^*$. Thus, when the final extremers are obtained, the set of outliers can be obtained as $\cap_{\boldsymbol{z}_I \in Z_0} z_I$, the set of examples appearing in all final extremers.

Let us give some intuitive interpretation to the facts given in this lemma. (Again we only consider, for the simplicity, the positive examples.) The fact (a) of the lemma shows that all final extremers are located on some hyperplane whose distance from the base hyperplane $\boldsymbol{w}^* \cdot \boldsymbol{z} = 0$ is η_+^*. On the other hand, the fact (c) states that all original *normal* examples (i.e., examples not in $X_{\mathrm{err},+}$) are located again on some hyperplane whose distance from the base hyperplane is $\theta_+^* > \eta_+^*$. Here consider the point $\boldsymbol{v}_+ \stackrel{\mathrm{def}}{=} \left(\sum_{\boldsymbol{x}_i \in X_{\mathrm{err},+}} \boldsymbol{x}_i \right) / \ell_+$, i.e., the center of positive outliers, and define $\mu_+^* = \boldsymbol{w}^* \cdot \boldsymbol{v}_+$. Then we have $\theta_+^* \geq \eta_+^* \geq \mu_+^*$; that is, the hyperplane defined by the final extremers is located between the one having all normal examples in the final extremers and the one having the center $\boldsymbol{v}_+$ of outliers. More specifically, since every final extremer is composed from all ℓ_+ positive outliers and $k - \ell_+$ normal examples, we have

$$\theta_+^* - \eta_+^* \;:\; \eta_+^* - \mu_+^* \;=\; k - \ell_+ \;:\; \ell_+.$$

Next we consider local solutions of (P5). We would like to solve (P5) by using the random sampling technique. That is, choose some small subset R of Z randomly according to current weight, and solve (P5) for R. Thus, let us examine local solutions obtained by solving (P5) for such a subset R of Z.

For any set R of composed examples in Z, let $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$ be the solution of (P5) for R. Similar to the above, we consider $\widetilde{Z}_0$ to be the set of extremers of R w.r.t. the solution $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$. On the other hand, we define here $\widetilde{X}$ to be the set of original examples appearing in some extremers in $\widetilde{Z}_0$.

As before, we will discuss about only positive composed/original examples. Let $\widetilde{Z}_{0,+}$ be the set of positive extremers. Different from the case where all composed examples are examined to solve (P5), here we cannot expect, for example, that all extremers in $\widetilde{Z}_{0,+}$ share the same set of misclassified examples. Thus, instead of sets like $X_{\mathrm{err},+}$, we consider a subset $\widetilde{X}'_+$ of the following set $\widetilde{X}_+$. (It may be the case that $\widetilde{X}_+$ is empty.)

$$\widetilde{X}_+ \;=\; \text{the set of positive examples appearing in all extremers in } \widetilde{Z}_{0,+}.$$

Intuitively, we want to discuss by using the set $\widetilde{X}'_+$ of "misclassified" examples appearing in all positive extremers. But such a set cannot be defined at this point because no threshold corresponding to θ_+^* has been given. Thus, for a while, let us consider any subset $\widetilde{X}'_+$ of $\widetilde{X}_+$. Let $\widetilde{\ell}'_+ = |\widetilde{X}'_+|$, and define $\boldsymbol{v}_+ = \left(\sum_{\boldsymbol{x}_i \in \widetilde{X}'_+} \boldsymbol{x}_i \right) / \widetilde{\ell}'_+$. Also for each $\boldsymbol{z}_I \in \widetilde{Z}_0$, we define a point $\boldsymbol{v}_I$ that is the center of all original examples in $z_I - \widetilde{X}'_+$. That is,

$$\boldsymbol{v}_I \stackrel{\mathrm{def}}{=} \frac{\sum_{\boldsymbol{x}_i \in z_I - \widetilde{X}'_+} \boldsymbol{x}_i}{k - \widetilde{\ell}'_+}.$$

Then we can prove the following fact that corresponds to Lemma 5.1 and that is proved similarly.

Lemma 5.2. For any subset R of Z, we use the symbols defined as above. There exists some $\widetilde{\theta}'_+$ such that for any extremer $\boldsymbol{z}_I$ in $\widetilde{Z}_{0,+}$, we have $\widetilde{\boldsymbol{w}} \cdot \boldsymbol{v}_I = \widetilde{\theta}'_+$.

Now for our $\widetilde{X}_{\mathrm{err},+}$, we use a subset $\widetilde{X}'_+$ of $\widetilde{X}_+$ satisfying the following for any $\boldsymbol{x}_i \in \widetilde{X}_+$.

$$\boldsymbol{x}_i \in \widetilde{X}'_+ \;\Leftrightarrow\; \widetilde{\boldsymbol{w}} \cdot \boldsymbol{x}_i \;<\; \widetilde{\theta}'_+,$$

where $\widetilde{\theta}'_+$, which we denote $\widetilde{\theta}_{\mathrm{err},+}$, is the threshold given in Lemma 5.2 for $\widetilde{X}'_+$. Such a set (while it could be empty) is well-defined. (In the case that $\widetilde{X}_{\mathrm{err},+}$ is empty, we define $\widetilde{\theta}_{\mathrm{err},+} = \widetilde{\eta}_+$.)

For any original positive example $\boldsymbol{x}_i \in X$, we call it a *missed example* (w.r.t. the local solution $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$) if $\boldsymbol{x}_i \notin \widetilde{X}_{\mathrm{err},+}$ and it holds that

$$\widetilde{\boldsymbol{w}} \cdot \boldsymbol{x}_i \;<\; \widetilde{\theta}_{\mathrm{err},+}. \tag{2}$$

We will use such a missed example as an evidence that there exists a "violator" to $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$, which is guaranteed by the following lemma.

Lemma 5.3. For any subset R of Z, let $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$ be the solution of (P5) for R. Then if there exists a missed example w.r.t. $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$, then we have some composed example in Z that is misclassified w.r.t. $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$. On the other hand, for any composed example $\boldsymbol{z}_I \in Z$, if it is misclassified w.r.t. $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$, then z_I contains some missed example.

We explain the idea of our new random sampling algorithm. As before, we choose (according to some weight) a set R consisiting of r composed examples in Z, and then solve (P5) for R. In the original sampling algorithm, this sampling is repeated until no violator exists. Recall that we are regarding (P5) as an LP-type problem and that by "a violator of R", we mean a composed example that is misclassified with the current solution $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$ of (P5) obtained for R. Thanks to the above lemma, we do not have to go through all composed examples in order to search for a violator. A violator exists if and only if there exists some missed example w.r.t. $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$. Thus, our first idea is to use the existence of missed example for the stopping condition. That is, the sampling procedure is repeated until no missed example exists.

The second idea is to use the weight of examples $\boldsymbol{x}_i$ in X to define the weight of composed examples. Let u_i denote the weight of the ith example $\boldsymbol{x}_i$. Then for each composed example $\boldsymbol{z}_I \in Z$, its weight U_I is defined as the total of weights of all examples contained in z_I; that is, $U_I = \sum_{\boldsymbol{x}_i \in z_I} u_i$. We use symbols u and U to refer these two weight schemes; we sometimes, use these symbols to denote mapping from a set of (composed) examples to its total weight. For example, $u(X) = \sum_i u_i$, and $U(Z) = \sum_I U_I$. As explained below, it is computationally easy to generate each z_I with probability $U_I / U(Z)$.

Our third idea is to increase weights u_i if it is a missed example w.r.t. the current solution. More specifically, we double the weight u_i if $\boldsymbol{x}_i$ is a missed example w.r.t. the current solution

procedure OptMarginComposed
  $u_i \leftarrow 1$, for each i, $1 \le i \le m$;
  $r \leftarrow 6\alpha\beta n$; % Parameters α and β are given in the discussion below
  repeat % For example, we may set $\beta = k(n+1)$.
    $R \leftarrow$ choose r elements from Z randomly according to their weights;
    $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-) \leftarrow$ the solution of (P5) for R;
    $\widetilde{X}_{\mathrm{err}} \leftarrow$ the set of missed examples w.r.t. the above solution;
    if $u(\widetilde{X}_{\mathrm{err}}) \le u(X)/(3\beta)$ **then** $u_i \leftarrow 2u_i$ for each $\boldsymbol{x}_i \in \widetilde{X}_{\mathrm{err}}$;
  until no missed example exists;
  return the last solution;
end-procedure.

Figure 2: A Modified Random Sampling Algorithm

for R. Lemma 5.3 guarantees that the weight of some element of an final extremer gets doubled so long as there is some missed example. This property is crucial to estimate the number of iterations.

Now we state our new algorithm in Figure 2. In the following, we explain some important points on this algorithm.

Random Generation of R

We explain how to generate each $\boldsymbol{z}_I$ proportional to U_I. Again we only consider the generation of positive composed examples, and we assume that all positive examples are re-indexed as $\boldsymbol{x}_1, ..., \boldsymbol{x}_m$. Also for simplifying our notation, we reuse m amd M to denote m_+ and M_+ respectively.

Recall that each $\boldsymbol{z}_I$ is defined as $(\boldsymbol{x}_{i_1} + \cdots + \boldsymbol{x}_{i_k})/k$, where $\boldsymbol{x}_{i_j}$ is an element of z_I. Here we assume that $i_k < i_{k-1} < \cdots < i_1$. Then each $\boldsymbol{z}_I$ is uniquely corresponds to some k-tuple $(i_k, ..., i_1)$, and we identify here the index I of $\boldsymbol{z}_I$ and this k-tuple. Let $\mathcal{I}$ be the set of all such k-tuples $(i_k, ..., i_1)$ that satisfy $1 \le i_j \le m$ (for each j, $1 \le j \le k$) and $i_k < \cdots < i_1$. Here we assume the standard lexcographic order in $\mathcal{I}$.

As stated in the above algorithm, we keep the weights $u_1, ..., u_m$ of examples in X. By using these weights, we can calculate the total weight $U(Z) = \sum_i u_i$. Similarly, for each $\boldsymbol{z}_I \in Z$, it is not so difficult to compute the following accumulated weight. (See Lemma 5.5 of Appendix B.)

$$\overline{U}(I) \stackrel{\mathrm{def}}{=} \sum_{J \le I} U_J.$$

Thus, for generating $\boldsymbol{z}_I$, (i) choose p randomly from $\{1, ..., U(Z)\}$, and (ii) search for the smallest element I of $\mathcal{I}$ such that $\overline{U}(I) \ge p$. The second step can be done by the standard binary search in $\{1, ..., M\}$, which needs $\log M$ ($\le k \log m$) steps.

Stopping Condition and Number of Successful Iterations

The correctness of our stopping condition is clear from Lemma 5.3. We estimate the number of the repeat-iterations. Here again we say that the repeat-iteration is *successful* if the if-condition holds. We give an upper bound for the number of successful iterations.

Lemma 5.4. Set $\beta = k(n+1)$ in the algorithm. Then the number of successful iterations is at most $3k(n+1)\ln m$.

Our Hypothesis and the Sampling Lemma

Finally, the most important point is to estimate how often we would have successful iterations. At each repeat-iteration of our algorithm, we consider the ratio $\rho_{\mathrm{miss}} = u(\widetilde{X}_{\mathrm{err}})/u(X)$. Recall that the repeat-iteration is successful if this ratio is at most $1/(3\beta)$. Our hypothesis is that the ratio ρ_{miss} is, on average, bounded by $1/(3\beta)$. Here we discuss when and for which parameter β, this hypothesis would hold.

For the analysis, we consider "violators" to the local solution of (P5) obtained at each repeat-iteration. Let R be the set of r' composed examples randomly chosen from Z with the probability proportional to their weights determined by U. Recall a violator of R is a composed example $\boldsymbol{z}_I \in Z$ that is misclassified with the obtained solution for R. Let V be the set of violators, and let v_R be its weight under U. Recall also that the total weight of Z is $U(Z)$. Thus, by the Sampling Lemma, the ratio $\rho_{\mathrm{vio}} \stackrel{\mathrm{def}}{=} v_R/U(Z)$ is bounded as follows.

$$\mathrm{Exp}(\rho_{\mathrm{vio}}) \;\le\; \frac{(n+1)(U(Z)-r')}{r'+1} \cdot \frac{1}{U(Z)} \;\le\; \frac{n}{r'}.$$

From Lemma 5.3, we know that every violator should contain at least one missed example. On the other hand, every missed example would contribute to some violator. Hence, it seems reasonable to expect that the ratio ρ_{miss} is bounded by $\alpha \cdot \rho_{\mathrm{vio}}$ for some constant $\alpha \ge 1$, or at least it holds quite often if it is not always true. (It is still o.k. even if α is a low degree polynomial w.r.t. n.) Here we propose the following technical hypothesis.

(Hypothesis) For some $\alpha \ge 1$, we have $\rho_{\mathrm{miss}} \le \alpha \cdot \rho_{\mathrm{vio}}$.

Under this hypothesis, we have $\rho_{\mathrm{miss}} \le n/r'$ on average; thus, by taking $r = 6\alpha\beta n$, we can show that the expected ratio ρ_{miss} is at most $1/6\beta$, which implies as before that the expected number of iterations is at most twice as the number of successful iterations. Therefore the average number of iterations is bounded by $6k(n+1)\ln m$.

6. Concluding Remarks: Toward A Method for Finding Outliers

In computational learning theory, one of the recent important topics is to develop an effective method for handling data with inherent errors. Here by an "inherent error", we mean an error or noise that cannot be corrected by resampling. Typically, an example that is mislabeled and this mislabeled situation does not change even though we resample this example again. Many learning algorithms fail to work under the existence of such inherent errors. For example, AdaBoost [FS97] is sensitive to errors. Although some boosting algorithms (e.g., MadaBoost [DW00]) are robust against stochastic noise, even such algorithms cannot handle inherent errors appropriately either. SVMs are more robust against errors, but it is still the state of art to determine parameters for erroneous examples. More specifically, the complexity of classifiers and the degree D of the influence of errors are usually selected based on the experts' knowledge and experiences.

Let us fix a hypothesis class as the set of hyperplanes of the sample domain. Also suppose, for the time being, that the parameter D is somehow appropriately chosen. Then we can formally define erroneous examples — outliers — as we did in Section 3. Clearly, outliers can be identified by solving (P2); by using the obtained hyperplane, we can check whether a given example is an outlier or not. But it would be nice if we can find outliers on the course of our computation. As we discussed in Section 5, outliers are not only misclassified examples but also misclassified examples that commonly appear in support vector composed examples. Thus, if there is a good iterative way to solve (P5), we may be able to identify outliers by checking for commonly appearing misclassified examples in support vector composed examples of each local solution. We think that our second algorithm can be used for this purpose.

Also a randomized sampling algorithm for solving (P5) can be used to determine the parameter $D = 1/k$. Note that if we use k that is not large enough, then (P5) does not have a solution; there is no hyperplane separating composed examples. In this case, we would have more violators than we expect. Thus, by running a randomized sampling algorithm for (P5) several rounds, we can detect that the current choice of k is too small if an *unsuccessful iteration* (i.e., an iteration where the if-condition fails) occurs frequently. Thus, we can revise k at an earlier stage.

References

[AS93] I. Adler and R. Shamir, A randomized scheme for speeding up algorithms for linear and convex programming with high constraints-to-variable ratio, *Math. Programming* 61, 39–52, 1993.

[BB00] K.P. Bennett and E.J. Bredensteiner, Duality and geometry in SVM classifiers, in *Proc. the 17th Int'l Conf. on Machine Learning* (ICML'2000), 57–64, 2000.

[Ber95] D.P. Bertsekas, *Nonlinear Programming*, Athena Scientific, 1995.

[BMS00] P.S. Bradley, O.L. Mangasarian, and D.R. Musicant, Optimization methods in massive datasets, in *Handbook of Massive Datasets* (J. Abello , P.M. Pardalos, and M.G.C. Resende, eds.) Kluwer Academic Pub., 2000, to appear.

[Lin01] C.J. Lin, On the convergence of the decomposition method for support vector machines, *IEEE Trans. on Neural Networks*, 2001, to appear. (Also available from `http://www.csie.ntu.edu.tw/~cjlin/papers/`.)

[Cla88] K.L. Clarkson, A Las Vegas algorithm for linear programming when the dimension is small, in *Proc. the 29th IEEE Sympos. on Foundations of Comp. Sci.* (FOCS'88), 1988. (See below for a better version.)

[Cla95] K.L. Clarkson, Las Vegas algorithms for linear and integer programming, *J.ACM* 42, 488–499, 1995.

[CV95] C. Cortes and V. Vapnik, Support-vector networks, *Machine Learning* 20, 273–297, 1995.

[CST00] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge University Press 2000.

[DW00] C. Domingo and O. Watanabe, MadaBoost: A modification of AdaBoost, in *Proc. of 13th Ann. Conference on Computational Learning Theory* (COLT'00), 180–189, 2000.

[FS97] Y. Freund and R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, *J. Comput. Syst. Sci.* 55, 119–139, 1997.

[GW00] B. Gärtner and E. Welzl, A simple sampling lemma: Analysis and applications in geometric optimization, *Discr. Comput. Geometry*, 2000, to appear. (Also available from `http://www.inf.ethz.ch/personal/gaertner/publications.html`.)

[KG01] S.S. Keerthi and E.G. Gilbert, Convergence of a generalized SMO algorithm for SVM classifier design, Technical Report CD-00-01, Dept. of Mechanical and Production Eng., National University of Singapore, 2000. (Available from `http://guppy.mpe.nus.edu.sg/~mpessk/svm/conv_ml`.)

[OFG97] E. Osuna, R. Freund, and F. Girosi, An improved training algorithm for support vector machines, in *Proc. IEEE Workshop on Neural Networks for Signal Processing*, 276–285, 1997.

[Pla99] J. Platt, Fast training of support vector machines using sequential minimal optimization, in *Advances in Kernel Methods – Support Vector Learning* (B. Scholkopf, C.J.C. Burges, and A.J. Smola, eds.), MIT Press, 185–208, 1999.

[SS98] A.J. Smola and B. Scholkopf, A tutorial on support vector regression, NeuroCOLT Technical Report NC-TR-98-030, Royal Holloway College, University of London, 1998.

Appendix A: Proofs of Lemma/Theorem in Section 2

The algorithm and the theorems stated in Section 2 are obtained almost immediately from more general ones given in [GW00]. But for the reader's convenience, we state their proofs here.

We first explain the abstract framework for discussing randomized sampling techniques that was given by Gärtner and Welzl [GW00]. (Note that the idea of this Sampling Lemma can be found in the paper by Clarkson [Cla88], where a randomized algorithm for linear programming has been proposed. Indeed a similar idea has been used [AS93] to design an efficient randomized algorithm for quadratic programming. But here we explain the framework and the algorithm following [GW00].)

Consider any LP-type problem $(\mathcal{D}, \phi)$. Let $\mathcal{U}$ be a set consisting u elements of $\mathcal{D}$. $\mathcal{U}$ may be a *multiple set*, a set containing some element more than once. In order to discuss the case where elements of $\mathcal{D}$ are chosen to $\mathcal{R}$ according to their weights, we had better use $\mathcal{U}$ as an example domain (instead of $\mathcal{D}$) and consider simply that $\mathcal{R}$ is a subset of $\mathcal{U}$. Though obvious, the following relation is important for our discussion.

$$e \text{ violates } \mathcal{R} \;\Leftrightarrow\; e \text{ is extreme in } \mathcal{R} \cup \{e\}. \tag{3}$$

Define $v_{\mathcal{R}}$ and $x_{\mathcal{R}}$ to be the number of violators and extremers of $\mathcal{R}$ in $\mathcal{U}$ respectively. The following bound, which is also easy from the definition, is important.

$$x_{\mathcal{R}} \;\leq\; \delta \;(= \text{ the combinatorial dimension of } (\mathcal{D}, \phi)). \tag{4}$$

First we show the Sampling Lemma given in Section 2.

Lemma 2.1 Assume some weight scheme u on $\mathcal{D}$ that gives an integer weight to each element of $\mathcal{D}$. Let $u(\mathcal{D})$ denote the total weight. For a given r, $0 \leq r < u(\mathcal{D})$, we consider the situation where r elements of $\mathcal{D}$ are chosen randomly according to their weights. Let $\mathcal{R}$ to denote the set of chosen elements, and let $v_{\mathcal{R}}$ be the weight of violators of $\mathcal{R}$. Then we have the following bound.

$$\mathrm{Exp}(v_{\mathcal{R}}) \;\leq\; \frac{u(\mathcal{D}) - r}{r+1} \cdot \delta. \tag{5}$$

Proof. For a given weight scheme u on $\mathcal{D}$, we define $\mathcal{U}$ as a multiple set containing exactly $u(x)$ copies of each element x in $\mathcal{D}$. Then choosing a set $\mathcal{R}$ of r elements randomly from $\mathcal{D}$ according to their weights is essentially the same as choosing one set $\mathcal{R}$ from all possible $\binom{u(\mathcal{D})}{r}$ subsets of $\mathcal{D}$ uniformly at random. Thus, below we consider this latter random situation.

For randomly selected $\mathcal{R}$, consider the values $v_{\mathcal{R}}$ and $x_{\mathcal{R}}$. Let v_r and x_r denote their expected values $\mathrm{Exp}(v_{\mathcal{R}})$ and $\mathrm{Exp}(x_{\mathcal{R}})$. Then we have

$$(r+1)v_r \;=\; x_{r+1}(u(\mathcal{D})-r). \tag{6}$$

The bound of the lemma follows from (4) and (6). On the other hand, the above relation (6) is derived easily from the following. (Here we use $\binom{\mathcal{U}}{r}$ to denote the set of all r-element subsets of $\mathcal{U}$.)

$$\begin{aligned}
\binom{u(\mathcal{D})}{r} v_r &= \sum_{\mathcal{R}\in\binom{u(\mathcal{D})}{r}} \sum_{e\in\mathcal{U}-\mathcal{R}} [\; e \text{ violates } R \;] \\
&= \sum_{\mathcal{R}\in\binom{\mathcal{U}}{r}} \sum_{e\in\mathcal{U}-\mathcal{R}} [\; e \text{ is extreme in } \mathcal{R}\cup\{e\} \;] \\
&= \sum_{\mathcal{Q}\in\binom{\mathcal{U}}{r+1}} \sum_{e\in\mathcal{Q}} [\; e \text{ is extreme in } \mathcal{Q} \;] \;=\; x_{r+1}\binom{u(\mathcal{D})}{r+1}.
\end{aligned}$$

□

Theorem 2.2 The average number of iterations executed in the OptMargin algorithm is bounded by $6\delta \ln m = O(n \ln m)$. (Recall $|X| = m$ and $\delta = n+1$.)

Proof. We say a repeat-iteration is *successful* if the if-condition holds in the iteration.

We first bound the number of successful iterations. For this, we analyze how the total weight $u(X)$ increases. Consider the execution of any successful iteration. Since $u(V) \le u(X)/3\delta$, by doubling the weight of all examples in V, i.e., all violators, $u(X)$ increases by at most $u(X)/(3\delta)$. Thus, after t successful iterations, we have $u(X) \le m(1+1/(3\delta))^t$. (Note that $u(X)$ is initially m.)

Let $X_0 \subseteq X$ be the set of support vectors of (P1). Note that if all elements of X_0 are chosen to R, i.e., $X_0 \subseteq R$, then there should be no violator for R. Thus, at each successful iteration (if it is not the end) some $\boldsymbol{x}_i$ of X_0 must not be in R, which in turn is a violator of R. Hence, $u(\boldsymbol{x}_i)$ gets doubled. Since $|X_0| \le \delta$, there is some $\boldsymbol{x}_i$ in X_0 that gets doubled at least once every δ successful iterations. Therefore, after t successful iterations, $u(\boldsymbol{x}_i) \ge 2^{t/\delta}$.

Therefore, we have the following upper and lower bounds for $u(X)$.

$$2^{t/\delta} \;\le\; u(X) \;\le\; m(1+1/(3\delta))^t.$$

This implies that $t < 3\delta \ln m$ (if the repeat-condition does not hold after t successful iterations). That is, the algorithm terminates within $3\delta \ln m$ successful iterations.

Next estimate how often successful iteration occurs. Here we use the Sampling Lemma. Consider the execution of any repeat-iteration. Let u be the current weight on X, and let R and V be the set chosen at this iteration and the set of violators of R. Then this R corresponds to $\mathcal{R}$ in the Sampling Lemma, and we have $u(V) = v_{\mathcal{R}}$. Hence from the above bound (5), we can bound the expectation v_r of $u(V)$ by $(u(X)-r)\delta/(r+1)$, which is smaller than $u(X)/(6\delta)$ by our choice of r. Thus, the probability that the if-condition is satisfied is at least $1/2$. This implies that the expected number of iterations is at most twice as large as the number of successful iterations. Therefore, the algorithm terminates *on average* within $2 \cdot 3\delta \ln m$ steps. □

Appendix B: Proofs of Lemmas in Section 5

Lemma 5.1 Let $\boldsymbol{z}_I$ be any positive final extremer, i.e., an element of Z_0 such that $y_I = 1$. Then the following properties hold. (Though the lemma is stated only for the positive case, i.e., the case $y_I = 1$, the corresponding properties hold for the negative case $y_I = -1$.)
(a) $\boldsymbol{w}^* \cdot \boldsymbol{z}_I = \eta_+^*$.
(b) $X_{\mathrm{err},+} \subseteq z_I$.
(c) For every $\boldsymbol{x}_i \in z_I$, if $\boldsymbol{x}_i \notin X_{\mathrm{err},+}$, then we have $\boldsymbol{w}^* \cdot \boldsymbol{x}_i = \theta_+^*$.

Proof. (a) Since Z_0 is the set of final extremers, (P5) can be solved only with the constraints corresponding to elements in Z_0. Suppose that $\boldsymbol{w}^* \cdot \boldsymbol{z}_J > \eta_+^*$ (resp., $\boldsymbol{w}^* \cdot \boldsymbol{z}_J < \eta_-^*$) for some positive (resp., negative) $\boldsymbol{z}_J \in Z_0$ including $\boldsymbol{z}_I$ of the lemma. Let Z' be the set of such $\boldsymbol{z}_J$'s of Z_0. If Z' indeed contained all positive examples in Z_0, then we could set θ_+ with $\theta_+^* - \epsilon$ for some $\epsilon > 0$ and still satisfy all the constraints, which contradicts the optimality of the solution. Hence, we may assume that $Z_0 - Z'$ still has some positive example. Then it is well known (see, e.g., [Ber95]) that a local optimal solution to the problem (P5) with the constraints corresponding to elements in Z_0 is also locally optimal to the problem (P5) with the constraints corresponding to only elements in $Z_0 - Z'$. Furthermore, since (P5) is a convex programming, a local optimal solution is globally optimal. Thus, the original problem (P5) is solved with the constrains corresponding to elements in $Z_0 - Z'$. This contradicts our assumption that Z_0 is the set of final extremers.

(b) Consider the KKT-point $(\boldsymbol{w}^*, \theta_+^*, \theta_-^*, \boldsymbol{\xi}^*, \boldsymbol{s}^*, \boldsymbol{u}^*)$ of (P2). Then the point must satisfy the following so called KKT-condition. (Below we use i to denote indices of examples, and let P and N respectively denote indices i of examples such that $y_i = 1$ and $y_i = 0$. We use $\boldsymbol{e}$ to denote the vector with 1 at every entry.)

$$
\begin{array}{rclrcl}
\boldsymbol{w}^* - \sum_{i \in P} s_i \boldsymbol{x}_i + \sum_{i \in N} s_i^* \boldsymbol{x}_i & = & 0, & \boldsymbol{De} - \boldsymbol{s}^* - \boldsymbol{u}^* & = & 0, \\
-1 + \sum_{i \in P} s_i^* & = & 0, & -1 + \sum_{i \in N} s_i^* & = & 0, \\
\forall i \in P \; [\; s_i^*(\boldsymbol{w}^* \cdot \boldsymbol{x}_i - \theta_+^* + \xi_i^*) & = & 0 \;], & \forall i \in N \; [\; s_i^*(\boldsymbol{w}^* \cdot \boldsymbol{x}_i - \theta_-^* - \xi_i^*) & = & 0 \;],
\end{array}
$$
$$
\boldsymbol{u}^* \cdot \boldsymbol{\xi}^* \;=\; 0 \text{ (which means } (\boldsymbol{De} - \boldsymbol{s}^*) \cdot \boldsymbol{\xi}^* = 0), \quad \text{and} \quad \boldsymbol{\xi}^*, \boldsymbol{u}^*, \boldsymbol{s}^* \;\geq\; 0.
$$

Note that $(\boldsymbol{w}^*, \theta_+^*, \theta_-^*, \boldsymbol{\xi}^*)$ is an optimal solution of (P2), since (P2) is a convex minimization problem. From these requirements, we have the following relation. (Note that the condition $\boldsymbol{s}^* \leq \boldsymbol{De}$ below is derived from the requirements $\boldsymbol{De} - \boldsymbol{s}^* - \boldsymbol{u}^* = 0$ and $\boldsymbol{u}^* \geq 0$.)

$$
\begin{array}{l}
\boldsymbol{w}^* \;=\; \sum_{i \in P} s_i^* \boldsymbol{x}_i - \sum_{i \in N} s_i^* \boldsymbol{x}_i, \\
\sum_{i \in P} s_i^* \;=\; 1, \quad \sum_{i \in N} s_i^* \;=\; 1, \quad \text{and} \quad 0 \;\leq\; \boldsymbol{s}^* \;\leq\; \boldsymbol{De}.
\end{array}
$$

In fact, $\boldsymbol{s}^*$ is exactly the optimal solution of (P3).

Here by the equivalence of (P4) and (P5), we see that the final extremers are exactly points contributing to the solution of (P4). That is, we have $\boldsymbol{z}_I \in Z_0$ if and only if $s_I^* > 0$, where s_I^* is the Ith element of the solution of (P4). Furthermore, it follows the equivalence between (P3) and (P4), for any i, we have

$$\frac{1}{k} \cdot \sum_{I: \boldsymbol{x}_i \in z_I} s_I^* \;=\; s_i^*. \tag{7}$$

Recall that each z_I is defined as the center of k examples of X. Hence, to show that every $\boldsymbol{x}_i \in X_{\mathrm{err},+}$ appears in all positive final extremers, it suffices to show that $s_i^* = 1/k$ for every $\boldsymbol{x}_i \in X_{\mathrm{err},+}$, which follows from the following argument. For any $\boldsymbol{x}_i \in X_{\mathrm{err},+}$, since $\xi_i^* > 0$, it follows from the requirements $(D\boldsymbol{e} - \boldsymbol{s}^*) \cdot \boldsymbol{\xi}^* = 0$ and $D\boldsymbol{e} - \boldsymbol{s}^* \geq 0$ that $D - s_i^* = 0$; that is, $s_i^* = 1/k$ for any $\boldsymbol{x}_i \in X_{\mathrm{err},+}$.

(c) Consider any index i in P such that $\boldsymbol{x}_i$ appears in some of the final extremer $\boldsymbol{z}_I \in Z_0$. Since $s_I^* > 0$, we can show that $s_i^* > 0$ by using the equation (7). Hence, from the requirement $s_i(\boldsymbol{w}^* \cdot \boldsymbol{x}_i - \theta_+^* + \xi_i^*) = 0$, we have

$$\boldsymbol{w}^* \cdot \boldsymbol{x}_i - \theta_+^* + \xi_i^* \;=\; 0.$$

Thus, if $\boldsymbol{x}_i \notin X_{\mathrm{err}}$, i.e., it is not an outlier or $\xi_i^* = 0$, then we have $\boldsymbol{w}^* \cdot \boldsymbol{x}_i = \theta_+^*$. □

Lemma 5.3 For any subset R of Z, let $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$ be the solution of (P5) for R. Then if there exists a missed example w.r.t. $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$, then we have some composed example in Z that is misclassified w.r.t. $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$. On the other hand, for any composed example $\boldsymbol{z}_I \in Z$, if it is misclassified w.r.t. $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$, then z_I contains some missed example.

Proof. We consider again only the positive case. Suppose that some missed positive example $\boldsymbol{x}_i$ exists. By definition, we have $\widetilde{\boldsymbol{w}} \cdot \boldsymbol{x}_i < \widetilde{\theta}_{\mathrm{err},+}$, and there exists some extremer $\boldsymbol{z}_I \in \widetilde{Z}_{0,+}$ that does not contain $\boldsymbol{x}_i$. Clearly, $\boldsymbol{z}_I$ contains some example $\boldsymbol{x}_j$ such that $\widetilde{\boldsymbol{w}} \cdot \boldsymbol{x}_j \geq \widetilde{\theta}_{\mathrm{err},+}$. Then we can easily see that a composed elements $\boldsymbol{z}_J$ consisting of $z_I - \{\boldsymbol{x}_j\} \cup \{\boldsymbol{x}_i\}$ does not satisfy the constraint $\widetilde{\boldsymbol{w}} \cdot \boldsymbol{z}_J \geq \widetilde{\eta}_+$.

For proving the second statement, note first that any "misclassified" original example $\boldsymbol{x}_i$, i.e., an example for which the inequality (2) holds, is either a missed example or an element of $\widetilde{X}_{\mathrm{err},+}$. Thus, if a composed element $\boldsymbol{z}_I$ does not contain any missed example, then it cannot contain any misclassified examples other than those in $\widetilde{X}_{\mathrm{err},+}$. Then it is easy to see that $\widetilde{\boldsymbol{w}} \cdot \boldsymbol{z}_I \geq \widetilde{\eta}_+$; that is, $\boldsymbol{z}_I$ is not misclassified w.r.t. $(\widetilde{\boldsymbol{w}}, \widetilde{\eta}_+, \widetilde{\eta}_-)$. □

Lemma 5.4 Set $\beta = k(n+1)$ in the algorithm. Then the number of successful iterations is at most $3k(n+1)\ln m$.

Proof. For any $t > 0$, we consider the total weight $u(X)$ after t successful iterations. As before, we can give an upper bound $u(X) \leq m(1 + 1/(3\beta))^t$. On the other hand, some missed example exists at each repeat-iteration, and from Lemma 5.3, we can indeed find it in any violator, in particular, some final extremer $\boldsymbol{z}_I \in Z_0$. Thus, there must be some element $\boldsymbol{x}_i$ of $\cup_{\boldsymbol{z}_I \in Z_0} z_I$ whose weight u_i gets doubled at least once every $k(n+1)$ steps. (Recall that $|Z_0| \leq n+1$.) Hence, we have

$$2^{t/k(n+1)} \;\leq\; u(X) \;\leq\; m(1 + 1/(3\beta))^t.$$

This implies, under the above choice of β, that $t < 3k(n+1)\ln m$. □

Lemma 5.5 There is an efficient way to compute $\overline{U}(I) \stackrel{\text{def}}{=} \sum_{J \le I} U_J$.

Proof. First we prepare some notations. Define $\overline{V}(I)$ as follows for each $I \in \mathcal{I}$.

$$\overline{V}(I) \stackrel{\text{def}}{=} \sum_{J \ge I} U_J.$$

Then it is easy to see that (i) $U_0 = \overline{V}((1,2,3,...,k))$, and (ii) $\overline{U}(I) = U_0 - \overline{V}(I) + (u_{i_k} + u_{i_{k-1}} + \cdots + u_{i_1})$ for each $I = (i_k, i_{k-1}, ..., i_1)$. Thus, for the lemma, it suffices to show how to compute $\overline{V}(I)$.

Consider any given $I = (i_k, i_{k-1}, ..., i_1)$ in $\mathcal{I}$. Also for any j, $1 \le j \le k$, we consider the prefix $I'_j = (i_j, i_{j-1}, ..., i_1)$ of I, and define the following values.

$$\begin{aligned} N_j &\stackrel{\text{def}}{=} \#\text{ of } I' \text{ such that } I' \ge I'_j, \quad \text{and} \\ V_j &\stackrel{\text{def}}{=} \sum_{I'=(i'_j,...,i'_1) \ge I'_j} (u_{i'_j} + u_{i'_j - 1} + \cdots + u_{i'_1}). \end{aligned}$$

Then clearly we have $\overline{V}(I) = V_k$, and our task is to compute V_k, which can be done inductively by using the following relation that holds for each j, $1 \le j \le k$. (Let $N_0 = 1$.)

$$N_j = N_{j-1} + \binom{m - i_j}{j}, \quad \text{and} \quad V_j = u_{i_j} \cdot N_{j-1} + \sum_{i_j+1 \le i \le m} u_i \cdot \binom{m-i}{j-1}.$$

□