Frank-Wolfe-based Boosting Algorithms and Its Extensions

三星, 諒太朗

Frank-Wolfe-based Boosting Algorithms and Its Extensions

Ryotaro Mitsuboshi

July, 2024

Acknowledgements

First of all, I would like to show my most significant appreciation to Professor Eiji Takimoto, my superviser, and my thesis committee member. He guided me to the world of theories of machine learning. I have learned a lot of valuable techniques and the ways of thinking from him. He also provided me knowledge and opportunities for valuable experiences.

I also express my deep gratitude to Professor Kohei Hatano. He also gives me many opportunities for my presentations and collaborations. The discussions with him gave me many ideas. Furthermore, he supported my life by employing me as a part-time researcher of RIKEN AIP.

In addition, I sincerely appreciate Professor Masayuki Takeda, Professor Hideo Bannai, Professor Shunsuke Inenaga, and Assistant Professor Yuto Nakashima. Thanks to the weekly discussion with them, I could acquire logical thinking and presentation skills. I also profoundly thank to Professor Jun'ichi Takeuchi and Associate Professor Hayato Waki, my thesis committee members. They gave me many suggestions to me, which improved my thesis.

I would like to express my appreciation to all co-authors: Dr. Holakou Rahmanian, Mr. Haruki Hamasaki, and Mr. Yuta Kurokawa. The discussions with them were fascinating and instructive.

Our research is supported by JSPS (Japan Society for the Promotion of Science) and RIKEN AIP.

I am also greatful to past and present members and technical staffs of our laboratory. In particular, I thank Assistant Professor Takuya Mieno, Dr. Yuta Fujishige, Dr. Kazuya Tsuruta, and Dr. Mitsuru Funakoshi, Mr. Keigo Hayata, and Mr. Yuki Urabe. The discussions with them were valuable and fascinating. Thanks to them, I could enjoy my laboratory life.

Last but not least, I thank my family for their support.

Abstract

The importance of developing machine learning algorithms is increasing day by day. Many algorithms have been invented for decades, but most have yet to theoretical results. Theoretical results yield new insights into algorithms, so developing algorithms along with theoretical developments is essential. Most machine learning algorithms are iterative optimization algorithms that update their solution for every round, so one wants to find an optimal solution as fast as possible. Some algorithms are known as good ones empirically, but theoretical analysis shows that some have a bad example that makes the algorithm converge slowly. On the other hand, some algorithms are shown to converge in polynomial time, while they are slow in practice since they need to solve optimization problems in their internal update. Therefore, one wants to develop an algorithm theoretically and practically well. This thesis has two main backbones. One is boosting algorithms, and the other is Frank-Wolfe algorithms. All algorithms we develop come from these algorithms and their connection. The connection is also our contribution.

This thesis mainly focuses on boosting algorithms, one of the most popular frameworks in machine learning. Intuitively, boosting is a class of algorithms that convert a set of low-performance predictors into a high-performance one. We focus on boosting algorithms for binary classification, although many variants of settings exist. Boosting algorithms for binary classification have been developed, but most are trying to improve empirical performance. In other words, they try to find a low-performance predictor as fast as possible. Theoretically, one wants to show how many hypotheses are sufficient to achieve some goal.

This thesis starts by analyzing boosting algorithms that maximize the *margin* instead of minimizing some losses. After that, we extend to the regression setting that handles the insensitive loss. Then, we extend these techniques to other fields. One is the extended formulation, a technique for optimization. The other is online combinatorial optimization, a class of machine learning. The following paragraphs summarize each contribution in

this thesis.

First, we propose a unified view for boosting algorithms that optimize the soft margin. With this view, some boosting algorithms for maximizing the soft margin are instances of the Frank-Wolfe algorithms. In addition, this unified view provides a significantly simplified analysis technique for some past algorithms. Furthermore, it invokes a better algorithm.

Second, we propose a boosting algorithm for regression whose loss is the insensitive loss function. The insensitive loss is convex but non-smooth; it is hard to analyze the convergence rate. We use a similar technique to the one in the unified view, and the resulting algorithm converges favorably.

Thirdly, we propose a boosting algorithm over a compressed space. Although this research was studied a few years ago, our algorithm works in a more general setting. Furthermore, we extend this algorithm to an extended formulation for general large-scale optimizations. The resulting algorithm can apply not only to convex optimization problems but also to mixed-integer problems.

Finally, we adopt the unified view attained at the beginning of this thesis to Metarounding, a tool for online combinatorial linear optimization. The proposed algorithm comes from the boosting algorithm but has been adjusted for this problem setting. As a result, the algorithm is a generalization of a boosting algorithm. Our algorithm improves the convergence rate from the previous one under the same setting.

All algorithms in this thesis verified their practical advantage in numerical experiments. Codes are available for free.

Contents

Chapter 1

Introduction

The importance of developing machine learning algorithms is increasing day by day. Many algorithms have been invented for decades, but most have yet to theoretical results. Some practical algorithms have proved to converge slowly for the worst case. Furthermore, theoretical results yield new insights into algorithms, so developing algorithms along with theoretical developments is essential. On the other hand, theoretically guaranteed algorithms are often slow in practice since they solve complex sub-problems. Therefore, one wants algorithms that have a convergence rate and are practical.

This thesis starts with boosting algorithms, one of the most popular frameworks in machine learning. Intuitively, boosting is a class of algorithms that iteratively collects a low-performance predictor from a weak learning algorithm and combines them into a high-performance one. We focus on boosting algorithms for binary classification, although many variants of settings exist. Boosting algorithms for binary classification have been developed, but most try to improve the weak learning algorithms. For example, XGBoost [17] and LightGBM [49] improve the performance of the decision tree algorithm, and CatBoost [72] provides a better feature engineering technique for categorical features. Although these algorithms perform well for real datasets, theoretical developments are few. Most algorithms use the Gradient Boosting Machine (GBM, for short) [29], which is known to converge if the objective function is smooth [36]. The objective function is an empirical (surrogate) loss in many situations. So, choosing a proper objective yields a fast algorithm. GBM is also known as an instance of the coordinate descent algorithm [58]. On the other hand, soft margin optimization is known as a popular boosting objective [21]. Intuitively, the objective aims to maximize confidence for all training points, ignoring some outliers. The resulting predictor performs well compared to the

one that minimizes the empirical loss [9, 81]. Since the objective function of the soft margin optimization does not have the strong-smoothness property, one cannot directly apply GBM to obtain an iteration bound. Many algorithms have been invented to deal with this issue [21, 88, 97, 99]. Some algorithms have a favorable convergence rate but are slow in practice, and vice versa. Therefore, we aim to design a boosting algorithm for soft margin optimization that has a convergence rate and is practical for real-world applications.

To achieve this goal, we use the well-known optimization algorithms, Frank-Wolfe algorithms. Frank-Wolfe algorithms are the class of first-order optimization algorithms over a closed and convex set. This thesis provides a unified view of some soft margin boosting algorithms via Frank-Wolfe. This insight provides a boosting algorithm with the same iteration bound to the previous works and converges fast for real-world datasets. The insight is also helpful in developing a boosting algorithm for a regression setting with a particular loss function. Furthermore, we divert the technique to other fields, such as extended formulations and online combinatorial linear optimizations. As Fujita et al. [31] proposed an algorithm that emulates AdaBoost* [76] over a decision diagram, we propose ERLPBoost [99] over the one. The resulting algorithm may yield a non-optimal solution but is faster than the original ERLPBoost. We also verify the effectiveness of our solution by numerical experiments. For the online combinatorial linear optimization, we develop a Metarounding algorithm. The previous Metarounding algorithm [30] was designed based on SoftBoost [97]. We design a Metarounding algorithm based on ERLPBoost [99]. The convergence rate for our algorithm is much better than that of Fujita et al. Furthermore, our algorithm can be seen as a generalization of ERLPBoost.

In summary, we obtain four practical algorithms with convergence rates. We verified the effectiveness of all algorithms by numerical experiments on artificial and real-world datasets.

1.1 Contributions

This thesis has two backbones. One is boosting algorithms, and the other is Frank-Wolfe algorithms. All contributions came and developed from either of them. Our contribution starts with developing a boosting algorithm for binary classification via the Frank-Wolfe technique. The following contributions use the insights into other fields, such as regression, extended formulation, and online combinatorial linear optimization.

Broadly speaking, this thesis proposes:

1. A unified view for soft margin maximizing boosting algorithms and a boosting scheme that comes from Frank-Wolfe algorithms,

2. A Frank-Wolfe-based boosting algorithm for regression whose objective function is the soft insensitive loss,

3. An extended formulation for general optimization problems that have linear constraints and its application to the soft margin optimization, and

4. A Metarounding algorithm, a tool for online combinatorial linear optimization, that is designed based on a Frank-Wolfe-based boosting algorithm.

Each of them corresponds to Chapters 3 to 6 in this order. The following paragraphs summarize each of the contributions.

First, we bring a new interpretation for soft margin boosting via the Fenchel duality theorem. As GBM can be seen as an instance of a coordinate descent algorithm, we show that some soft margin boosting algorithms can be seen as instances of Frank-Wolfe algorithms. The duality view provides us with an analysis technique so one can prove the convergence rate in a much simpler way. Furthermore, with this insight, we propose a general boosting algorithm for the soft margin optimization that can be combined with an arbitrary algorithm. With an appropriate choice of algorithm, our algorithm can reproduce ERLPBoost [99] and Corrective ERLPBoost [88]. Given a set of m-training examples for binary classification and a hyperparameter $\nu \in [1, m]$, our algorithm finds ϵ-approximate solution of the soft margin optimization in $O(\ln(m/\nu)/\epsilon^2)$ iterations. Furthermore, our analysis simplifies the ones for ERLPBoost and Corrective ERLPBoost. Chapter 3 handles this topic.

Second, using a similar technique, we propose a regression boosting algorithm for the insensitive loss. The insensitive loss regards a small loss as zero. As discussed in Chapter 4, this setting is the ℓ_1-version of the one in ν-support vector regression algorithm [83]. The problem is considered in some papers [21, 24]. Similar to the soft margin optimization, the loss function is not smooth, so GBM does not guarantee any convergence rate. We propose a Frank-Wolfe-based boosting algorithm that converges in polynomial time. Similar to the first contribution, the resulting algorithm takes m-examples for regression and a parameter $\nu \in [1, m]$ and then returns an ϵ-approximate solution of the objective function in $O(\ln(m/\nu)/\epsilon^2)$ iterations. Chapter 4 handles this topic.

Thirdly, we propose an extended formulation for solving large-scale optimization problems. Our method compresses the set of constraints into a data structure called Nondeterministic ZDD (NZDD, for short) [31], and then solves the compressed problem to get an optimal solution. We also emulate ERLPBoost [99] over the NZDD and show the convergence rate, which depends on the size of the data structure for compression. Emulating a boosting algorithm over an NZDD was studied by Fujita et al. [31], which works when the training examples are separable. In this sense, our algorithm is a generalization of theirs. Although the proposed algorithm may return a different solution to the one before compression, we verify the effectiveness of our algorithm in experiments. In addition, we demonstrate some experiment for mixed integer programs. As the result shows, our work is effective when the compression works. Chapter 5 handles this topic.

Lastly, we consider Metarounding, a tool for the online combinatorial linear optimization problem, by diverting the insights of boosting algorithms. Roughly speaking, Metarounding converts a point $\boldsymbol{x} \in \mathrm{relax}(\mathcal{C})$ of a closed convex set $\mathrm{relax}(\mathcal{C}) \subset \mathbb{R}^n$ of a combinatorial set $\mathcal{C} \subset \mathrm{relax}(\mathcal{C})$ into a combinatorial concept $\boldsymbol{c} \in \mathcal{C}$ whose loss is not so bad in expectation. If a Metarounding exists, one can combine it with an online linear optimization algorithm $\mathcal{A}$ over $\mathrm{relax}(\mathcal{C})$ to construct an online combinatorial linear optimization algorithm whose α-regret is the same as the one for the 1-regret of $\mathcal{A}$. In this sense, Metarounding is a generic tool for online combinatorial linear optimization. The previous Metarounding algorithm by Fujita et al. [30] finds an ϵ-approximate solution in $O(n^2 \ln(n)/\epsilon^2)$. Our algorithm drops the n^2 term under the same assumption; that is, the proposed algorithm finds an ϵ-approximate solution in $O(\ln(n)/\epsilon^2)$. The computational complexity per iteration is the same, so our algorithm is superior to Fujita et al.'s [30]. We also verify the performance of our algorithm by some numerical experiments. Chapter 6 handles this topic.

Chapter 7 summarizes this thesis and leaves some future works.

Chapter 2

Preliminary

Throughout this thesis, we denote scalars by lower case letters, vectors by bold lower case letters, and matrices by upper case letters. We denote the ith entry of a vector $\boldsymbol{w}$ by w_i, and we denote by $A_{i,j}$ the (i,j)th entry of a matrix A. O denotes the zero matrix, and I_n denotes the n-dimensional identity matrix. We omit the subscript n of I_n if the dimension is clear from the context. $\mathbf{1}_n$ denotes the n-dimensional vector whose entries are 1. We omit the dimension and denote $\mathbf{1}$ if it is clear from the context. We denote the jth canonical basis vector by $\boldsymbol{e}_j$.

The inner product of two vectors $\boldsymbol{x}, \boldsymbol{y}$ is denoted by $\boldsymbol{x} \cdot \boldsymbol{y}$ or $\boldsymbol{x}^\top \boldsymbol{y}$. The ℓ_p-norm of a vector $\boldsymbol{x}$ is denoted by $\|\boldsymbol{x}\|_p$. Especially, we use ℓ_1-norm $\|\boldsymbol{x}\|_1 = \sum_i |x_i|$, ℓ_2-norm $\|\boldsymbol{x}\|_2 = \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}}$, and ℓ_∞-norm $\|\boldsymbol{x}\|_\infty = \max_i |x_i|$. We say that a symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite if $\boldsymbol{x}^\top A \boldsymbol{x} \geq 0$ holds for all $\boldsymbol{x} \in \mathbb{R}^n$. In other words, all eigenvalues of A are non-negative. We denote the positive semi-definiteness of A by $A \succeq O$. For two matrices $A, B \in \mathbb{R}^{n \times n}$, we denote $A - B \succeq O$ by $A \succeq B$.

Basically, sets are represented by calligraphic capital letters. Exceptionally, we denote n-dimensional probability simplex by $\Delta_n = \{\boldsymbol{d} \in [0,1]^n \mid \|\boldsymbol{d}\|_1 = 1\}$, and the set $\{1, 2, \ldots, n\}$ is denoted by $[n]$. For a finite set $\mathcal{C}$, we denote the probability simplex $\{\boldsymbol{d} \in [0,1]^{\mathcal{C}} \mid \|\boldsymbol{d}\|_1 = 1\}$ by $\Delta_{\mathcal{C}}$. For sets $\mathcal{C}, \mathcal{D} \subset \mathbb{R}^n$, the set operations, union, intersection, and difference, are denoted by $\mathcal{C} \cup \mathcal{D}$, $\mathcal{C} \cap \mathcal{D}$, and $\mathcal{C} \setminus \mathcal{D}$, respectively. In addition, we use the notion $\mathcal{C} - \mathcal{D} = \{\boldsymbol{c} - \boldsymbol{d} \mid \boldsymbol{c} \in \mathcal{C}, \boldsymbol{d} \in \mathcal{D}\}$, and for a matrix $A \in \mathbb{R}^{m \times n}$, we write $A\mathcal{C} = \{A\boldsymbol{c} \mid \boldsymbol{c} \in \mathcal{C}\}$. The core of a set $\mathcal{C} \subset \mathbb{R}^n$ is denoted by $\mathrm{core}(\mathcal{C}) = \{\boldsymbol{w} \in \mathcal{C} \mid \forall \boldsymbol{v} \in \mathbb{R}^n, \exists t > 0, \forall \tau \in [0,t], \boldsymbol{w} + \tau \boldsymbol{v} \in \mathcal{C}\}$. Furthermore, for a set $\mathcal{C} \subset \mathbb{R}^n$, we introduce the indicator function $\mathbb{I}_{\mathcal{C}} : \mathbb{R}^n \to \{0, +\infty\}$ defined by $\mathbb{I}_{\mathcal{C}}(\boldsymbol{x}) = 0$ iff $\boldsymbol{x} \in \mathcal{C}$.

Finally, we introduce an important lemma for our analysis, which appears in the proofs of the Frank-Wolfe algorithms (e.g., [46]).

Lemma 2.1. *Let* $\{\epsilon_t\}_{t\in\mathbb{N}} \subset \mathbb{R}$ *be a sequence such that*

$$\exists c > 0, \quad \forall t \in \mathbb{N}, \quad \epsilon_{t+1} \leq (1-\lambda_t)\epsilon_t + c\lambda_t^2, \tag{2.1}$$

where $\lambda_t = \frac{2}{t+2}$ *for all* $t \in \mathbb{N}$*. Then, the following holds.*

$$\epsilon_t \leq \frac{4c}{t+2} \quad \textit{for all } t = 1, 2, \ldots \tag{2.2}$$

Proof. We prove this lemma by induction on t. For the base case $t = 1$, using Eq. (2.1),

$$\epsilon_1 \leq \left(1 - \frac{2}{0+2}\right)\epsilon_0 + c\left(\frac{2}{0+2}\right)^2 = c \leq \frac{4c}{1+2}.$$

Now we move to the induction step. Assume that Eq. (2.2) holds for $t \geq 1$. Using Eq. (2.1) again, we have

$$\epsilon_{t+1} \leq \left(1 - \frac{2}{t+2}\right)\epsilon_t + c\left(\frac{2}{t+2}\right)^2 = \frac{4c}{t+2}\frac{t+1}{t+2} \leq \frac{4c}{t+2}\frac{t+2}{t+3} = \frac{4c}{t+3},$$

where at the second inequality, we use the fact $\frac{t+1}{t+2} \leq \frac{t+2}{t+3}$. Therefore, Eq. (2.2) holds for all $t = 1, 2, \ldots$, which concludes the proof. □

2.1 Convex analysis

Our algorithm and analysis highly depend on convex analysis, so we introduce some basic notations and properties. Basically, we follow the notation adopted the books by Borwein and Lewis [7] and Boyd and Vandenberghe [8]. For a function $f : \mathcal{C} \to (-\infty, +\infty]$, we denote its domain by

$$\operatorname{dom} f = \{\boldsymbol{x} \in \mathcal{C} \mid f(\boldsymbol{x}) < +\infty\}.$$

That is, $\operatorname{dom} f$ denotes the set of points whose value of f takes a finite value.

We say that a set $\mathcal{C} \subset \mathbb{R}^n$ is convex if the following holds.

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}, \quad \forall \lambda \in [0,1], \quad \lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y} \in \mathcal{C}$$

Similarly, we say that a function $f : \mathcal{C} \to \mathbb{R}$, defined over a convex set $\mathcal{C}$, is convex if the following holds.

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}, \quad \forall \lambda \in [0,1], \quad f(\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y})$$

If a function f is differentiable, we denote its gradient vector at $\boldsymbol{x}$ by $\nabla f(\boldsymbol{x})$. If f is convex, the following holds.

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}, \quad f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x})$$

That is, the tangent planes of f at any point are always below f. If f is not differentiable, we denote its sub-differential by $\partial f(\boldsymbol{x}) = \{\boldsymbol{g} \mid f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{g} \cdot (\boldsymbol{y} - \boldsymbol{x}), \forall \boldsymbol{y}\}$. We call each element $\boldsymbol{g} \in \partial f(\boldsymbol{x})$ as a sub-gradient of f at $\boldsymbol{x}$. If f is twice differentiable, we denote its Hessian at $\boldsymbol{x}$ by $\nabla^2 f(\boldsymbol{x})$.

Definition 2.1 (Strong convexity)**.** *A differentiable function $f : \mathcal{C} \to \mathbb{R}$ is said to be α-strongly convex w.r.t. ℓ_p-norm over $\mathcal{C}$ if*

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{C},\ \forall \alpha \in [0, 1], \quad f(\alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{y}) \leq \alpha f(\boldsymbol{x}) + (1-\alpha) f(\boldsymbol{y}) - \frac{1}{2}\alpha(1-\alpha)\|\boldsymbol{x} - \boldsymbol{y}\|_p^2.$$

If an α-strongly convex function f is differentiable, the above definition is equivalent to the following one.

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}, \quad f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x}) + \frac{\alpha}{2}\|\boldsymbol{y} - \boldsymbol{x}\|_p^2.$$

Furthermore, if f is twice differentiable, the above is equivalent to $\nabla^2 f(\boldsymbol{x}) \succeq \alpha I$ for all $\boldsymbol{x} \in \mathcal{C}$.

Definition 2.2 (Smoothness)**.** *A differentiable function $f : \mathcal{C} \to \mathbb{R}$ is said to be β-smooth w.r.t. ℓ_p-norm over $\mathcal{C}$ if*

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}, \quad f(\boldsymbol{y}) \leq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x}) + \frac{\beta}{2}\|\boldsymbol{y} - \boldsymbol{x}\|_p^2.$$

If f is twice differentiable, the above is equivalent to $\nabla^2 f(\boldsymbol{x}) \preceq \beta I$ for all $\boldsymbol{x} \in \mathcal{C}$.

The strong convexity of the entropy function is a central property for our analysis.

The relative entropy function $D_{\mathrm{KL}} : \Delta_m \times \Delta_m \to \mathbb{R}_+ \cup$ is defined by

$$D_{\mathrm{KL}}(\boldsymbol{p} \| \boldsymbol{q}) = \sum_{i=1}^{m} p_i \ln \frac{p_i}{q_i}.$$

Here, we define $p_i \ln \frac{p_i}{q_i} = 0$ if $p_i = 0$, $p_i \ln \frac{p_i}{q_i} = +\infty$ if $p_i \neq 0$ and $q_i = 0$. The following Lemma shows strong convexity of the relative entropy function from the uniform distribution.

Lemma 2.2 (Example 2.5 in [87]). *Let $R(\boldsymbol{d}) = \sum_{i=1}^m d_i \ln \frac{d_i}{1/m}$ be the relative entropy function from the uniform distribution $\frac{1}{m}\mathbf{1} \in \Delta_m$. Then, R is 1-strongly convex w.r.t. ℓ_1-norm over Δ_m.*

Proof. By definition of R, for any $\boldsymbol{p}, \boldsymbol{q} \in \Delta_m$ and for any $\alpha \in [0,1]$,

$$\begin{aligned}
&\alpha R(\boldsymbol{p}) + (1-\alpha)R(\boldsymbol{q}) - R(\alpha\boldsymbol{p} + (1-\alpha)\boldsymbol{q}) \\
&= \alpha \sum_{i=1}^m p_i \ln p_i + (1-\alpha) \sum_{i=1}^m q_i \ln q_i - \sum_{i=1}^m (\alpha p_i + (1-\alpha)q_i) \ln(\alpha p_i + (1-\alpha)q_i) \\
&= \alpha \sum_{i=1}^m p_i \ln \frac{p_i}{\alpha p_i + (1-\alpha)q_i} + (1-\alpha) \sum_{i=1}^m q_i \ln \frac{q_i}{\alpha p_i + (1-\alpha)q_i} \\
&= \alpha D_{\mathrm{KL}}(\boldsymbol{p} \| \alpha\boldsymbol{p} + (1-\alpha)\boldsymbol{q}) + (1-\alpha) D_{\mathrm{KL}}(\boldsymbol{q} \| \alpha\boldsymbol{p} + (1-\alpha)\boldsymbol{q})
\end{aligned}$$

Using the Pinsker's inequality, $D_{\mathrm{KL}}(\boldsymbol{p}\|\boldsymbol{q}) \geq \frac{1}{2}\|\boldsymbol{p} - \boldsymbol{q}\|_1^2$ for all $\boldsymbol{p}, \boldsymbol{q} \in \Delta_m$, we have

$$\begin{aligned}
&\alpha R(\boldsymbol{p}) + (1-\alpha)R(\boldsymbol{q}) - R(\alpha\boldsymbol{p} + (1-\alpha)\boldsymbol{q}) \\
&\geq \alpha \frac{1}{2} \left\| \boldsymbol{p} - (\alpha\boldsymbol{p} + (1-\alpha)\boldsymbol{q}) \right\|_1^2 + (1-\alpha) \frac{1}{2} \left\| \boldsymbol{q} - (\alpha\boldsymbol{p} + (1-\alpha)\boldsymbol{q}) \right\|_1^2 \\
&= \alpha(1-\alpha) \|\boldsymbol{p} - \boldsymbol{q}\|_1^2,
\end{aligned}$$

which implies the 1-strong convexity of R. □

The following lemma is helpful to make a convex function to be a strongly convex one.

Lemma 2.3. *Let f be a convex function and let g be an α-strongly convex function. Then $f + g$ is also an α-strongly convex function.*

Proof. The result follows from the definition. For all $\boldsymbol{x}$ and $\boldsymbol{y}$ and for all $\alpha \in [0,1]$,

$$\begin{aligned}
&(f+g)(\alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y}) \\
&= f(\alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y}) + g(\alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y}) \\
&\leq \alpha f(\boldsymbol{x}) + (1-\alpha) f(\boldsymbol{y}) + \alpha g(\boldsymbol{x}) + (1-\alpha) g(\boldsymbol{y}) - \frac{1}{2}\alpha(1-\alpha)\|\boldsymbol{x} - \boldsymbol{y}\|^2 \\
&= \alpha(f+g)(\boldsymbol{x}) + (1-\alpha)(f+g)(\boldsymbol{y}) - \frac{1}{2}\alpha(1-\alpha)\|\boldsymbol{x} - \boldsymbol{y}\|^2
\end{aligned}$$

which is the α-strong convexity of $f + g$. □

Definition 2.3 (Fenchel conjugate). *The Fenchel conjugate $f^\star : \mathbb{R}^n \to [-\infty, +\infty]$ of a function $f : \mathbb{R}^n \to [-\infty, +\infty]$ is defined as*

$$f^\star(\boldsymbol{y}) = \sup_{\boldsymbol{x} \in \mathbb{R}^n} \boldsymbol{y} \cdot \boldsymbol{x} - f(\boldsymbol{x}).$$

Table 2.1: Summary of notations.

$\mathbb{R}$	The set of real numbers
$\mathbb{N}$	The set of natural numbers
Δ_n	n-dimensional probability simplex
$[n]$	The set of positive integers up to n
$\mathcal{C}, \mathcal{D}$	Sets
x, θ	Scalars
$\boldsymbol{x}, \boldsymbol{\theta}$	Vectors
A, M	Matrices
$\boldsymbol{e}_j$	The jth canonical basis vector
$\mathbf{1}_n, \mathbf{1}$	n-dimensional all 1 vector
$f^\star$	The Fenchel conjugate of f

The Fenchel conjugate of the max function $\boldsymbol{x} \mapsto \max_{i\in[n]} x_i$ is the indicator function $\mathbb{I}_{\Delta_n}$ of the probability simplex and vice versa. This relation is used throughout our thesis.

By definition, the following is derived as observations.

Observation 2.1. *Let $f, g : \mathbb{R}^m \to (-\infty, +\infty]$ be functions such that*

$$\exists c > 0, \quad \forall \boldsymbol{x}, \quad f(\boldsymbol{x}) \leq g(\boldsymbol{x}) \leq f(\boldsymbol{x}) + c.$$

Then, $f^\star(\boldsymbol{y}) - c \leq g^\star(\boldsymbol{y}) \leq f^\star(\boldsymbol{y})$ holds for all $\boldsymbol{y}$.

The following lemma shows a relation between f and its conjugate $f^\star$. This lemma is shown in the Ph.D. thesis of Shalev-Shwartz [86].

Lemma 2.4. *Let $\|\cdot\|$ be some norm and let $\|\cdot\|_\star$ be the dual norm of $\|\cdot\|$. If $f : \mathbb{R}^m \to \mathbb{R}$ is an α-strongly convex function w.r.t. the norm $\|\cdot\|$, its Fenchel conjugate $f^\star$ is an $1/\alpha$-smooth function w.r.t. the dual norm $\|\cdot\|_\star$.*

The following theorem is a central tool for our analysis.

Theorem 2.1 (Fenchel duality theorem [7])**.** *Let $f : \mathbb{R}^m \to (-\infty, +\infty]$ and $g : \mathbb{R}^n \to (-\infty, +\infty]$ be convex functions, and a linear map $A : \mathbb{R}^m \to \mathbb{R}^n$. Define the Fenchel problems*

$$\gamma = \inf_{\boldsymbol{x}} f(\boldsymbol{x}) + g(A^\top \boldsymbol{x}), \qquad \rho = \sup_{\boldsymbol{y}} -f^\star(-A\boldsymbol{y}) - g^\star(\boldsymbol{y}). \tag{2.3}$$

Input: A set of training examples $S = \{(\boldsymbol{x}_1, y_i), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\} \subset \mathcal{X} \times \mathcal{Y}$.
For $t = 1, 2, \ldots, T$:

1. Booster chooses a distribution $\boldsymbol{d}_t \in \Delta_m$ over S.
2. Weak Learner returns a hypothesis $h_t \in \mathcal{H}$.

Booster defines the weights $\{w_1, w_2, \ldots, w_T\}$ on the hypotheses $\{h_1, h_2, \ldots, h_T\}$.
Output: A convex combination $\sum_{t=1}^{T} w_t h_t$ of hypotheses $\{h_1, h_2, \ldots, h_T\} \subset \mathcal{H}$.

Figure 2.1: The boosting protocol.

Then, $\gamma \geq \rho$ holds. Further, $\gamma = \rho$ holds if $\mathbf{0} \in \operatorname{core}\left(\operatorname{dom} g - A^\top \operatorname{dom} f\right)$. Furthermore, points $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{y}}$ are optimal solutions for problems in Eq. (2.3) *respectively if and only if $-A\bar{\boldsymbol{y}} \in \partial f(\bar{\boldsymbol{x}})$ and $\bar{\boldsymbol{y}} \in \partial g(A^\top \bar{\boldsymbol{x}})$.*

Table 2.1 summarizes important notations.

2.2 Boosting

Boosting is known as a class of algorithms for supervised learning. Boosting is a protocol between two algorithms: the booster and the weak learner. For each iteration $t = 0, 1, 2, \ldots, T$, the booster chooses a distribution $\boldsymbol{d}_t \in \Delta_m$ over the training examples $S = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\} \subset \mathcal{X} \times \mathcal{Y}$. Here, $\mathcal{X}$ is some set we do not need to care about, and $\mathcal{Y}$ is the set of target values that depends on the problem. Then, the weak learner returns a hypothesis $h_t \in \mathcal{H}$ to the booster from some pre-defined set of hypotheses $\mathcal{H}$ from $\mathcal{X}$ to $\mathcal{Y}$. Throughout this thesis, we assume that $\mathcal{H}$ is finite. This assumption is reasonable since most practical hypothesis sets, such as decision trees, are induced to be a finite set throughout training examples even though it is not. The boosting algorithm aims to produce a convex combination $H_T = \sum_{t=1}^{T} w_t h_t$ of the hypotheses $\{h_1, h_2, \ldots, h_T\} \subset \mathcal{H}$ that achieves a goal. Here, we note that some boosting algorithms, such as Graph Separation Boosting [2], consider other compositions $H_T = f(h_1, h_2, \ldots, h_T)$ with an aggregation rule f rather than a convex combination of functions. But this setting is beyond the scope of our thesis.

For the binary classification setting, i.e., $\mathcal{Y} = \{-1, +1\}$, one of the goal is to minimize a surrogate loss $\sum_{i=1}^{m} e^{-y_i H_T(\boldsymbol{x}_i)}$. This goal is used for AdaBoost [28], the most well-known

Input: Initial point $\boldsymbol{x}_1 \in \mathcal{C}$ and an accuracy parameter $\epsilon > 0$.
For $t = 1, 2, \ldots, T$:

1. Observe a gradient $\nabla f(\boldsymbol{x}_t)$ at $\boldsymbol{x}_t \in \mathcal{C}$.
2. Find an extreme point $\boldsymbol{s}_t \in \arg\min_{\boldsymbol{s} \in \mathcal{C}} \boldsymbol{s} \cdot \nabla f(\boldsymbol{x}_t)$.
3. If $(\boldsymbol{x}_t - \boldsymbol{s}_t)^\top \nabla f(\boldsymbol{x}_t) \leq \epsilon$ then set $T = t - 1$ and **break**.
4. Choose a next iterate $\boldsymbol{x}_{t+1} \in \mathcal{C}$.

Output: $\boldsymbol{x}_{T+1} \in \mathcal{C}$ such that $f(\boldsymbol{x}_{T+1}) - \min_{\boldsymbol{x} \in \mathcal{C}} f(\boldsymbol{x}) \leq \epsilon$.

Figure 2.2: The Frank-Wolfe protocol.

boosting algorithm. Chapter 3 adopts the soft margin maximization as the goal, so we describe it in Section 3.1 for details.

For the regression setting, i.e., $\mathcal{Y} = \mathbb{R}$, minimizing the squared loss $\sum_{i=1}^{m} (y_i - H_T(\boldsymbol{x}_i))^2$ is a common goal. Chapter 4 considers a different goal and analyze the convergence rate of the algorithm for this setting, we explain this goal in Section 4.1.

Boosting is also applicable for other settings such as Multi-class classification [28, 82], Ranking [27], and so on. These algorithms are beyond the scope of this thesis, so we do not describe them. See [67, 80] for details.

Fig. 2.1 summarizes the boosting protocol.

2.3 The Frank-Wolfe algorithms

The Frank-Wolfe (FW) method is a class of optimization algorithms that only uses first-order information of the objective and a linear optimization oracle over the feasible region. The best advantage of the FW algorithm is the projection-free property; there is no projection onto $\mathcal{C}$, so the running time per iteration is faster than the projected gradient methods. The original FW algorithm is a first-order iterative algorithm invented by Frank and Wolfe [26]. The FW algorithm solves the problems of the form: $\min_{\boldsymbol{x} \in \mathcal{C}} f(\boldsymbol{x})$, where $\mathcal{C} \subset \mathbb{R}^m$ is a closed convex set and $f : \mathcal{C} \to \mathbb{R}^m$ is an η-smooth and convex function.

In each iteration $t = 1, 2, \ldots$, the FW algorithm seeks an extreme point $\boldsymbol{s}_t \in \mathcal{C}$ that minimizes the inner product $\boldsymbol{s}_t^\top \nabla f(\boldsymbol{x}_t)$. Then, it updates the iterate as $\boldsymbol{x}_{t+1} =$

$\boldsymbol{x}_t + \lambda_t(\boldsymbol{s}_t - \boldsymbol{x}_t)$ for some $\lambda_t \in [0, 1]$. Although the classical result [26, 46] suggests $\lambda_t = 2/(t+1)$, one can choose other step size like

$$\lambda_t := \text{clip}\ \frac{(\boldsymbol{x}_t - \boldsymbol{s}_{t+1})^\top \nabla f(\boldsymbol{x}_t)}{\eta \|\boldsymbol{s}_{t+1} - \boldsymbol{x}_t\|^2}, \tag{2.4}$$

where clip $x = \max\{0, \min\{1, x\}\}$. This λ_t minimizes the right-hand-side of the smoothness relation

$$f(\boldsymbol{x}_{t+1}) = f(\boldsymbol{x}_t + \lambda_t(\boldsymbol{s}_{t+1} - \boldsymbol{x}_t)) \leq f(\boldsymbol{x}_t) + \lambda_t \nabla f(\boldsymbol{x}_t) \cdot (\boldsymbol{s}_{t+1} - \boldsymbol{x}_t) + \frac{\eta}{2}\lambda_t^2 \|\boldsymbol{s}_{t+1} - \boldsymbol{x}_t\|^2,$$

and is often called the *short-step* strategy. Alternatively, one can choose

$$\lambda_t \in \arg\min_{\lambda \in [0,1]} f(\boldsymbol{x}_t + \lambda(\boldsymbol{s}_{t+1} - \boldsymbol{x}_t))$$

by line search. This step size improve the objective more than the short-step strategy. Since the FW algorithm aims to find an optimal solution, one can choose $\boldsymbol{x}_{t+1}$ by solving the problem: $\boldsymbol{x}_{t+1} \leftarrow \arg\min_{\boldsymbol{x} \in \text{ConvHull}(\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{t+1}\})} f(\boldsymbol{x})$. This update rule is called the *Fully Corrective* FW algorithm (e.g., [46]). Although the fully corrective update yields $\boldsymbol{x}_{t+1}$ that most decreases the objective over the convex hull, it loses the fast computational advantage per iteration. One can find other updated rules developed so far, such as Away-step, Pairwise, and Blended Pairwise rules, but this topic is outside the scope of our thesis, so please check the papers [19, 45, 54, 71, 94]. Fig. 2.2 summarizes the Frank-Wolfe protocol. Line 3 measures the optimality gap. By the optimality of $\boldsymbol{s}_{t+1}$ and the convexity of f, the following holds for all $\boldsymbol{x}^\star \in \mathcal{C}$.

$$(\boldsymbol{x}_t - \boldsymbol{s}_{t+1})^\top \nabla f(\boldsymbol{x}_t) \geq (\boldsymbol{x}_t - \boldsymbol{x}^\star)^\top \nabla f(\boldsymbol{x}_t) \geq f(\boldsymbol{x}_t) - f(\boldsymbol{x}^\star)$$

Thus, if the optimality gap satisfy $(\boldsymbol{x}_t - \boldsymbol{s}_{t+1})^\top \nabla f(\boldsymbol{x}_t) \leq \epsilon$, then $f(\boldsymbol{x}_t) - \min_{\boldsymbol{x}^\star \in \mathcal{C}} f(\boldsymbol{x}^\star) \leq \epsilon$ holds.

Most of the FW algorithms converge to an ϵ-approximate solution in $O(\eta/\epsilon)$ iterations if the objective function is η-smooth[1] w.r.t. some norm over $\mathcal{C}$ [26, 46]. This bound is optimal for non-strongly convex but smooth objective since some paper showed $\Omega(1/t^{1+\delta})$ for arbitrarily small $\delta > 0$ [11, 102]. On the other hand, some FW-type algorithms converge linearly when the objective function is not only smooth but also strongly convex [54, 71, 94].

[1]Some papers, such as Jiang and Ravikumar [77], Hazan and Kale [39], and Hazan and Minasyan [40], suggest techniques for general convex functions.

Chapter 3

Boosting as Frank-Wolfe

This chapter introduces the relationship between some boosting algorithms for soft margin optimization and the Frank-Wolfe algorithms. After that, we propose a boosting algorithm that optimizes the soft margin while maintaining a convergence guarantee.

Theory and algorithms for large-margin classifiers have been studied extensively since those classifiers guarantee low generalization errors when they have large margins over training examples (e.g., [81, 67]). In particular, the ℓ_1-norm regularized soft margin optimization problem, defined later, is a formulation of finding sparse large-margin classifiers based on the linear program (LP). This problem aims to optimize the ℓ_1-margin by combining multiple hypotheses from some hypothesis class $\mathcal{H}$. The resulting classifier tends to be sparse, so ℓ_1-margin optimization is helpful for feature selection tasks. Off-the-shelf LP solvers can solve the problem, but they are still not efficient enough for a huge class $\mathcal{H}$.

Boosting is a framework for solving the ℓ_1-norm regularized margin optimization even though $\mathcal{H}$ is infinitely large. Various boosting algorithms have been invented. LPBoost [21] is a practical algorithm that often works effectively. Although LPBoost terminates rapidly, It is shown that it takes $\Omega(m)$ iterations in the worst case, where m is the number of training examples [97]. Shalev-Shwartz and Singer [88] invented an algorithm called Corrective ERLPBoost (we call this algorithm Corr. ERLPBoost for shorthand) in the paper on ERLPBoost [99]. C-ERLPBooost and ERLPBoost find ϵ-approximate solutions in $O(\ln(m/\nu)/\epsilon^2)$ iterations, where $\nu \in [1, m]$ is the soft margin parameter. The difference is the time complexity per iteration; ERLPBoost solves a convex program (CP) for each iteration, while C-ERLPBooost solves a sorting-like problem. Although ERLPBoost takes much time per iteration, it takes fewer iterations than Corr. ERLPBoost in

practical applications. For this reason, ERLPBoost is faster than Corr. ERLPBoost. Our primary motivation is to investigate boosting algorithms with provable iteration bounds, which perform as fast as LPBoost.

This paper has two contributions. Our first contribution is to give a unified view of boosting for soft margin optimization. We show that LPBoost, ERLPBoost, and Corr. ERLPBoost are instances of the Frank-Wolfe algorithm.

Our second contribution is to propose a generic scheme for boosting based on the unified view. Our scheme combines a standard Frank-Wolfe algorithm and *any* algorithm and switches one to the other at each iteration in a non-trivial way. We show that this scheme guarantees the same convergence rate, $O(\ln(m/\nu)/\epsilon^2)$, as ERLPBoost and Corr. ERLPBoost. One can incorporate any update rule to this scheme without losing the convergence guarantee so that it takes advantage of better updates of the second algorithm in practice. In particular, we propose to choose LPBoost as the secondary algorithm, and we call the resulting algorithm Modified LPBoost (MLPBoost).

In experiments on real datasets, MLPBoost works comparably with LPBoost, and MLPBoost is the fastest among theoretically guaranteed algorithms, as expected.

The results in this chapter appear in [64].

3.1 Soft margin optimization

Most boosting algorithms aim to find a convex combination $H = \operatorname{sgn} \circ \sum_{h \in \mathcal{H}} w_h h$ of hypotheses in $\mathcal{H} \subset [-1,+1]^{\mathcal{X}}$ that minimizes the empirical losses $\sum_{i=1}^m L(y_i, H(\boldsymbol{x}_i))$, where $\operatorname{sgn} : \mathbb{R} \to \{-1,+1\}$ is the function such that $\operatorname{sgn}(\theta) = +1$ if and only if $\theta \geq 0$. It is well known that the generalization error of h is

$$\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}[L(y,h(\boldsymbol{x}))] = \frac{1}{m}\sum_{i=1}^m L(y_i,h(\boldsymbol{x}_i)) + O\left(\frac{1}{\sqrt{m}}\right), \quad \text{w.h.p.},$$

where $\mathcal{P}$ is a distribution over $\mathcal{X} \times \{\pm 1\}$ such that $S = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\} \sim \mathcal{P}^m$. Therefore, minimizing the empirical loss is a reasonable goal for machine learning. AdaBoost [28], known to minimize the exponential loss $(\boldsymbol{x}, y) \mapsto e^{-yh(\boldsymbol{x})}$, decreases its test error by collecting hypotheses in $\mathcal{H}$ even after zero training loss is achieved. This phenomenon has been studied extensively [9, 81] and it has been found that AdaBoost tends to yield a large margin hypothesis. Here, the margin of a point $(\boldsymbol{x}, y) \in \mathcal{X} \times \{\pm 1\}$ for a hypothesis $h : \mathcal{X} \to [-1,+1]$ is defined by $\rho_h(\boldsymbol{x}, y) = yh(\boldsymbol{x})$ and the margin of a sample

$S = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\} \subset \mathcal{X} \times \{\pm 1\}$ for h is $\rho^\star = \min_{i \in [m]} \rho_h(\boldsymbol{x}_i, y_i)$. If a hypothesis h achieves its minimum margin $\rho^\star \geq \theta$ for some $\theta > 0$, a better generalization ability is attained. That is, if h has margin at least θ for all points in S, the following relation holds [81].

$$\Pr_{(\boldsymbol{x},y)\sim\mathcal{P}}[y \neq h(\boldsymbol{x})] = \Pr_{i\sim\mathcal{U}_{[m]}}[\rho_h(\boldsymbol{x}_i, y_i) \leq \theta] + O\left(\frac{1}{\theta}\sqrt{\frac{\ln m}{m}}\right), \quad \text{w.h.p.},$$

where $\mathcal{U}_{[m]}$ is the uniform distribution over $[m]$. Both terms on the right-hand side become small if the margin parameter θ is large. So, one wants to maximize the margin of a sample S. Although many other margin-based bounds exist [9, 32], a large margin still implies a better generalization guarantee[1]. Finding a large margin classifier can be written as a linear program.

$$\max_{\boldsymbol{w}} \min_{i\in[m]} y_i \sum_{h\in\mathcal{H}} w_h h(\boldsymbol{x}_i) \quad \text{s.t.} \quad \boldsymbol{w} \in \Delta_{\mathcal{H}}$$

This optimization problem is called the hard margin optimization since the formulation is valid if there exists $\boldsymbol{w} \in \Delta_{\mathcal{H}}$ satisfying hypothesis $\boldsymbol{x} \mapsto \sum_{h\in\mathcal{H}} w_h h(\boldsymbol{x})$ perfectly classifies all training examples. Unfortunately, the margin of the output hypothesis of AdaBoost is known to asymptotically optimizes the margin (e.g., [76, 78]). With these issues, several algorithms for margin-maximizing boosting have been developed. AdaBoost* [76], TotalBoost [101], and SparsiBoost [59] are the most successful ones that maximize the hard margin. If a convex combination $\sum_{h\in\mathcal{H}} w_h h$ exists that perfectly classifies all training points, these algorithms guarantee the finding of a combined hypothesis whose margin is the largest in polynomial time. Since linear separability does not generally hold, the relaxed version of margin optimization, named soft margin optimization is considered [21, 75].

$$\begin{aligned} \max_{\rho,\boldsymbol{w},\boldsymbol{\xi}} \quad & \rho - \frac{1}{\nu}\sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad & y_i \sum_{h\in\mathcal{H}} w_h h(\boldsymbol{x}_i) \geq \rho - \xi_i, \quad \forall i \in [m], \\ & \boldsymbol{w} \in \Delta_{\mathcal{H}}, \quad \boldsymbol{\xi} \geq \boldsymbol{0}. \end{aligned}$$

Intuitively, the soft margin optimization aims to find a convex combination of hypotheses that maximizes the margin for most points. The rest points are regarded as outliers. In this sense, soft margin optimization is robust for outliers. The parameter $\nu \in [1, m]$

[1] Lower bounds are also studied, see for example [35].

controls the ratio of outliers. That is, the optimal $\boldsymbol{w}$ regards at most ν points as outliers. Similar formulation is found in Support Vector Machines [83]. Therefore, an algorithm that optimizes the soft-margin is applicable even though a convex combination of hypotheses that perfectly classifies the training points does not exist.

In order to analyze the soft margin boosting algorithms, we need to define the weak learnability. To define the weak learnability for margin optimization, we consider the dual problem of the soft margin optimization. For notational simplicity, let $A \in [-1, +1]^{m \times \mathcal{H}}$ be the matrix such that $A_{i,h} = y_i h(\boldsymbol{x}_i)$. Then, the soft margin optimization can be written as

$$\begin{aligned} \max_{\rho, \boldsymbol{w}, \boldsymbol{\xi}} \quad & \rho - \frac{1}{\nu} \sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad & (A\boldsymbol{w})_i \geq \rho - \xi_i, \quad \forall i \in [m], \\ & \boldsymbol{w} \in \Delta_{\mathcal{H}}, \quad \boldsymbol{\xi} \geq \boldsymbol{0}. \end{aligned}$$

The edge minimization problem, the Lagrange dual problem of the soft margin optimization, is

$$\min_{\boldsymbol{d}} \max_{h \in \mathcal{H}} (\boldsymbol{d}^\top A)_h + \mathbb{I}_{\Delta_{m,\nu}}(\boldsymbol{d}), \tag{3.1}$$

where $\Delta_{m,\nu} = \{\boldsymbol{d} \in \Delta_m \mid \|\boldsymbol{d}\|_\infty \leq 1/\nu\}$. The quantity $(\boldsymbol{d}^\top A)_h = \sum_{i=1}^m d_i y_i h(\boldsymbol{x}_i)$ is often called the *edge* of the hypothesis h w.r.t. the distribution $\boldsymbol{d} \in \Delta_{m,\nu}$. Shalev-Shwartz and Singer [88] showed that one can rewrite the soft margin optimization problem in a much simpler form via Theorem 2.1:

$$\max_{\boldsymbol{w} \in \Delta_{\mathcal{H}}} -\mathbb{I}^\star_{\Delta_{m,\nu}}(-A\boldsymbol{w}) = \max_{\boldsymbol{w} \in \Delta_{\mathcal{H}}} \min_{\boldsymbol{d} \in \Delta_{m,\nu}} \boldsymbol{d}^\top A \boldsymbol{w}. \tag{3.2}$$

Furthermore, the duality gap between Eq. (3.1) and Eq. (3.2) is zero. The soft margin optimization aims to find an optimal combined hypothesis $\sum_{h \in \mathcal{H}} \bar{w}_h h$, where $\bar{\boldsymbol{w}} \in \Delta_{\mathcal{H}}$ is an optimal solution of Eq. (3.2). Although the edge minimization and soft margin optimization problems are formulated as a linear program, solving the problem for a huge class $\mathcal{H}$ is hard. Boosting is a standard approach to dealing with the problem.

To analyze a boosting algorithm, one needs to assume the performance of the weak learner.

Assumption 3.1 (The g-weak learnability for margin optimization)**.** *Let $g > 0$ be a fixed but unknown parameter. A g-weak learner takes a distribution $\boldsymbol{d} \in \Delta_m$ over training examples and returns a hypothesis $h \in \mathcal{H}$ such that $(\boldsymbol{d}^\top A)_h = \sum_{i=1}^m d_i y_i h(\boldsymbol{x}_i) \geq g$.*

Table 3.1: Soft margin boosting algorithms their upper and lower bound of the number of weak learner call. The column Sub-problem per round shows the problem that the boosting algorithms solve. m is the number of training examples, ϵ is the accuracy parameter for Eq. (3.3), and $\nu \in [1, m]$ is the capping parameter.

Algorithm	Upper bound	Lower bound	Sub-problem per round
LPBoost [21]	-	$\Omega(m)$	Linear program (LP)
C-ERLPBoost [88]	$O(\ln(m/\nu)/\epsilon^2)$	-	Sorting
SoftBoost [97]	$O(\ln(m/\nu)/\epsilon^2)$	-	Convex program (CP)
ERLPBoost [99]	$O(\ln(m/\nu)/\epsilon^2)$	-	Convex program (CP)

Here, we emphasize that the parameter g is unknown to the boosting algorithm. With this assumption, the goal of booster is to find a solution $\boldsymbol{w} \in \Delta_{\mathcal{H}}$ such that

$$\min_{\boldsymbol{d} \in \Delta_{m,\nu}} \boldsymbol{d}^\top A \boldsymbol{w} \geq g - \epsilon \tag{3.3}$$

for arbitrarily small $\epsilon > 0$. Note that if the weak learner returns a hypothesis $h \in \mathcal{H}$ that maximizes the edge $(\boldsymbol{d}^\top A)_h$, the above goal becomes to find an ϵ-approximate solution of Eq. (3.2).

Find $\bar{\boldsymbol{w}} \in \Delta_{\mathcal{H}}$ such that

$$-\mathbb{I}^\star_{\Delta_{m,\nu}}(-A\bar{\boldsymbol{w}}) = \min_{\boldsymbol{d} \in \Delta_{m,\nu}} \boldsymbol{d}^\top A \bar{\boldsymbol{w}} \geq \max_{\boldsymbol{w} \in \Delta_{\mathcal{H}}} \min_{\boldsymbol{d} \in \Delta_{m,\nu}} \boldsymbol{d}^\top A \boldsymbol{w} - \epsilon = \max_{\boldsymbol{w} \in \Delta_{\mathcal{H}}} -\mathbb{I}^\star_{\Delta_{m,\nu}}(-A\boldsymbol{w}) - \epsilon.$$

Therefore, this chapter aims to find a convex combination $\sum_{h \in \mathcal{H}} w_h h$ of hypotheses in $\mathcal{H}$ satisfying Eq. (3.3).

3.2 Related work

FWBoost [96] seems to be related to our work. FWBoost is a boosting algorithm designed for minimizing a general loss function by the Frank-Wolfe algorithm. Since Frank-Wolfe algorithms work over the closed convex set, they introduce the ℓ_1-norm ball constraint. Note that the objective function for the maximization problem in Eq. (3.2) is not a smooth function, so one cannot apply the FWBoost directly to guarantee the convergence rate.

Many boosting algorithms that maximizes the soft margin have been invented [22, 85],

but here we introduce three important algorithms, LPBoost [21], ERLPBoost [99], and Corrective ERLPBoost [88].

LPBoost [21] is a practical boosting algorithm for solving problem in Eq. (3.2). In each iteration t, LPBoost updates its distribution as an optimal solution to problem

$$\boldsymbol{d}_t \leftarrow \arg\min_{\boldsymbol{d}} \max_{k\in[t]} (\boldsymbol{d}^\top A)_{h_k} + \mathbb{I}_{\Delta_{m,\nu}}(\boldsymbol{d}). \tag{3.4}$$

That is, LPBoost uses an optimal solution to the edge minimization problem over the hypothesis set $\{h_1, h_2, \ldots, h_t\} \subset \mathcal{H}$. LPBoost converges to an ϵ-accurate solution rapidly in practice. However, Warmuth et al. [97] proved that LPBoost converges in $\Omega(m)$ iterations for the worst case. In the same paper, SoftBoost [97], the soft margin version of TotalBoost [101], is invented. SoftBoost converges in $O(\ln(m/\nu)/\epsilon^2)$ iterations, but it tends to be too conservative. Thus, SoftBoost is not practical because it takes time. After that, the stabilized version of LPBoost, ERLPBoost, was invented by Warmuth et al. [99]. The main idea behind ERLPBoost comes from the bundle method [91]. Bundle methods [51, 55, 84] are stabilized versions of the cutting-plane method [50]. The cutting-plane method is known to cause a zig-zag phenomenon, and thus, it converges very slowly in some situations [3, 55]. The cutting-plane method corresponds to LPBoost in this case. Therefore, ERLPBoost can be seen as a stabilized version of LPBoost, i.e., an instance of the bundle method. ERLPBoost updates the distribution as the solution of

$$\boldsymbol{d}_t \leftarrow \arg\min_{\boldsymbol{d}} \max_{k\in[t]} (\boldsymbol{d}^\top A)_{h_k} + \mathbb{I}_{\Delta_{m,\nu}}(\boldsymbol{d}) + \frac{1}{\eta} R(\boldsymbol{d}). \tag{3.5}$$

Here, $R(\boldsymbol{d}) = \sum_{i=1}^m d_i \ln d_i + \ln m$ is the relative entropy from the uniform distribution $\frac{1}{m}\mathbf{1} \in \Delta_{m,\nu}$. They proved that ERLPBoost finds a solution that achieves Eq. (3.3) in $O(\ln(m/\nu)/\epsilon^2)$ iterations. They also demonstrate that ERLPBoost tends to terminate in fewer iterations than LPBoost. The disadvantage of ERLPBoost is its computational complexity; ERLPBoost solves convex programs in each iteration. This disadvantage leads to much more computation time than LPBoost. The Corrective ERLPBoost (Corr. ERLPBoost, for short) [88] is the corrective version of ERLPBoost. This algorithm achieves the same iteration bound with much faster computation per iteration than LPBoost. Corr. ERLPBoost maintains the weight $\boldsymbol{w}_t \in \Delta_{\mathcal{H}}$ that only has non-zero values on the entries corresponding to the past hypotheses $\{h_1, h_2, \ldots, h_t\} \subset \mathcal{H}$. Corr. ERLPBoost updates its distribution over the training instances as

$$\boldsymbol{d}_t \leftarrow \arg\min_{\boldsymbol{d}} \boldsymbol{d}^\top A \boldsymbol{w}_t + \mathbb{I}_{\Delta_{m,\nu}}(\boldsymbol{d}) + \frac{1}{\eta} R(\boldsymbol{d}). \tag{3.6}$$

Unlike LPBoost and ERLPBoost, Corr. ERLPBoost maintains weights on the hypotheses attained until the current round. After receiving a hypothesis $h_{t+1} \in \mathcal{H}$ with the corresponding basis $\boldsymbol{e}_{h+1} \in \Delta_{\mathcal{H}}$, Corr. ERLPBoost updates the weights on hypotheses as $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \lambda(\boldsymbol{e}_{h_{t+1}} - \boldsymbol{w}_t)$, where $\lambda_t \in [0, 1]$ is some proper value. Although this update rule seems to be a convex program, Shalev-Shwartz and Singer [88] showed an algorithm that solves Eq. (3.6) in $O(m \ln m)$ time. This algorithm seems better than LPBoost and ERLPBoost. However, Warmuth et al. [99] demonstrated that Corr. ERLPBoost takes much more iterations than LPBoost and ERLPBoost. Therefore, the overall computation time is worse than LPBoost. Table 3.1 summarizes the boosting algorithms for soft margin optimization.

To the best of our knowledge, there is only one previous work suggests a relation between a boosting algorithm and the Frank-Wolfe algorithm [18]. Their paper shows that AdaBoost [28] is an instance of the Frank-Wolfe algorithm. However, their interpretation is only for the primal problem, while ours comes from the primal-dual problems.

3.3 The relation between soft margin boostings and Frank-Wolfe algorithms

We first show a unified view of the boosting algorithms via Fenchel duality. From this view, LPBoost, ERLPBoost, and Corr. ERLPBoost can be seen as instances of the Frank-Wolfe algorithm with different step sizes and objectives. Using this knowledge, we derive a new boosting scheme.

3.3.1 A unified view of boosting for the soft margin optimization

This section assumes that the weak learner always returns a hypothesis $h \in \mathcal{H}$ that maximizes the edge w.r.t. the given distribution. We start by revisiting Corr. ERLPBoost. Recall that Corr. ERLPBoost (and ERLPBoost) aim to solve the convex program

$$\min_{\boldsymbol{d}} g(\boldsymbol{d}^\top A) + f^\star(\boldsymbol{d}), \tag{3.7}$$

where $f = \mathbb{I}_{\Delta_{m,\nu}} + \frac{1}{\eta} R$ and $g(\boldsymbol{\theta}) = \max_{h \in \mathcal{H}} \theta_h$. Since $\frac{1}{\eta} R$ is a $\frac{1}{\eta}$-strongly convex function w.r.t. ℓ_1-norm, so is f. By Fenchel duality, the dual problem is

$$\max_{\boldsymbol{w} \in \Delta_{\mathcal{H}}} -f^\star(-A\boldsymbol{w}) = -\min_{\boldsymbol{w} \in \Delta_{\mathcal{H}}} f^\star(-A\boldsymbol{w}) = -\min_{\boldsymbol{\theta} \in -A\Delta_{\mathcal{H}}} f^\star(\boldsymbol{\theta}), \tag{3.8}$$

where $-A\Delta_{\mathcal{H}} = \{-A\boldsymbol{d} \mid \boldsymbol{d} \in \Delta_{\mathcal{H}}\}$. Since $\operatorname{dom} g = \mathbb{R}^{\mathcal{H}}$ and $\operatorname{dom} f = \Delta_{m,\nu}$, $\operatorname{dom} g - A^\top \operatorname{dom} f = \{\boldsymbol{\theta} - A^\top \boldsymbol{d} \mid \boldsymbol{\theta} \in \mathbb{R}^{\mathcal{H}}, \boldsymbol{d} \in \Delta_{m,\nu}\} = \mathbb{R}^{\mathcal{H}}$. Obviously, $\mathbf{0} \in \operatorname{core}(\operatorname{dom} g - A^\top \operatorname{dom} f)$ so that the optimality gap is zero. Further, $f^\star$ is an η-smooth function w.r.t. ℓ_∞-norm. Thus, the soft margin optimization problem becomes a minimization problem of a smooth function.

In each iteration t, Corr. ERLPBoost updates the distribution $\boldsymbol{d}_t \in \Delta_{m,\nu}$ over examples as the optimal solution of Eq. (3.6). This computation corresponds to the gradient computation $\nabla f^\star(\theta_t)$, where $\theta_t = -A\boldsymbol{w}_t$. Then, obtain a basis vector $\boldsymbol{e}_{h_{t+1}} \in \Delta_{\mathcal{H}}$ corresponding to hypothesis $h_{h_{t+1}} \in \mathcal{H}$ that maximizes the edge; $h_{t+1} \in \arg\max_{h \in \mathcal{H}} (\boldsymbol{d}_t^\top A)_h$. We can write this calculation regarding the gradient of $f^\star$;

$$\arg\max_{\boldsymbol{e}_h : h \in \mathcal{H}} \boldsymbol{d}_t^\top A \boldsymbol{e}_h = \arg\min_{\boldsymbol{e}_h : h \in \mathcal{H}} (-A\boldsymbol{e}_h)^\top \nabla f^\star(\boldsymbol{\theta}_t) = \arg\min_{\boldsymbol{\theta} \in -A\Delta_{\mathcal{H}}} \boldsymbol{\theta}^\top \nabla f^\star(\boldsymbol{\theta}_t).$$

Thus, finding a hypothesis that maximizes edge corresponds to solving linear programming in the Frank-Wolfe algorithm. Further, Corr. ERLPBoost updates the weights as $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \lambda_t(\boldsymbol{e}_{h_{t+1}} - \boldsymbol{w}_t)$, where λ_t is the short-step[2]. From these observations, we can say that the Corr. ERLPBoost is an instance of the Frank-Wolfe algorithm. Since $f^\star$ is η-smooth, we can say that this algorithm converges in $O(\eta/\epsilon)$ iterations for a max-edge weak learner.

Similarly, we can say that LPBoost and ERLPBoost are instances of the Frank-Wolfe algorithm. Let $\mathcal{E}_t := \{\boldsymbol{e}_h \mid h \in \{h_1, h_2, \ldots, h_t\}\}$ be the set of basis vectors corresponding to the hypotheses up to round t. LPBoost and ERLPBoost update the distribution as the optimal solutions $\boldsymbol{d}_t^{\mathrm{L}} \in \partial \mathbb{I}^\star_{\Delta_{m,\nu}}(-A\boldsymbol{w}_t^{\mathrm{L}})$ and $\boldsymbol{d}_t^{\mathrm{E}} = \nabla f^\star(-A\boldsymbol{w}_t^{\mathrm{E}})$, where

$$\text{(LPBoost)} \quad \boldsymbol{w}_t^{\mathrm{L}} \leftarrow \arg\max_{\boldsymbol{w} \in \mathrm{ConvHull}(\mathcal{E}_t)} -\mathbb{I}^\star_{\Delta_{m,\nu}}(-A\boldsymbol{w}), \tag{3.9}$$

$$\text{(ERLPBoost)} \quad \boldsymbol{w}_t^{\mathrm{E}} \leftarrow \arg\max_{\boldsymbol{w} \in \mathrm{ConvHull}(\mathcal{E}_t)} -f^\star(-A\boldsymbol{w}). \tag{3.10}$$

Therefore, we can say that LPBoost and ERLPBoost are instances of the fully-corrective FW algorithm for objectives $\mathbb{I}^\star_{\Delta_{m,\nu}}$ and $f^\star$, respectively. Under the max-edge weak learner assumption, one can derive the same iteration bound for ERLPBoost since $f^\star$ is η-smooth. We summarize these connections to the following theorem.

Theorem 3.1. *LPBoost, ERLPBoost, and Corr. ERLPBoost are instances of the FW algorithm.*

[2]They also suggests the line search update. This case yields a better progress than short-step, so the same iteration bound holds.

Algorithm 1 A theoretically guaranteed boosting scheme

Input: Training examples $((\boldsymbol{x}_i, y_i))_{i=1}^m \in (\mathcal{X} \times \{\pm 1\})^m$, a g-weak learner $\mathrm{WL} : \Delta_m \to \mathcal{H}$, a FW algorithm $\mathcal{F}$, a secondary algorithm $\mathcal{B}$, and parameters $\nu > 0$ and $\epsilon > 0$.

Set $A = (y_i h(\boldsymbol{x}_i)) \in [-1, +1]^{m \times \mathcal{H}}$.
Obtain a hypothesis $h_1 = \mathrm{WL}(\boldsymbol{d}_0)$, where $\boldsymbol{d}_0 = \frac{1}{m}\mathbf{1}$.
Set $\boldsymbol{w}_1 = \boldsymbol{e}_1$.
for $t = 1, 2, \ldots, T$ **do**
  Compute the distribution $\boldsymbol{d}_t = \nabla f^\star(-A\boldsymbol{w}_t) = \arg\min_{\boldsymbol{d} \in \Delta_{m,\nu}} \left[\boldsymbol{d}^\top A \boldsymbol{w}_t + \frac{1}{\eta} R(\boldsymbol{d})\right]$.
  Obtain a hypothesis $h_{t+1} = \mathrm{WL}(\boldsymbol{d}_t)$.
  Set $\epsilon_t := \min_{0 \le \tau \le t} (\boldsymbol{d}_\tau^\top A)_{h_{\tau+1}} + f^\star(-A\boldsymbol{w}_t)$ and let $\mathcal{E}_{t+1} := \{\boldsymbol{e}_{h_\tau}\}_{\tau=1}^{t+1}$.
  if $\epsilon_t \le \epsilon/2$ **then**
    Set $T = t$, **break**.
  end if
  Compute the FW weight $\boldsymbol{w}_{t+1}^{\mathcal{F}} = \mathcal{F}(A, \boldsymbol{w}_t, \boldsymbol{e}_{h_{t+1}}, \mathcal{E}_t, \boldsymbol{d}_t)$.
  Compute the secondary weight $\boldsymbol{w}_{t+1}^{\mathcal{B}} = \mathcal{B}(A, \mathcal{E}_{t+1})$.
  Update the weight $\boldsymbol{w}_{t+1} \leftarrow \arg\min_{\boldsymbol{w} \in \{\boldsymbol{w}_{t+1}^{\mathcal{F}}, \boldsymbol{w}_{t+1}^{\mathcal{B}}\}} f^\star(-A\boldsymbol{w})$.
end for

Output: Combined classifier $H_T = \sum_{t=1}^T w_{T,t} h_t$.

Algorithm 2 LPBoost rule $\mathcal{B}(A, \mathcal{E})$

Input: A matrix $A \in [-1, +1]^{m \times \mathcal{H}}$ and a set of vectors $\mathcal{E} \subset \mathbb{R}^{\mathcal{H}}$.

Output: $\boldsymbol{w} \in \arg\max_{\boldsymbol{w} \in \mathrm{ConvHull}(\mathcal{E})} \min_{\boldsymbol{d} \in \Delta_{m,\nu}} \boldsymbol{d}^\top A \boldsymbol{w}$.

3.4 A general boosting scheme for soft margin optimization

We propose FW-like boosting schemes from the above observations, shown in Algorithm 1. Algorithm 1 takes two update rules, a FW update rule $\mathcal{F}$ and a secondary update rule $\mathcal{B}$. Both algorithms return a weight $\boldsymbol{w} \in \Delta_{\mathcal{H}}$. Intuitively, the FW update rule $\boldsymbol{w}_t^{\mathcal{F}}$ is a safety net for the convergence guarantee. Further, the convergence analysis only depends on the FW update $\boldsymbol{w}_{t+1}^{\mathcal{F}}$, so that one can incorporate any update rule to $\mathcal{B}$. For example, one can use the update rule Eq. (3.10) as $\mathcal{B}(A, \mathcal{E}_{t+1})$. Algorithm 1 becomes ERLPBoost in this case since $\boldsymbol{w}_{t+1} = \boldsymbol{w}_{t+1}^{\mathcal{B}}$ holds for any t. Even though this setting, the convergence guarantee holds so that we can prove the same convergence rate for ERLPBoost by our

Algorithm 3 Short step rule $\mathcal{F}(A, \boldsymbol{w}, \boldsymbol{s}, \mathcal{E}, \boldsymbol{d})$

Input: A matrix $A \in [-1, +1]^{m \times \mathcal{H}}$, vectors $\boldsymbol{w}, \boldsymbol{s} \in \mathbb{R}^{\mathcal{H}}$ and $\boldsymbol{d} \in \mathbb{R}^m$, and a set of vectors $\mathcal{E} \subset \mathbb{R}^{\mathcal{H}}$.

Output: $\boldsymbol{w}^{\mathcal{F}} = \boldsymbol{w} + \lambda(\boldsymbol{s} - \boldsymbol{w})$, where $\lambda = \text{clip}\, \frac{\boldsymbol{d}^\top A(\boldsymbol{s}-\boldsymbol{w})}{\eta \|A(\boldsymbol{s}-\boldsymbol{w})\|_\infty^2}$.

general analysis.

Recall that our primary objective is to find a weight vector $\boldsymbol{w}$ that optimizes Eq. (3.2). The most practical algorithm, LPBoost, solves Eq. (3.2) over past hypotheses, so using the solution as $\mathcal{B}$ is a natural choice. Algorithm 2 summarizes this update. Note that the LPBoost update differs from the fully-corrective FW algorithm since the objective function is $f^\star$, not $\mathbb{I}^\star_{\Delta_{m,\nu}}$.

Furthermore, as described by Shalev-Shwartz and Singer [88], one can compute the distribution $\boldsymbol{d}_t = \nabla f^\star(-A\boldsymbol{w}_t)$ by a sorting-based algorithm, which takes $O(m \ln m)$ iterations[3]. Thus, the time complexity per iteration depends on the secondary algorithm $\mathcal{B}$.

Before getting into the convergence analysis, we first justify the stopping criterion in Algorithm 1. This criterion is similar to the one in FW but strictly better than it. Therefore, our algorithms tend to converge in early iterations.

Lemma 3.1. *Let* $\epsilon_t := \min_{0 \le \tau \le t} (\boldsymbol{d}_\tau^\top A)_{h_{\tau+1}} + f^\star(-A\boldsymbol{w}_t)$ *be the optimality gap and let* $\eta = \frac{2}{\epsilon} \ln \frac{m}{\nu}$. *Then,* $\epsilon_t \le \frac{\epsilon}{2}$ *implies* $-\mathbb{I}^\star_{\Delta_{m,\nu}}(-A\boldsymbol{w}_t) \ge g - \epsilon$.

Proof. By the weak-learnability assumption, $\epsilon_t \ge g + f^\star(-A\boldsymbol{w}_t)$. The statement follows from Observation 2.1. □

Now, we prove the convergence rate for our scheme. This theorem also covers the convergence rate for ERLPBoost and Corr. ERLPBoost.

Theorem 3.2 (A convergence rate for Algorithm 1)**.** *Assume that the weak learner returns a hypothesis* $h_{t+1} \in \mathcal{H}$ *that satisfies* $(\boldsymbol{d}_t^\top A)_{h_{t+1}} \ge g$ *for some unknown guarantee* g. *Let* $\mathcal{F}$ *be the FW update with classic step* $\lambda_t = \frac{2}{t+2}$, *or short-step as in Algorithm 3. Then, for any secondary algorithm* $\mathcal{B}$, *Algorithm 1 converges to an* ϵ*-accurate solution of Eq.* (3.2) *in* $O\left(\frac{1}{\epsilon^2} \ln \frac{m}{\nu}\right)$ *iterations.*

[3]They also suggest a linear time algorithm, see [42].

Proof. First of all, we prove the bound for the classic step size. We start by showing the recursion

$$\epsilon_{t+1} \leq (1 - \lambda_t)\epsilon_t + 2\eta\lambda_t^2. \tag{3.11}$$

By using the definition of $\boldsymbol{w}_{t+1}$ and the η-smoothness of $f^\star$,

$$\begin{aligned}
\epsilon_t - \epsilon_{t+1} &\geq f^\star(-A\boldsymbol{w}_t) - f^\star(-A\boldsymbol{w}_t^{\mathcal{F}}) \\
&= f^\star(-A\boldsymbol{w}_t) - f^\star(-A\boldsymbol{w}_t + \lambda_t A(\boldsymbol{w}_t - \boldsymbol{e}_{h_{t+1}})) \\
&\geq \lambda_t (A(\boldsymbol{e}_{h_{t+1}} - \boldsymbol{w}_t))^\top \nabla f^\star(-A\boldsymbol{w}_t) - 2\eta\lambda_t^2,
\end{aligned} \tag{3.12}$$

where Eq. (3.12) holds since $A \in [-1, +1]^{m \times n}$ and $\boldsymbol{e}_{h_{t+1}}, \boldsymbol{w}_t \in \Delta_{\mathcal{H}}$. By the non-negativity of the entropy function and the definition of $\boldsymbol{d}_t$, we get

$$\begin{aligned}
(A(\boldsymbol{e}_{h_{t+1}} - \boldsymbol{w}_t))^\top \nabla f^\star(-A\boldsymbol{w}_t) &= \boldsymbol{d}_t^\top A(\boldsymbol{e}_{h_{t+1}} - \boldsymbol{w}_t) \\
&\geq \left[\min_{0 \leq \tau \leq t} (\boldsymbol{d}_\tau^\top A)_{h_{\tau+1}} - \boldsymbol{d}_t^\top A\boldsymbol{w}_t - \frac{1}{\eta} R(\boldsymbol{d}_t) \right] \\
&= \left[\min_{0 \leq \tau \leq t} (\boldsymbol{d}_\tau^\top A)_{h_{\tau+1}} + f^\star(-A\boldsymbol{w}_t) \right] = \epsilon_t.
\end{aligned} \tag{3.13}$$

Combining Eq. (3.12) and Eq. (3.13), we obtain Eq. (3.11).

Now, we have Eq. (3.11). Applying Lemma 2.1, we obtain $\epsilon_t \leq \frac{8\eta}{t+2}$. By the definition of η, $\epsilon_T \leq \frac{\epsilon}{2}$ holds in $T = \frac{32}{\epsilon^2} \ln \frac{m}{\nu} - 2$ iterations. Lemma 3.1 yields the convergence rate.

For the short-step case, that is, the case where we employ Algorithm 3 as $\mathcal{F}$, we get a similar recursion:

$$\begin{aligned}
\epsilon_t - \epsilon_{t+1} &\geq f^\star(-A\boldsymbol{w}_t) - f^\star(-A\boldsymbol{w}_t^{\mathcal{F}}) \\
&\geq \lambda_t (A(\boldsymbol{e}_{h_{t+1}} - \boldsymbol{w}_t))^\top \nabla f^\star(-A\boldsymbol{w}_t) - \frac{\eta}{2}\lambda_t^2 \|A(\boldsymbol{w}_t - \boldsymbol{e}_{h_{t+1}})\|_\infty^2 \\
&\geq \lambda (A(\boldsymbol{e}_{h_{t+1}} - \boldsymbol{w}_t))^\top \nabla f^\star(-A\boldsymbol{w}_t) - 2\eta\lambda^2, \qquad \forall \lambda \in [0, 1].
\end{aligned} \tag{3.14}$$

Since the short-step is the maximizer of Eq. (3.14), the last inequality holds for all $\lambda \in [0, 1]$. Optimizing λ in RHS and applying the inequality Eq. (3.13), we get $\epsilon_t - \epsilon_{t+1} \geq \frac{\epsilon_t^2}{8\eta}$. Using Lemma 20 in [88], we obtain the same iteration bound. □

Theorem 3.2 shows a convergence guarantee for the *classic step* and the *short-step*. The line search $\arg\min_{\lambda \in [0,1]} f^\star\left(-A(\boldsymbol{w}_t + \lambda(\boldsymbol{e}_{h_{t+1}} - \boldsymbol{w}_t))\right)$ always yields better progress than the *short-step*, so the same iteration bound holds.

Algorithm 4 Pairwise rule $\mathcal{F}(A, \boldsymbol{w}, \boldsymbol{s}, \mathcal{E}, \boldsymbol{d})$

Input: A matrix $A \in [-1, +1]^{m \times \mathcal{H}}$, vectors $\boldsymbol{w}, \boldsymbol{s} \in \mathbb{R}^{\mathcal{H}}$ and $\boldsymbol{d} \in \mathbb{R}^m$, and a set of vectors $\mathcal{E} \subset \mathbb{R}^{\mathcal{H}}$.

1: Let $\boldsymbol{w} = \sum_{\boldsymbol{v} \in \mathcal{E}} \alpha_{\boldsymbol{v}} \boldsymbol{v}$ be the current representation of $\boldsymbol{w}$ and let $E = \{\boldsymbol{v} \in \mathcal{E} \mid \alpha_{\boldsymbol{v}} > 0\}$.

2: Compute an *away* basis $\boldsymbol{v}^{\mathrm{Away}} \in \arg\min_{\boldsymbol{v} \in E} \boldsymbol{d}^\top A \boldsymbol{v}$ and set $\lambda_{\max} = \alpha_{\boldsymbol{v}^{\mathrm{Away}}}$.

3: $\lambda \in \arg\min_{\lambda \in [0, \lambda_{\max}]} f^\star(-A(\boldsymbol{w} + \lambda(\boldsymbol{s} - \boldsymbol{v}^{\mathrm{Away}})))$.

Output: $\boldsymbol{w}^{\mathcal{F}} = \boldsymbol{w} + \lambda(\boldsymbol{s} - \boldsymbol{v}^{\mathrm{Away}})$.

Other variants of the boosting scheme The FW update rule $\boldsymbol{w}_{t+1}^{\mathcal{F}}$ of Algorithm 3 comes from the FW algorithm with short-step sizes. One can apply other updates rules as $\mathcal{F}$. Pairwise Frank-Wolfe (PFW) is the one of a state-of-the-art Frank-Wolfe algorithm [54]. The basic idea of PFW is to move the weight from the most worthless hypothesis to the newly attained one. Algorithm 4 is the scheme that applies the PFW. By a similar argument, one can prove the convergence rate for Algorithm 4.

Corollary 3.1 (A convergence rate for Algorithm 4)**.** *Let* $k(t) = |\{\tau \in [t] \mid \lambda_\tau < \lambda_{\tau,\max}\}|$ *be the number of good steps by iteration* t*. Then, Algorithm 4 converges with rate* $O(\eta/k(t))$*.*

3.5 Experiment

First of all, we note that the gradient boosting algorithms, like XGBoost [17] or LightGBM [49], solve different problems, so we do not compare our work to them. We compare LPBoost, ERLPBoost, Corr. ERLPBoost, and our scheme on Gunnar Rätsch's benchmark datasets[4], Kaggle datasets[5], and LIBSVM datasets[6] We use a computer with Intel Xeon Gold 6124 CPU 2.60GHz processors. We call Algorithm 1 with the secondary algorithm shown in Algorithm 2 as MLPBoost. In MLPBoost, we adopt the short-step strategy as the Frank-Wolfe step size. Since the parameter η is a huge value, we expect the secondary algorithm, the LPBoost sub-routine, to be adopted much more frequently. One can try other step strategy, but it is enough for this experiment to show the performance of our algorithm. We also use the short-step strategy for Corr. ERLPBoost. We

[4] `http://theoval.cmp.uea.ac.uk/~gcc/matlab/default.html#benchmarks`

[5] `https://www.kaggle.com/`

[6] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

use the Gurobi optimizer 9.0.1[7] to solve the sub-problems of these boosting algorithms. The sub-problem of ERLPBoost cannot be solvable directory, so we use the sequential quadratic programming technique. See the following paragraph.

Implementation of ERLPBoost For each round $t = 1, 2, \ldots$, ERLPBoost solves the following convex programming

$$\begin{aligned}(\gamma, \boldsymbol{d}) = \arg\min_{\gamma, \boldsymbol{d}} \gamma + \frac{1}{\eta} \sum_{i=1}^{m} d_i \ln \frac{d_i}{1/m} \\ \text{s.t.} \quad (\boldsymbol{d}^\top A)_h \leq \gamma, \quad \forall h \in \{h_1, h_2, \ldots, h_t\}, \\ \boldsymbol{d} \in \Delta_m.\end{aligned} \tag{3.15}$$

Since Gurobi does not accept such entropy terms $d_i \ln \frac{d_i}{1/m}$, we use the sequential quadratic programming technique [70]. More precisely, let $\boldsymbol{d}_1 = \frac{1}{m}\mathbf{1}$ be the initial distribution. Starting from $k = 1$, we solve the following quadratic programming until it converges.

$$\begin{aligned}(\gamma_{k+1}, \boldsymbol{d}_{k+1}) = \arg\min_{\gamma, \boldsymbol{d}} \gamma + \frac{1}{\eta} \sum_{i=1}^{m} \frac{1}{2d_{k,i}} d_i^2 + \left(\ln \frac{d_{k,i}}{1/m}\right) d_i \\ \text{s.t.} \quad \left(\boldsymbol{d}^\top A\right)_h \leq \gamma, \quad \forall h \in \{h_1, h_2, \ldots, h_t\}, \\ \boldsymbol{d} \in \Delta_{m,\nu}\end{aligned}$$

Since the objective function is strongly convex, we expect this procedure to converge in a few rounds. This procedure terminates if either one of the following holds: (i) the optimal value does not decrease more than 10^{-9} (ii) there exists $i \in [m]$ such that $d_i = 0$. If the quadratic programming terminates at round k, it outputs $\boldsymbol{d}_k$ as the solution to Eq. (3.15).

Settings We use the capping parameters $\nu \in \mathcal{N} := \{0.01m, 0.05m, 0.1m, 0.2m\}$ and the tolerance parameter $\epsilon = 0.01$, where m is the number of training examples. As discussed in the previous sections, the soft margin optimization regards at most ν examples as outliers. We use the weak learner that returns a decision tree of depth 2. The splitting rule for the decision tree is based on the entropic impurity.

Computation time We measure the running time for each round. The algorithms may take long to converge, so we abort the computations after an hour. Since LIBSVM datasets are huge, we abort the computations for them after five hours. Figs. 3.1 to 3.4,

[7] `https://www.gurobi.com/`

Table 3.2: The worst-case behavior of LPBoost when given m training examples with capping parameter $\nu = 1$. As suggested by Warmuth et al. [97], one can extend to the soft margin setting $\nu \in (1, m]$. Even for such cases, similar phenomenon happen.

	LPBoost	ERLPBoost	MLPBoost
# of iterations	$m/2$	2	2

Figs. 3.5 to 3.8, and Figs. 3.9 to 3.12 show the comparison for the benchmark datasets, Kaggle datasets, and LIBSVM datasets, respectively. In order to clear the endpoints of the curves, we draw marks. The marks `L`, `E`, `C`, and `M` correspond to LPBoost, ERLPBoost, Corr. ERLPBoost, and MLPBoost. Here, we note that the curves show the soft margin objective defined in Eq. (3.2), which is not the objective function for ERLPBoost, Corr. ERLPBoost, and MLPBoost. Therefore, the curves for these algorithms do not increase monotonically. As we expected, MLPBoost performs like LPBoost for some datasets. Some experiments show that the stopping criterion may be bad since MLPBoost does not terminate even though it has already reached an approximate value. Although the results highly depend on the datasets, MLPBoost performs well compared to ERLPBoost and Corr. ERLPBoost. Furthermore, MLPBoost is as fast as LPBoost for some datasets, such as Ringnorm, Twonorm, and Waveform.

The worst case for LPBoost Figs. 3.1 to 3.12 show that LPBoost outperforms other algorithms. Although LPBoost has no non-trivial iteration bound, it has the worst-case lower bound. For the worst case, it takes $m/2$ iterations [97]. The paper suggests a weak learner for the worst case, which picks one from $O(m)$ hypotheses. The hypothesis set has two good hypotheses that construct a margin-maximizing combined hypothesis, and the rest are the bad ones. One of the good hypotheses can be attained at the first iteration, which gives uniform distribution to the weak learner. Since LPBoost assigns zero weight to some examples, the weak learner chooses bad hypotheses repeatedly. Even in this case, MLPBoost and ERLPBoost terminate in 2 iterations[8]. These algorithms put a non-zero weight for all examples so the weak learner can produce the good hypothesis in the second iteration. We summarize this fact in Table 3.2 since the experiment figure has no information due to the behavior.

[8]One can try this experiment by using `https://github.com/rmitsuboshi/miniboosts`.

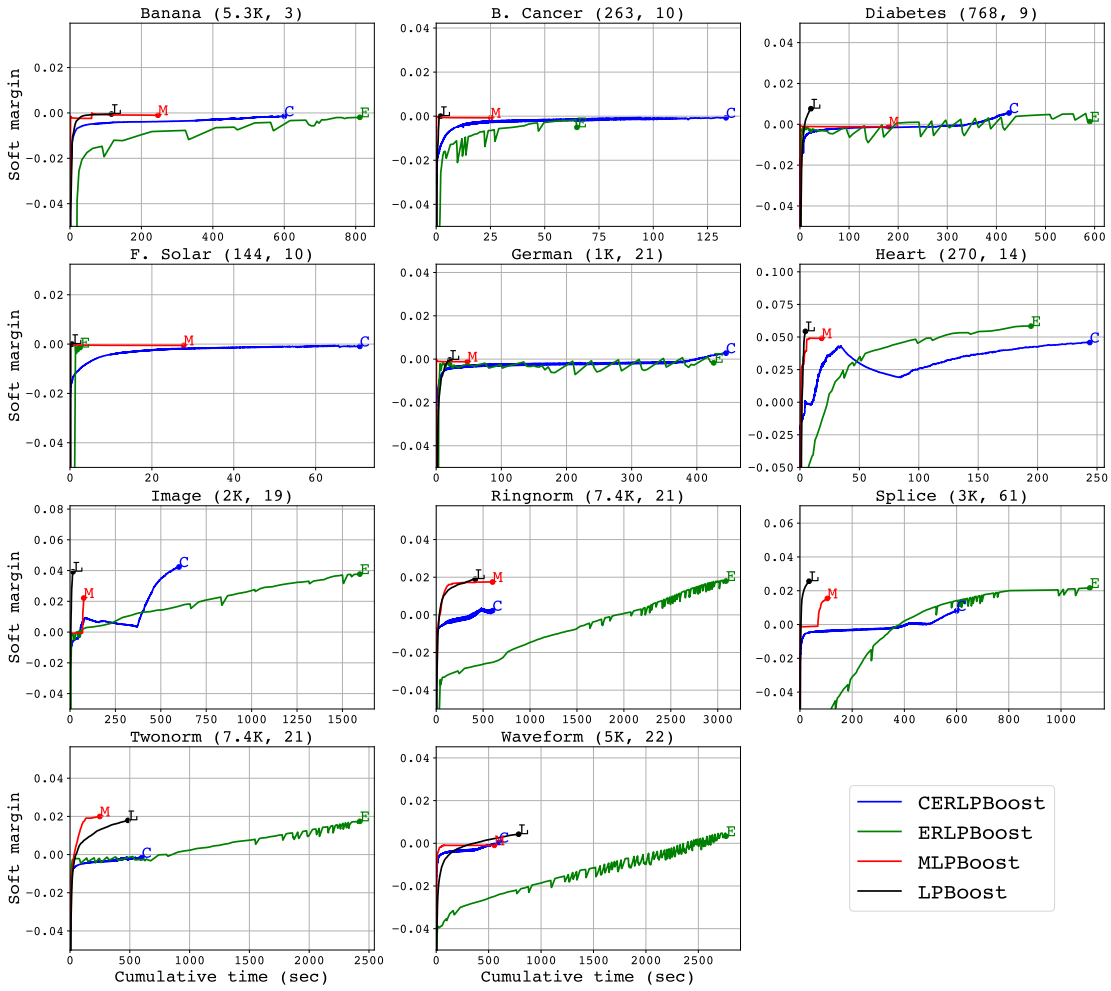

Figure 3.1: Comparison of the algorithm for the benchmark datasets with parameters $\epsilon = 0.01$ and $\nu = 0.01m$. The vertical axis shows the soft margin objective values and the horizontal one shows the cumulative time (seconds). All algorithms aborted in an hour, so some algorithms did not reach the approximate solution. The tuple on each title shows (# of examples, # of features).

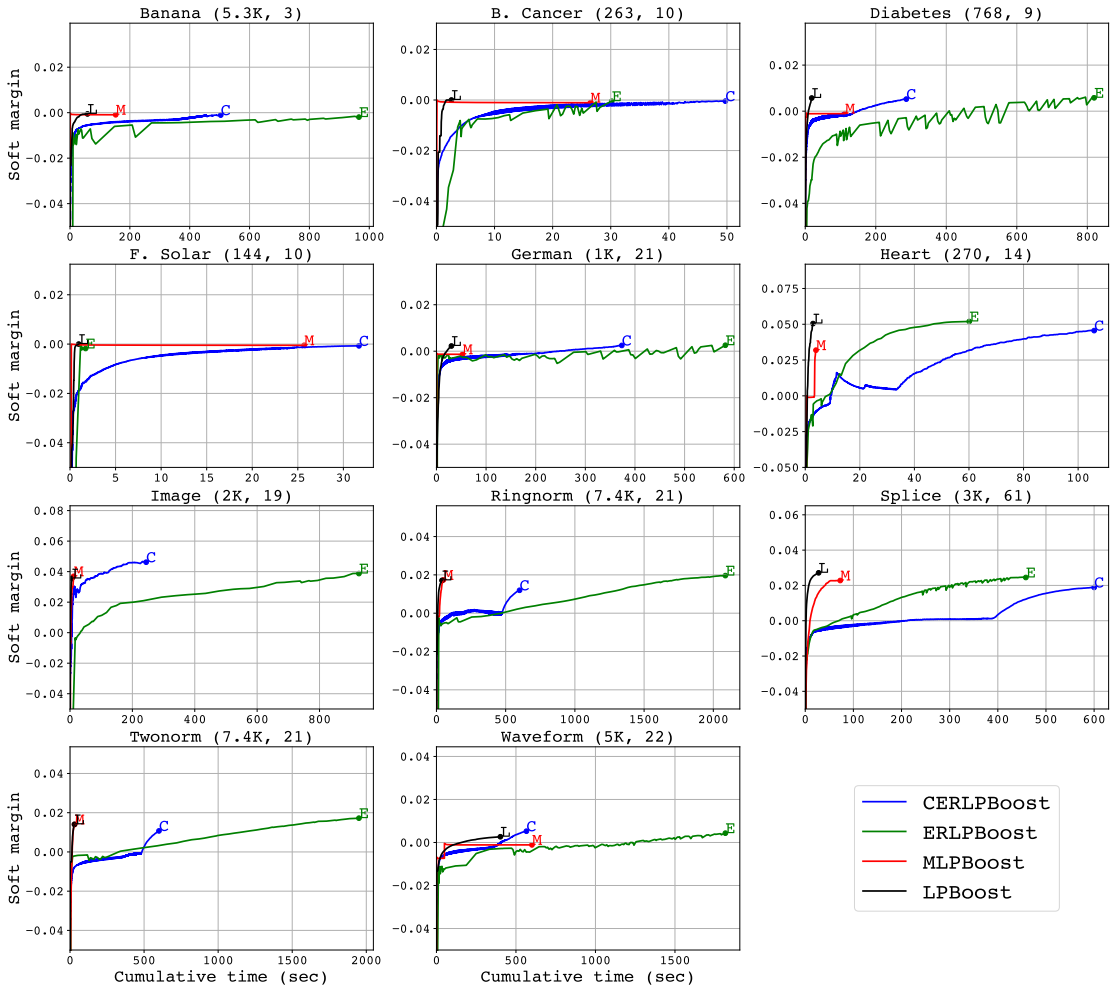

Figure 3.2: Comparison of the algorithm for the benchmark datasets with parameters $\epsilon = 0.01$ and $\nu = 0.05m$. The vertical axis shows the soft margin objective values and the horizontal one shows the cumulative time (seconds). All algorithms aborted in an hour, so some algorithms did not reach the approximate solution. The tuple on each title shows (# of examples, # of features).

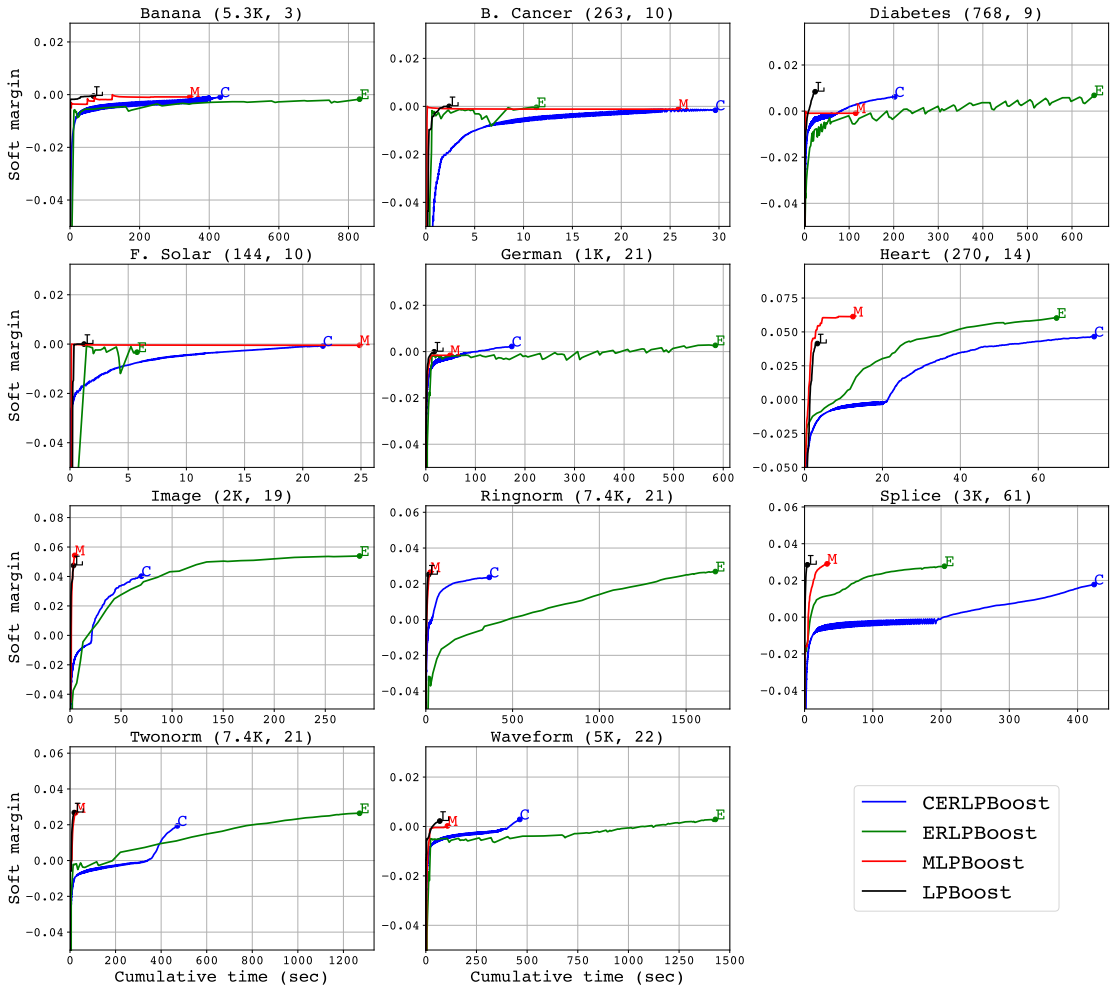

Figure 3.3: Comparison of the algorithm for the benchmark datasets with parameters $\epsilon = 0.01$ and $\nu = 0.1m$. The vertical axis shows the soft margin objective values and the horizontal one shows the cumulative time (seconds). All algorithms aborted in an hour, so some algorithms did not reach the approximate solution. The tuple on each title shows (# of examples, # of features).

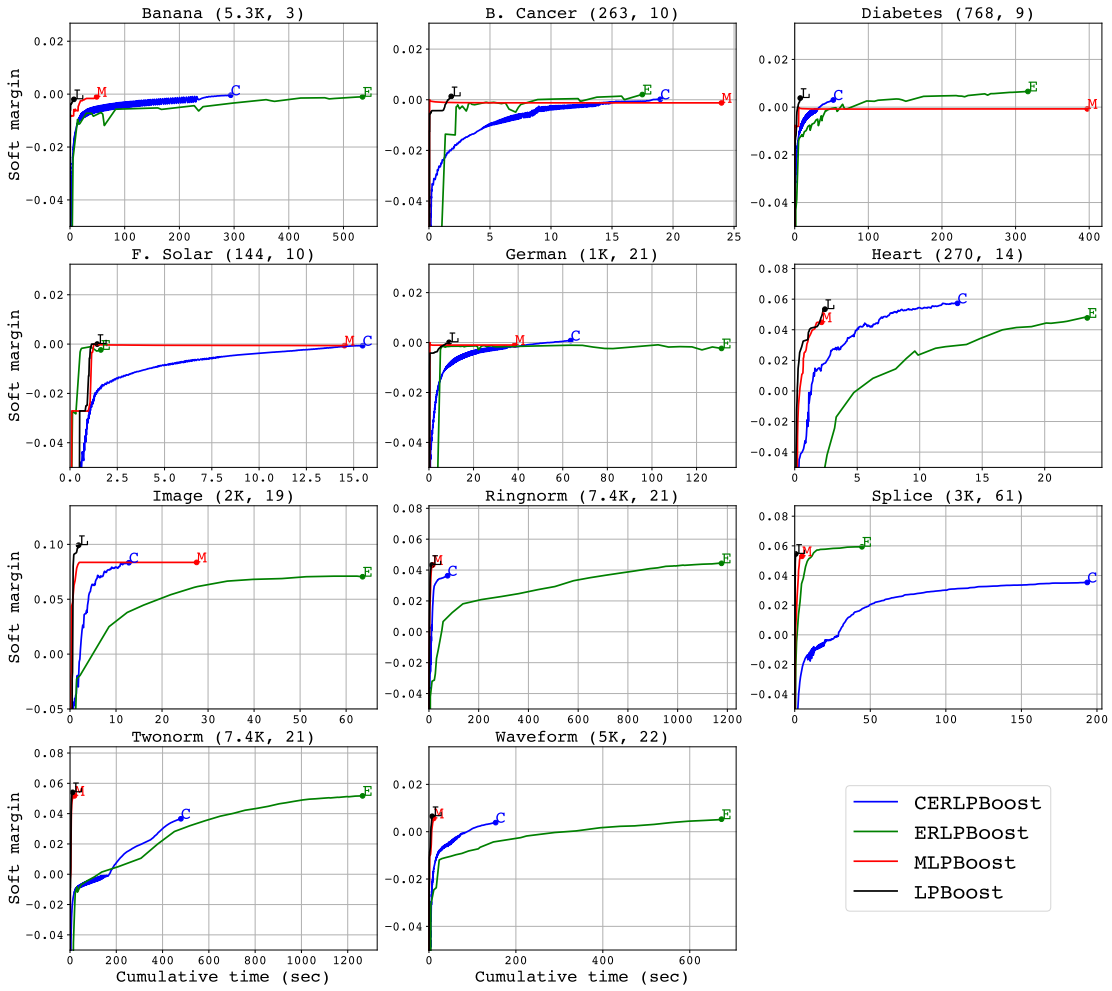

Figure 3.4: Comparison of the algorithm for the benchmark datasets with parameters $\epsilon = 0.01$ and $\nu = 0.2m$. The vertical axis shows the soft margin objective values and the horizontal one shows the cumulative time (seconds). All algorithms aborted in an hour, so some algorithms did not reach the approximate solution. The tuple on each title shows (# of examples, # of features).

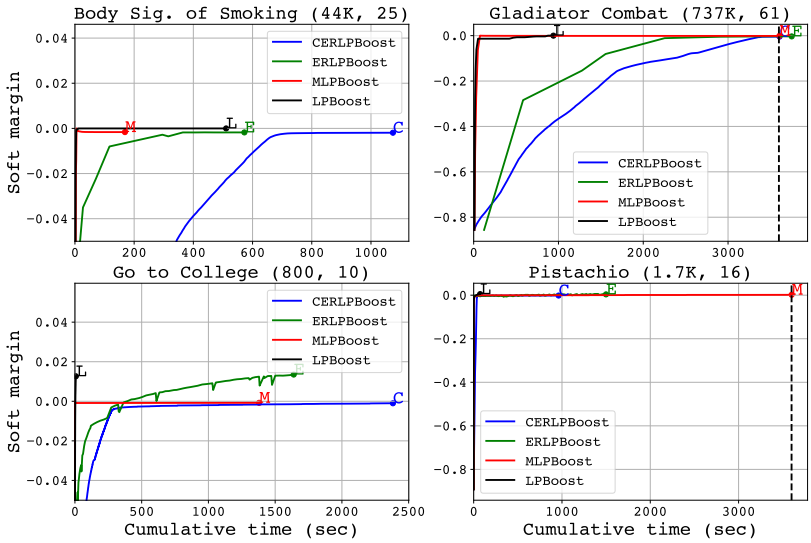

Figure 3.5: Comparison of the algorithm for the Kaggle datasets with parameters $\epsilon = 0.01$ and $\nu = 0.01m$. The vertical axis shows the soft margin objective values and the horizontal one shows the cumulative time (seconds). All algorithms aborted in an hour, so some algorithms did not reach the approximate solution. The vertical dashed line shows the an-hour time limit. The tuple on each title shows (# of examples, # of features).

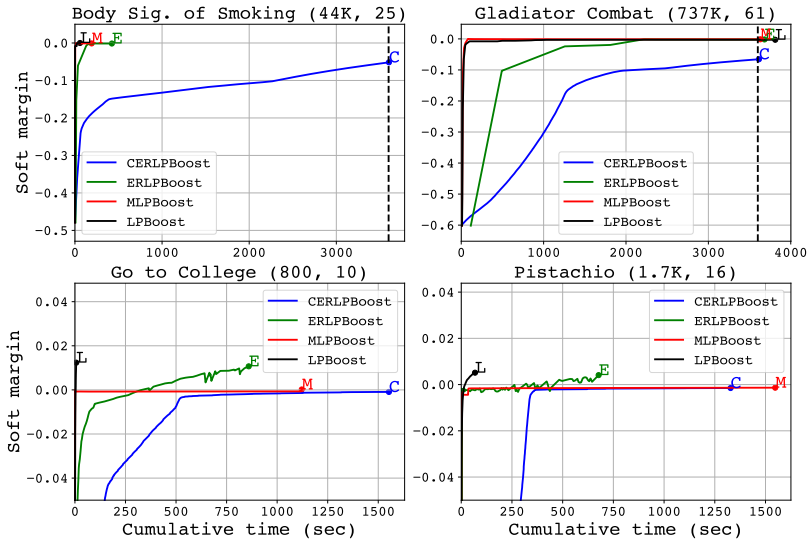

Figure 3.6: Comparison of the algorithm for the Kaggle datasets with parameters $\epsilon = 0.01$ and $\nu = 0.05m$. The vertical axis shows the soft margin objective values and the horizontal one shows the cumulative time (seconds). All algorithms aborted in an hour, so some algorithms did not reach the approximate solution. The vertical dashed line shows the an-hour time limit. The tuple on each title shows (# of examples, # of features).

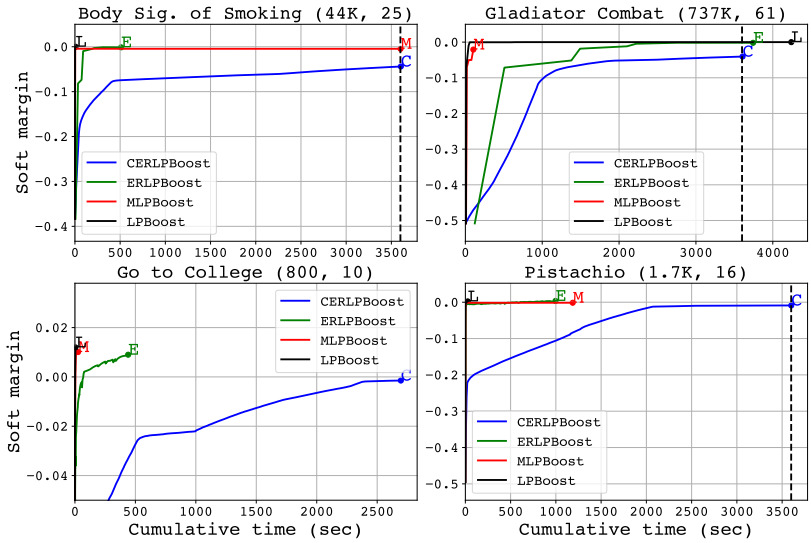

Figure 3.7: Comparison of the algorithm for the Kaggle datasets with parameters $\epsilon = 0.01$ and $\nu = 0.1m$. The vertical axis shows the soft margin objective values and the horizontal one shows the cumulative time (seconds). All algorithms aborted in an hour, so some algorithms did not reach the approximate solution. The vertical dashed line shows the an-hour time limit. The tuple on each title shows (# of examples, # of features).

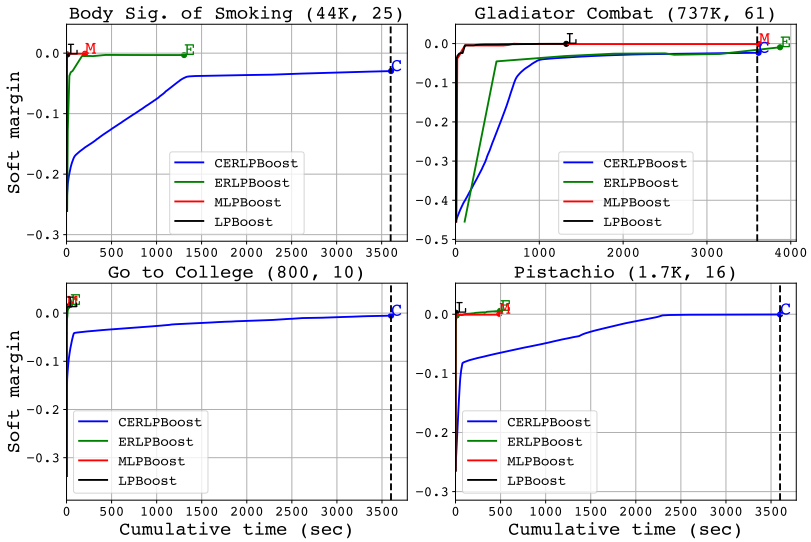

Figure 3.8: Comparison of the algorithm for the Kaggle datasets with parameters $\epsilon = 0.01$ and $\nu = 0.2m$. The vertical axis shows the soft margin objective values and the horizontal one shows the cumulative time (seconds). All algorithms aborted in an hour, so some algorithms did not reach the approximate solution. The vertical dashed line shows the an-hour time limit. The tuple on each title shows (# of examples, # of features).

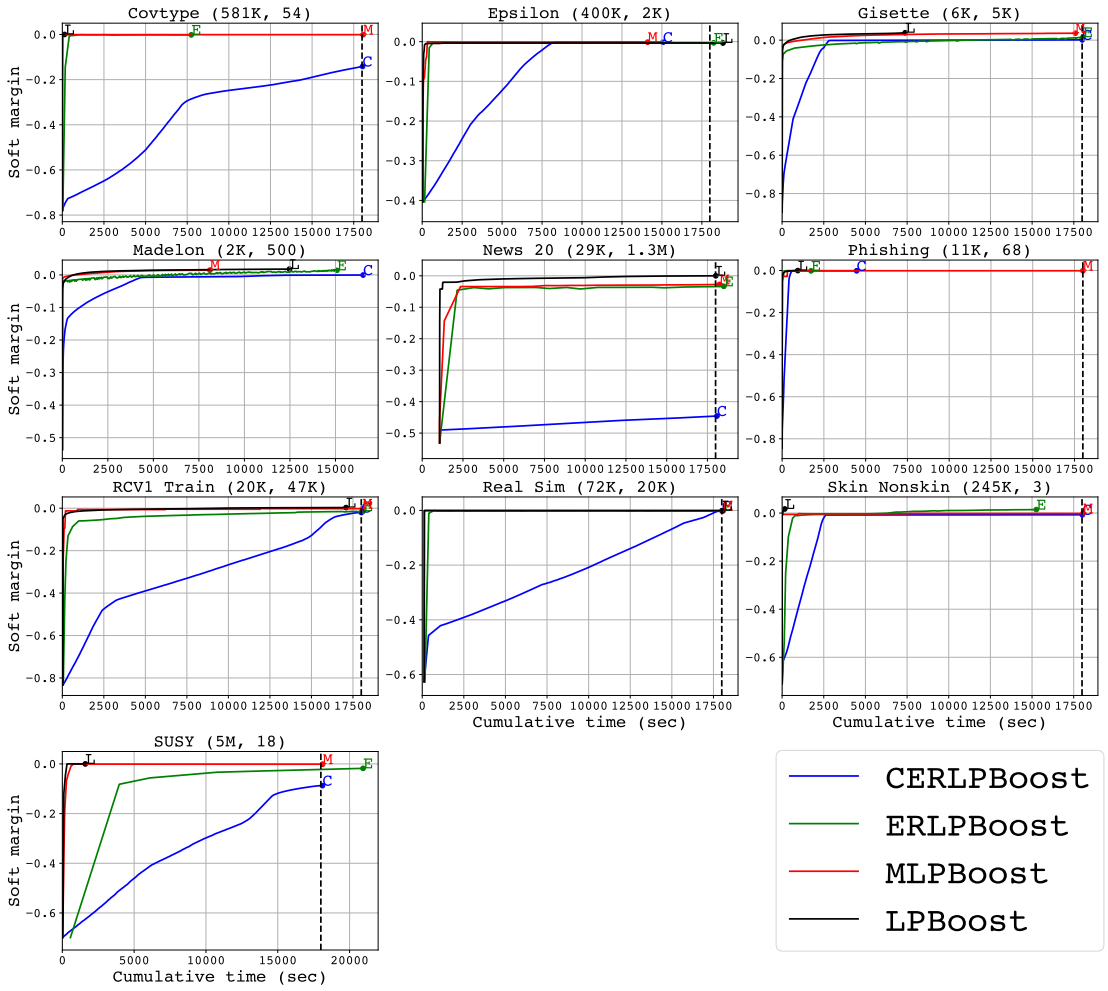

Figure 3.9: Comparison of the algorithm for the LIBSVM datasets with parameters $\epsilon = 0.01$ and $\nu = 0.01m$. The vertical axis shows the soft margin objective values and the horizontal one shows the cumulative time (seconds). All algorithms aborted in five hours, so some algorithms did not reach the approximate solution. The vertical dashed line shows the an-hour time limit. The tuple on each title shows (# of examples, # of features).

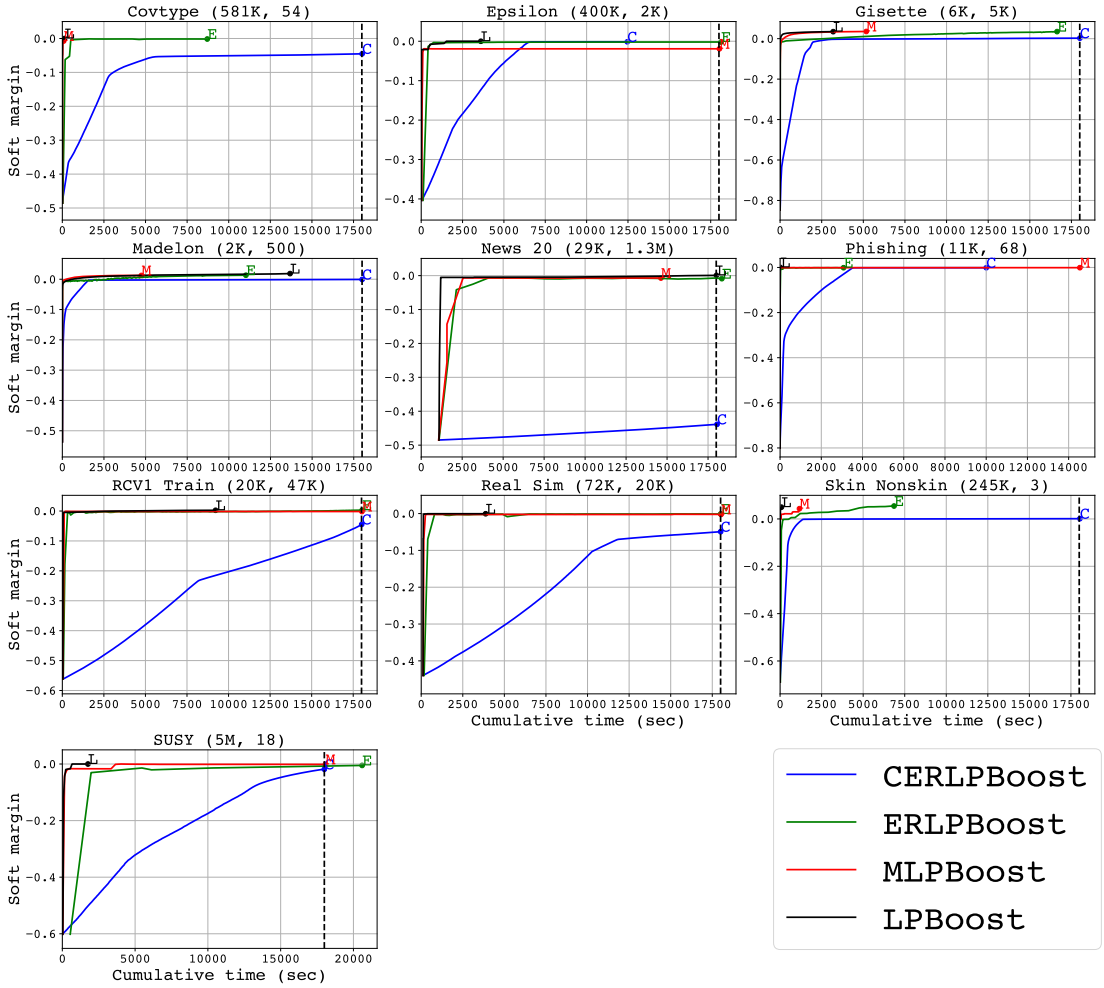

Figure 3.10: Comparison of the algorithm for the LIBSVM datasets with parameters $\epsilon = 0.01$ and $\nu = 0.05m$. The vertical axis shows the soft margin objective values and the horizontal one shows the cumulative time (seconds). All algorithms aborted in five hours, so some algorithms did not reach the approximate solution. The vertical dashed line shows the an-hour time limit. The tuple on each title shows (# of examples, # of features).

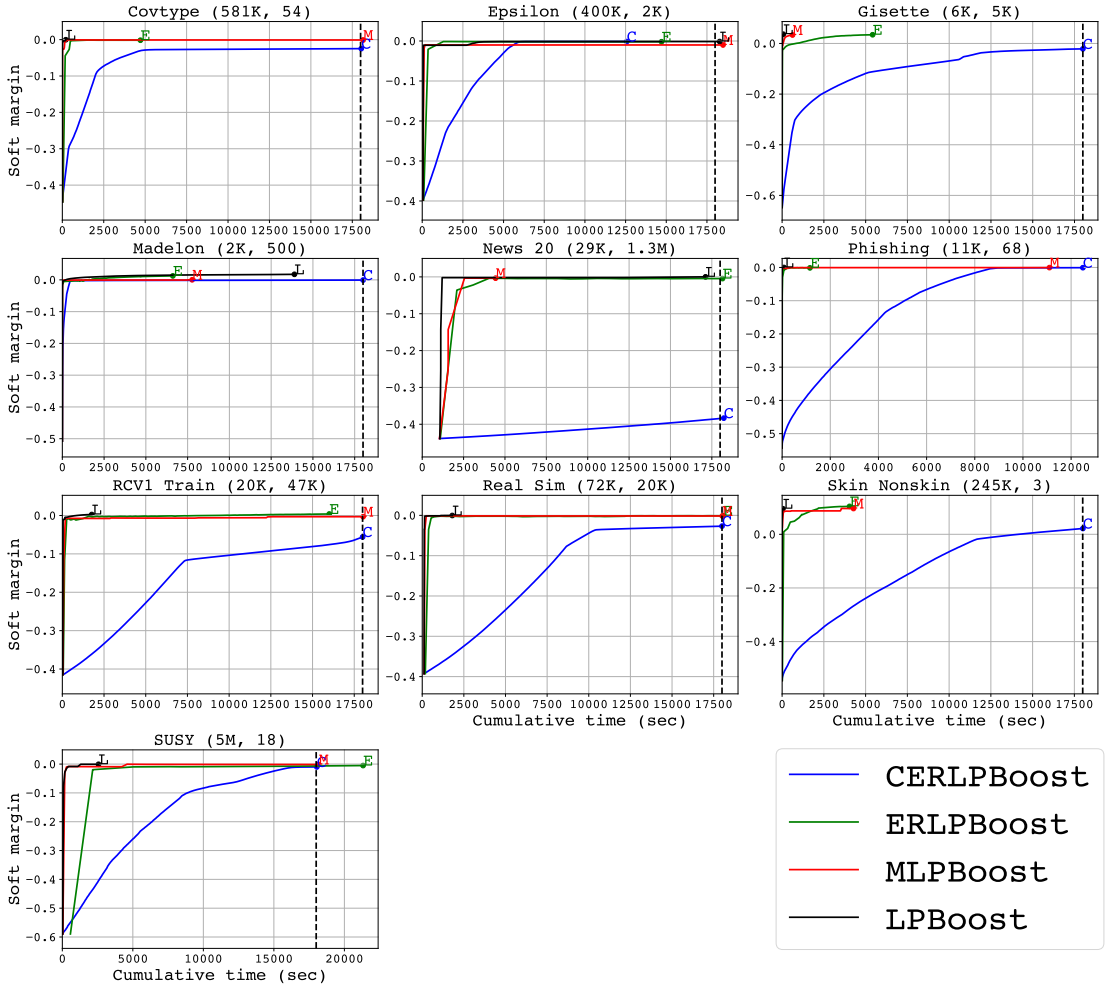

Figure 3.11: Comparison of the algorithm for the LIBSVM datasets with parameters $\epsilon = 0.01$ and $\nu = 0.1m$. The vertical axis shows the soft margin objective values and the horizontal one shows the cumulative time (seconds). All algorithms aborted in five hours, so some algorithms did not reach the approximate solution. The vertical dashed line shows the an-hour time limit. The tuple on each title shows (# of examples, # of features).

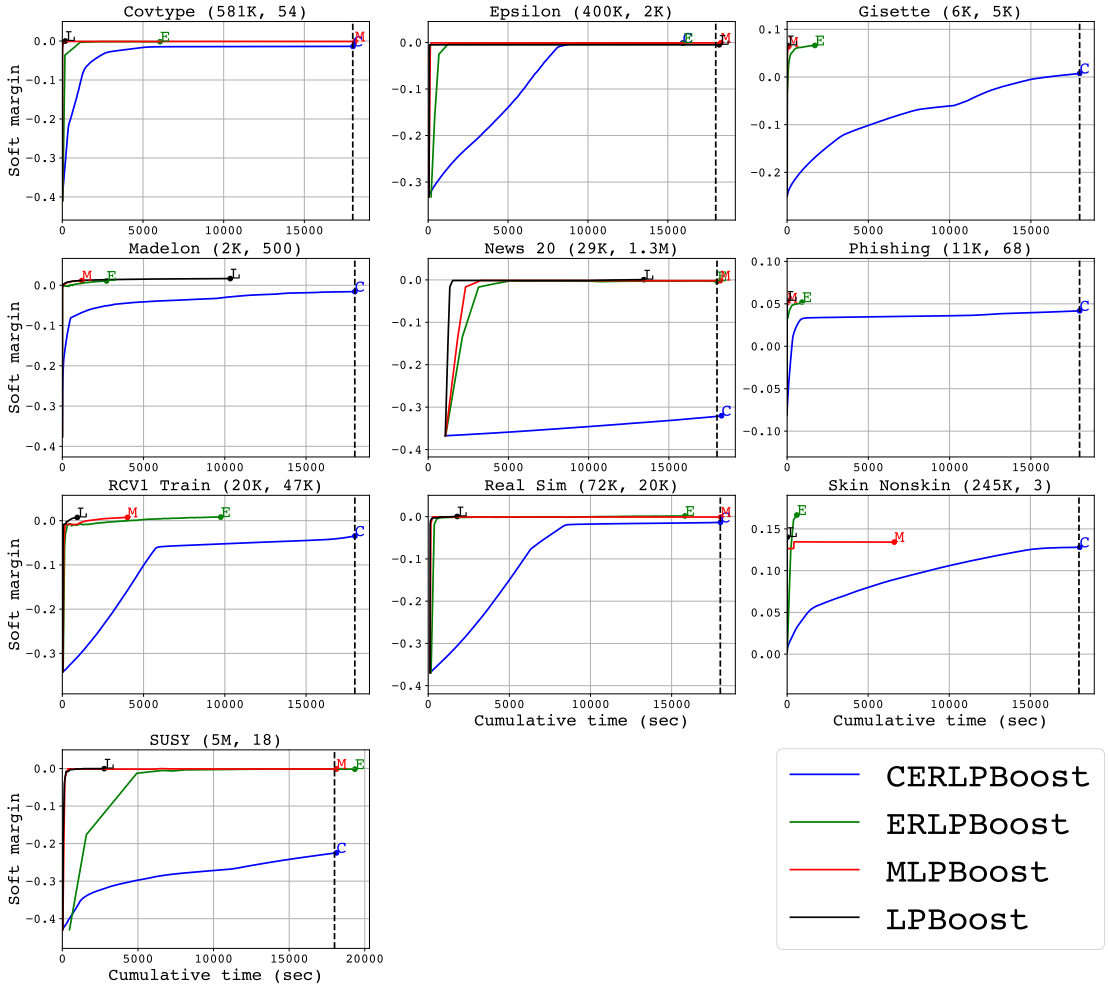

Figure 3.12: Comparison of the algorithm for the LIBSVM datasets with parameters $\epsilon = 0.01$ and $\nu = 0.2m$. The vertical axis shows the soft margin objective values and the horizontal one shows the cumulative time (seconds). All algorithms aborted in five hours, so some algorithms did not reach the approximate solution. The vertical dashed line shows the an-hour time limit. The tuple on each title shows (# of examples, # of features).

Chapter 4

Solving Linear Regression with Insensitive Loss by Boosting

This chapter develops a boosting algorithms for regression problem, while the previous one considers binary classification. Many regression algorithms aim to find a function that minimizes the average loss for training points. The regression boosting algorithm we consider here aims to find a function that minimizes the sum of Hinge loss plus the Hinge loss parameter. The most recent work for this setting is the regression version of LPBoost [21]. Similar to the LPBoost algorithm for classification, the regression version does not have a convergence rate.

This chapter introduces a new boosting algorithm for the setting. The proposed algorithm converges to an ϵ-approximate solution in $O\left(\ln(m/\nu)/\epsilon^2\right)$ rounds.

The results in this chapter appear in [66].

4.1 Problem setting

Throughout this chapter, we consider a regression problem. Given a sequence $S = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\} \subset \mathcal{X} \times \mathcal{Y}$ of training examples and a parameter $\nu \in [1, m]$,

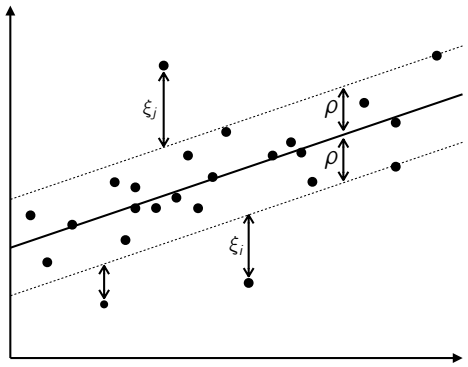

Figure 4.1: An one-dimensional example of Eq. (4.1). All examples except for some outliers only differ ρ from its prediction.

we aim to solve the following problem

$$\begin{aligned} &\min_{\rho,\boldsymbol{w},\boldsymbol{\xi}} \rho + \frac{1}{\nu}\sum_{i=1}^{m}\xi_i \\ &\text{s.t.} \quad \xi_i \geq \left| y_i - \sum_{h\in\mathcal{H}} w_h h(\boldsymbol{x}_i) \right| - \rho, \quad \forall i \in [m], \\ &\qquad \sum_{h\in\mathcal{H}} w_h = 1, \quad \boldsymbol{w} \in \mathbb{R}_+^{\mathcal{H}}, \quad \boldsymbol{\xi} \geq \boldsymbol{0}, \end{aligned} \tag{4.1}$$

where $\mathcal{Y} = [-b, b] \subset \mathbb{R}$ is a bounded set for some $b > 0$ and $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$ is a hypothesis set to be combined. That is, we aim to find the best convex combination of hypotheses in $\mathcal{H}$ for (4.1). Here, the hyperparameter ν plays a role in the upper-bound of the fraction of outliers. This formulation is effective when one knows the fraction of outliers. Figure 4.1 illustrates this problem. Since the ρ-insensitive loss to be considered is not smooth so that one cannot apply the previous algorithm solves Eq. (4.1) by modifying LPBoost to regression problems [21]. Similar to LPBoost, their algorithm for Eq. (4.1), LPBoost-Regression, does not have a convergence guarantee. This paper aims to derive a boosting algorithm that converges in polynomial iterations regarding accuracy parameter $\epsilon > 0$ and sample size $m \in \mathbb{N}$.

We consider the following protocol for regression boosting; In each round t, Booster chooses a distribution over S. Then, Weak Learner returns a hypothesis $h \in \mathcal{H}$ whose prediction is slightly better than random guessing. Note that our boosting protocol only changes the distribution over examples, while gradient boosting protocol also change the target values $\{y_i\}_{i=1}^m$. At the end of this paper, we also verify the effectively and property of our algorithm through some real-data demonstration.

4.2 Related work

The following problem is the ℓ_2-regularized version of Eq. (4.1), which is known as ν-SVR [83].

$$\begin{aligned} \min_{\rho,\boldsymbol{w},b,\boldsymbol{\xi}} \rho + \frac{1}{\nu}\sum_{i=1}^{m}\xi_i + \frac{C}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} |y_i - (\boldsymbol{w}\cdot\boldsymbol{x}_i + b)| \leq \rho + \xi_i, \quad \forall i \in [m] \end{aligned} \tag{4.2}$$

They introduced a parameter ν to SVR, which controls an upper bound of the fraction of outliers. Similar to the result for ν-SVR, one can prove that the parameter ν is an upper bound of the fraction of examples that lie outside of the tube of width ρ. Shawe-Taylor and Cristianini [89] analyzed the generalization ability of non-separable cases of SVR, which includes ν-SVR. Roughly speaking, they showed in Theorem VIII.3 that the probability that a test point $(\boldsymbol{x}, y)$ has non-zero ρ-insensitive loss is bounded by $O(\frac{1}{\rho^2}\lg\frac{1}{\rho^2}\lg\frac{m}{\rho})$.

Solving Eq. (4.2) is equivalent to the following problem for some $R > 0$.

$$\begin{aligned} \min_{\rho,\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \rho + \frac{1}{\nu}\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & |y_i - (\boldsymbol{w}\cdot\boldsymbol{x}_i + b)| \leq \rho + \xi_i, \quad \forall i \in [m], \\ & \|\boldsymbol{w}\|_2 \leq R. \end{aligned} \tag{4.3}$$

One can derive this equivalence in the following way; Let $\boldsymbol{w}^\star$ be the optimal solution to Eq. (4.2). Setting $R = \|\boldsymbol{w}^\star\|_2$, one can show that an optimal solution of Eq. (4.2) is a feasible solution of Eq. (4.3), and vice versa. Our formulation, Eq. (4.1), is obtained by restricting $R = 1$ and replacing the norm of $\boldsymbol{w}$ from ℓ_2 to ℓ_1. Thus, we expect the resulting hypothesis to have a small generalization error and some degree of sparsity.

Gradient Boosting Machine algorithms, also known as GBMs, are invented by Friedman [29]. Unlike the classical boosting setting, their protocol modifies target values $\{y_k\}_{k=1}^m$ instead of the distribution over examples. XGBoost [17] and LightGBM [49], are the most known GBMs for regression. These algorithms restrict the weak learner to produce decision trees. Even though they work very well in practice, boosting is more wide framework in origin. Further, the weak learner is assumed to return a decision tree that approximately minimizes the training loss with respect to some loss function. This approximation comes from two factors; feature binning, which significantly decreases running time, and quadratic loss approximation, which allows using general loss functions.

Both are great ideas for speedup, but theoretical analysis becomes hard since most gradient boosting-based analyses assume a weak learner that minimizes certain loss functions.

4.3 Our algorithm

First, we derive the Lagrange dual problem to Eq. (4.1). Eq. (4.1) can be represented as

$$\begin{aligned}
&\min_{\rho,\boldsymbol{w},\boldsymbol{\xi}} \rho + \frac{1}{\nu}\sum_{i=1}^{m}\xi_i \\
&\text{s.t.} \quad \xi_i \geq y_i - \sum_{h\in\mathcal{H}} w_h h(\boldsymbol{x}_i) - \rho, \quad \forall i \in [m], \\
&\qquad \xi_i \geq \sum_{h\in\mathcal{H}} w_h h(\boldsymbol{x}_i) - y_i - \rho, \quad \forall i \in [m], \\
&\qquad \sum_{h\in\mathcal{H}} w_h = 1, \quad \boldsymbol{w} \in \mathbb{R}_+^{\mathcal{H}}, \quad \boldsymbol{\xi} \in \mathbb{R}_+^m.
\end{aligned}$$

Let $\boldsymbol{d}^+, \boldsymbol{d}^- \in \mathbb{R}_+^m, \gamma \in \mathbb{R}, \boldsymbol{s} \in \mathbb{R}_+^{\mathcal{H}}$, and $\boldsymbol{\psi} \in \mathbb{R}_+^m$ be the Lagrange variables for the above problem. Define the Lagrangian as

$$\begin{aligned}
&L(\rho, \boldsymbol{w}, \boldsymbol{\xi}; \boldsymbol{d}^+, \boldsymbol{d}^-, \gamma, \boldsymbol{s}, \boldsymbol{\psi}) \\
=&\rho + \frac{1}{\nu}\sum_{i=1}^{m}\xi_i \\
&+ \sum_{i=1}^{m} d_i^+ \left(y_i - \sum_{h\in\mathcal{H}} w_h h(\boldsymbol{x}_i) - \rho - \xi_i \right) + \sum_{i=1}^{m} d_i^- \left(\sum_{h\in\mathcal{H}} w_h h(\boldsymbol{x}_i) - y_i - \rho - \xi_i \right) \\
&+ \gamma \left(1 - \sum_{h\in\mathcal{H}} w_h \right) - \boldsymbol{w}\cdot\boldsymbol{s} - \boldsymbol{\xi}\cdot\boldsymbol{\psi}. \qquad (4.4)
\end{aligned}$$

Solving Eq. (4.4) w.r.t. the primal variables $(\rho, \boldsymbol{w}, \boldsymbol{\xi})$, we obtain the dual problem of the following form.

$$\begin{aligned}
&\max_{\boldsymbol{d}^+,\boldsymbol{d}^-,\gamma} \sum_{i=1}^{m}(d_i^+ - d_i^-)y_i + \gamma, \\
&\text{s.t.} \quad \gamma \leq -\sum_{i=1}^{m}(d_i^+ - d_i^-)h(\boldsymbol{x}_i), \quad \forall h \in \mathcal{H}, \\
&\qquad \boldsymbol{d}^+ + \boldsymbol{d}^- \leq \frac{1}{\nu}\mathbf{1}, \\
&\qquad \sum_{i=1}^{m} d_i^+ + d_i^- = 1, \quad \boldsymbol{d}^+, \boldsymbol{d}^- \geq \mathbf{0}.
\end{aligned}$$

Given any feasible solution $\boldsymbol{d}^+$ and $\boldsymbol{d}^-$, the optimal γ is $\min_{h\in\mathcal{H}} -\sum_{i=1}^m (d_i^+ - d_i^-)h(\boldsymbol{x}_i)$. Thus, the objective function becomes

$$\sum_{i=1}^m (d_i^+ - d_i^-)y_i + \min_{h\in\mathcal{H}} -\sum_{i=1}^m (d_i^+ - d_i^-)h(\boldsymbol{x}_i) = \min_{h\in\mathcal{H}} \sum_{i=1}^m (d_i^+ - d_i^-)(y_i - h(\boldsymbol{x}_i)),$$

Thus, we get the Lagrange dual problem of Eq. (4.1).

$$\begin{aligned} &\max_{\boldsymbol{d}^+,\boldsymbol{d}^-} \min_{h\in\mathcal{H}} \sum_{i=1}^m (d_i^+ - d_i^-)(y_i - h(\boldsymbol{x}_i)) \\ &\text{s.t.} \quad \boldsymbol{d}^+ + \boldsymbol{d}^- \in \Delta_{m,\nu}, \boldsymbol{d}^+, \boldsymbol{d}^- \geq \boldsymbol{0}, \end{aligned} \tag{4.5}$$

or equivalently,

$$\max_{\boldsymbol{d}\in\Delta_{m,\nu}} \min_{h\in\mathcal{H}} \sum_{i=1}^m d_i |y_i - h(\boldsymbol{x}_i)|. \tag{4.6}$$

That is, the dual problem Eq. (4.6) aims to find a distribution over S that maximizes the average of absolute error for all $h \in \mathcal{H}$. Based on the duality, we define the weak-learnability for Eq. (4.1) as follows. Note that this weak learnability is motivated for the one for classification.

Definition 4.1. *A hypothesis set $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$ is said to be γ-weak learnable if for any distribution $\boldsymbol{d} \in \Delta_{m,\nu}$, there exists a hypothesis $h \in \mathcal{H}$ such that $\sum_{i=1}^m d_i |y_i - h(\boldsymbol{x}_i)| \leq \gamma$. We call an algorithm that returns such a hypothesis as a γ-weak learner.*

A γ-weak learner is "weak" in the following sense; As described in Eq. (4.1), our objective is to find a convex combination of hypotheses whose loss for *each* training point $(\boldsymbol{x}_i, y_i)$ is at most ρ, except for some outliers. On the other hand, a γ-weak learner is only required to return a hypothesis whose *average* absolute loss w.r.t. a given distribution is at most γ.

From now on, we will construct a boosting algorithm that finds an ϵ-approximate solution of Eq. (4.1), assuming the existence of a γ-weak learner. For notational simplicity, let $A \in \mathbb{R}^{m\times\mathcal{H}}$ be the matrix such that $A_{i,h} = y_i - h(\boldsymbol{x}_i)$. With this notation, we define

the entropy regularized problem of Eq. (4.5).

$$\begin{aligned} &\max_{\boldsymbol{d}^+,\boldsymbol{d}^-} g\left(-\begin{bmatrix}\boldsymbol{d}^+\\ \boldsymbol{d}^-\end{bmatrix}^\top \begin{bmatrix}A\\ -A\end{bmatrix}\right) - f\left(\begin{bmatrix}\boldsymbol{d}^+\\ \boldsymbol{d}^-\end{bmatrix}\right) \\ &\text{where } f\left(\begin{bmatrix}\boldsymbol{d}^+\\ \boldsymbol{d}^-\end{bmatrix}\right) = \mathbb{I}_{\Delta_{m,\nu}}(\boldsymbol{d}^+ + \boldsymbol{d}^-) + \frac{1}{\eta} R(\boldsymbol{d}^+ + \boldsymbol{d}^-), \\ &\quad g(\boldsymbol{\theta}) = \max_{h\in\mathcal{H}} \theta_h. \end{aligned} \tag{4.7}$$

By Theorem 2.1, one can obtain the dual problem of Eq. (4.7):

$$\min_{\boldsymbol{w}\in\Delta_{\mathcal{H}}} f^\star\left(\begin{bmatrix}A\\ -A\end{bmatrix}\boldsymbol{w}\right), \tag{4.8}$$

where $f^\star$ is the Fenchel conjugate function defined as

$$\begin{aligned} f^\star\left(\begin{bmatrix}A\\ -A\end{bmatrix}\boldsymbol{w}\right) &= \sup_{(\boldsymbol{d}^+,\boldsymbol{d}^-)} \begin{bmatrix}\boldsymbol{d}^+\\ \boldsymbol{d}^-\end{bmatrix}^\top \begin{bmatrix}A\\ -A\end{bmatrix}\boldsymbol{w} - f\left(\begin{bmatrix}\boldsymbol{d}^+\\ \boldsymbol{d}^-\end{bmatrix}\right) \\ &= \sup_{\boldsymbol{d}\in\Delta_{m,\nu}} \sum_{i=1}^m d_i |(A\boldsymbol{w})_i| - R(\boldsymbol{d}). \end{aligned} \tag{4.9}$$

By definitions of f and g,

$$\begin{aligned} \mathbf{0} \in &\text{core}\left(\operatorname{dom} g - \begin{bmatrix}A\\ -A\end{bmatrix}^\top \operatorname{dom} f\right) \\ =&\text{core}\left(\left\{\boldsymbol{w} - A^\top(\boldsymbol{d}^+ - \boldsymbol{d}^-) \mid \boldsymbol{w}\in\mathbb{R}^{\mathcal{H}}, \begin{bmatrix}\boldsymbol{d}^+\\ \boldsymbol{d}^-\end{bmatrix} \in \operatorname{dom} f\right\}\right) \end{aligned}$$

holds so that the optimal value of Eq. (4.7) and the one of Eq. (4.8) are the same.

Note that one can obtain Eq. (4.1) in terms of Fenchel dual problem by adopting Theorem 2.1 to the following problem

$$\max_{\boldsymbol{d}^+,\boldsymbol{d}^-} g\left(-\begin{bmatrix}\boldsymbol{d}^+\\ \boldsymbol{d}^-\end{bmatrix}^\top \begin{bmatrix}A\\ -A\end{bmatrix}\right) - \mathbb{I}_{\Delta_{m,\nu}}(\boldsymbol{d}^+ + \boldsymbol{d}^-).$$

The resulting primal problem is:

$$\min_{\boldsymbol{w}\in\Delta_{\mathcal{H}}} \max_{\boldsymbol{d}\in\Delta_{m,\nu}} \sum_{i=1}^m d_i \left| y_i - \sum_{h\in\mathcal{H}} w_h h(\boldsymbol{x}_i)\right| \tag{4.10}$$

Since the strong duality also holds for this case, one can see Eq. (4.10) as an alternative formulation for the original problem in Eq. (4.1).

With an appropriate choice of $\eta > 0$, the following relation holds for problems Eq. (4.1) and Eq. (4.8).

Lemma 4.1. *If $\eta \geq \frac{2}{\epsilon} \ln \frac{m}{\nu}$, an $\frac{\epsilon}{2}$-approximate solution to Eq. (4.8) is an ϵ-approximate solution to Eq. (4.1).*

Proof. We first note that the Fenchel dual problem

$$\begin{aligned}\min_{\boldsymbol{w} \in \Delta_{\mathcal{H}}} \mathbb{I}^{\star}_{\Delta_{m,\nu}} \left(\begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w} \right) &= \min_{\boldsymbol{w} \in \Delta_{\mathcal{H}}} \max_{(\boldsymbol{d}^+, \boldsymbol{d}^-)} (\boldsymbol{d}^+ - \boldsymbol{d}^-)^\top \begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w} - \mathbb{I}_{\Delta_{m,\nu}}(\boldsymbol{d}^+ + \boldsymbol{d}^-) \\ &= \min_{\boldsymbol{w} \in \Delta_{\mathcal{H}}} \max_{\boldsymbol{d} \in \Delta_{m,\nu}} \sum_{i=1}^{m} d_i |(A\boldsymbol{w})_i|\end{aligned}$$

of Eq. (4.5) is equivalent to Eq. (4.1) since $\operatorname{dom} g = \mathbb{R}^{\mathcal{H}}$ and $\operatorname{dom} \mathbb{I}_{\Delta_{m,\nu}} = \Delta_{m,\nu}$. By our choice of η, the regularization term $\frac{1}{\eta} R(\boldsymbol{d})$ takes a value in $[0, \frac{\epsilon}{2}]$. Thus, $\mathbb{I}_{\Delta_{m,\nu}}(\boldsymbol{d}) \leq f(\boldsymbol{d}) \leq \mathbb{I}_{\Delta_{m,\nu}}(\boldsymbol{d}) + \frac{\epsilon}{2}$ holds for all $\boldsymbol{d} \in \Delta_{m,\nu}$. Using Observation 2.1, we have

$$\begin{aligned} \mathbb{I}^{\star}_{\Delta_{m,\nu}} \left(\begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w} \right) - \frac{\epsilon}{2} \leq & f^{\star} \left(\begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w} \right) \\ \text{and} \qquad f^{\star} \left(\begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w} \right) \leq & \mathbb{I}^{\star}_{\Delta_{m,\nu}} \left(\begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w} \right) \end{aligned} \tag{4.11}$$

for all $\boldsymbol{w} \in \mathbb{R}^{\mathcal{H}}$. Therefore, an $\epsilon/2$-approximate solution to Eq. (4.8) is also an ϵ-approximate solution to Eq. (4.1). □

According to Lemma 4.1, it is enough to solve Eq. (4.8).

We summarize our proposed algorithm in Algorithm 5. This algorithm is based on Frank-Wolfe algorithms [26, 46] to problem Eq. (4.8). The most time-consuming part of Algorithm 5 is the computation of distribution $\boldsymbol{d}_t$ in Line 3. For a given vector $|(A\boldsymbol{w})_i|$, the optimization problem in Line 3 can be done in $O(m \ln m)$ time. See [88] for more details.

Further, Algorithm 5 differs from the Finding Universal Strategy (FUS) algorithm proposed by Vadhan and Zheng [95]. Though we can modify the FUS algorithm to solve our problem, it is computationally expensive in the projection step, while our algorithm uses an effective projection as in Line 3. Furthermore, our algorithm terminates as soon as the condition in Line 6 is satisfied, while the FUS algorithm has no stopping criterions.

Algorithm 5 A regression boosting algorithm for insensitive loss

Input: Training set $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m \subset \mathcal{X} \times \mathcal{Y}$,

a γ-weak learner $\mathcal{W}_\gamma : \Delta_m \to \mathcal{H}$, and parameters $\nu > 0$ and $\epsilon > 0$.

Set $A \in \mathbb{R}^{m \times \mathcal{H}}$ as $A_{i,h} = y_i - h(\boldsymbol{x}_i)$, and $\boldsymbol{w} = \boldsymbol{0}$.
for $t = 0, 1, \ldots, T$ **do**
  Compute the distribution $\boldsymbol{d}_t = \arg\max_{\boldsymbol{d}} \sum_{i=1}^m d_i |(A\boldsymbol{w})_i| - f(\boldsymbol{d})$.
  Obtain a hypothesis $h_{t+1} \leftarrow \mathcal{W}_\gamma(\boldsymbol{d}_t)$.
  $\epsilon_t := f^\star \left(\begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w}_t \right) - \max_{q=0,1,\ldots,t} \sum_i \boldsymbol{d}_{q,i} |A_{i,h_q}|$.
  if $\epsilon_t \leq \epsilon/2$ **then**
    Set $T = t$, **break**.
  end if
  Update $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{2}{t+2}(\boldsymbol{e}_{h_{t+1}} - \boldsymbol{w}_t)$.
end for

Output: Combined hypothesis $\sum_{h \in \mathcal{H}} w_{T,h} h$.

Theorem 4.1 (A convergence rate for Algorithm 5)**.** *Let $\eta \geq \frac{2}{\epsilon} \ln \frac{m}{\nu}$. Under the γ-weak learnability assumption, Algorithm 5 outputs a combined hypothesis $H = \sum_{h \in \mathcal{H}} w_{T,h} h$ satisfying*

$$\max_{\boldsymbol{d} \in \Delta_{m,\nu}} \sum_{i=1}^m d_i |y_i - H(\boldsymbol{x}_i)| \leq \gamma + \epsilon$$

in $T = O(b^2 \ln(m/\nu)/\epsilon^2)$ iterations.

Note that if the weak learner always returns the best hypothesis, that is, a hypothesis satisfying

$$h_t \in \arg\min_{h \in \mathcal{H}} \sum_{i=1}^m d_{t,i} |y_i - h(\boldsymbol{x}_i)|,$$

Theorem 4.1 guarantees the algorithm to find an ϵ-approximate solution to Eq. (4.1). To prove Theorem 4.1, we use Lemma 2.4. Since the Hessian of f at $\boldsymbol{d}$ is $\nabla^2 f(\boldsymbol{d}) = \frac{1}{\eta} \operatorname{diag}(1/d_1, 1/d_2, \ldots, 1/d_m)$, the minimal eigenvalue is at least $1/\eta$, which immediately implies $1/\eta$-strongly convexity of f w.r.t. ℓ_1-norm. Thus, applying Lemma 2.4, we can say that $f^\star$ is η-smooth w.r.t. ℓ_∞-norm.

Now we move on to the proof of Theorem 4.1. Note that this analysis slightly generalizes the one for Frank-Wolfe algorithms; Frank-Wolfe algorithms assume an oracle

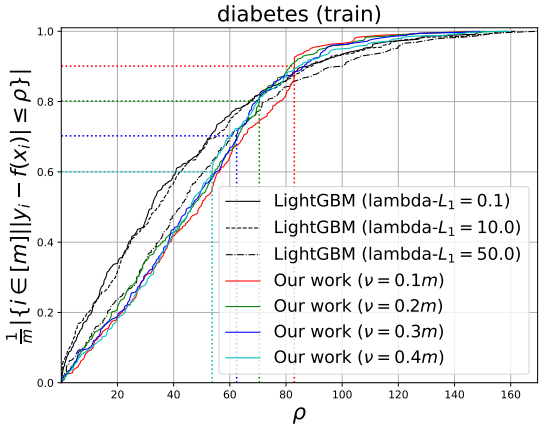

Figure 4.2: Absolute losses and the number of training examples whose losses are at most ρ. As this figure shows, Algorithm 5 tends to output a hypothesis with a small error for the farthest example.

that always returns an extreme point maximizing the inner porduct to the given gradient vector, which corresponds to the weak learner that returns the best hypothesis for our case. Further, the stopping criterion defined from line 6 to 8 in Algorithm Algorithm 5 is better than those in the Frank-Wolfe algorithms.

Proof of Theorem 4.1. Let ϵ_t be the optimality gap defined in line 5 of Algorithm 5 and let $\lambda_t = \frac{2}{t+2}$. We first show that $\epsilon_t \leq \epsilon/2$ implies the ϵ-optimality of Eq. (4.1).

By the γ-weak learnability assumption and our choice of η,

$$\epsilon_t = f^\star\left(\begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w}_t\right) - \max_{q=0,1,\ldots,t} \sum_{i=1}^{m} \boldsymbol{d}_{q,i} |A_{i,h_q}| \geq f^\star\left(\begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w}_t\right) - \gamma$$

Combining the above inequality with Eq. (4.11), we have

$$\epsilon_t \geq \mathbb{I}^\star_{\Delta_{m,\nu}}\left(\begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w}\right) - \gamma - \frac{\epsilon}{2}$$

Using $\epsilon_t \leq \epsilon/2$ yields the ϵ-optimality of Eq. (4.1).

Now we prove the convergence rate for $\{\epsilon_t\}_t$. By definition of $\boldsymbol{w}_{t+1}$ and the smoothness

of $f^\star$, we get

$$\begin{aligned}
\epsilon_t - \epsilon_{t+1} \geq & f^\star\left(\begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w}_t\right) - f^\star\left(\begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w}_{t+1}\right) \\
\geq & -\lambda_t \begin{bmatrix} \boldsymbol{d}_t^+ \\ \boldsymbol{d}_t^- \end{bmatrix}^\top \begin{bmatrix} A \\ -A \end{bmatrix} (\boldsymbol{e}_{h_{t+1}} - \boldsymbol{w}_t) - \frac{\eta}{2}\lambda_t^2 \left\| \begin{bmatrix} A \\ -A \end{bmatrix} (\boldsymbol{e}_{h_{t+1}} - \boldsymbol{w}_t) \right\|_\infty^2 \\
\geq & -\lambda_t \begin{bmatrix} \boldsymbol{d}_t^+ \\ \boldsymbol{d}_t^- \end{bmatrix}^\top \begin{bmatrix} A \\ -A \end{bmatrix} (\boldsymbol{e}_{h_{t+1}} - \boldsymbol{w}_t) - 2\lambda_t^2 \eta b^2 \\
\geq & -\lambda_t \begin{bmatrix} \boldsymbol{d}_t^+ \\ \boldsymbol{d}_t^- \end{bmatrix}^\top \begin{bmatrix} A \\ -A \end{bmatrix} (\boldsymbol{e}_{h_{t+1}} - \boldsymbol{w}_t) - 2\lambda_t^2 \eta b^2 \\
\geq & \lambda_t \left(\begin{bmatrix} \boldsymbol{d}_t^+ \\ \boldsymbol{d}_t^- \end{bmatrix}^\top \begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w}_t - \gamma \right) - 2\lambda_t^2 \eta b^2 \\
\geq & \lambda_t \left(\begin{bmatrix} \boldsymbol{d}_t^+ \\ \boldsymbol{d}_t^- \end{bmatrix}^\top \begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w}_t - f\left(\begin{bmatrix} \boldsymbol{d}_t^+ \\ \boldsymbol{d}_t^- \end{bmatrix}\right) - \gamma \right) - 2\lambda_t^2 \eta b^2,
\end{aligned}$$

where we use the non-negativity of f for the last inequality. Since $\begin{bmatrix} \boldsymbol{d}_t^+ & \boldsymbol{d}_t^- \end{bmatrix}$ is the maximizer of Eq. (4.9), we can write

$$f^\star\left(\begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w}_t\right) = \begin{bmatrix} \boldsymbol{d}_t^+ \\ \boldsymbol{d}_t^- \end{bmatrix}^\top \begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w}_t - f\left(\begin{bmatrix} \boldsymbol{d}_t^+ \\ \boldsymbol{d}_t^- \end{bmatrix}\right).$$

Therefore,

$$\epsilon_t - \epsilon_{t+1} \geq \lambda_t \left[f^\star\left(\begin{bmatrix} A \\ -A \end{bmatrix} \boldsymbol{w}_t\right) - \gamma \right] - 2\lambda_t^2 \eta b^2 \geq \lambda_t \epsilon_t - 2\lambda_t^2 \eta b^2.$$

Rearranging the terms, we obtain the recursion

$$\epsilon_{t+1} \leq \frac{t}{t+2}\epsilon_t + 2\left(\frac{2}{t+2}\right)^2 \eta b^2 \quad \text{for all } t \geq 0. \tag{4.12}$$

Using Lemma 2.1, we have $\epsilon_t \leq \frac{8\eta b^2}{t+2}$. Plugging our choice of η into the inequality,

$$\epsilon_t \leq \frac{16 b^2 \ln(m/\nu)}{\epsilon} \frac{1}{t+2}.$$

Thus, $\epsilon_t \leq \epsilon/2$ holds in $T = \frac{32 b^2}{\epsilon^2} \ln \frac{m}{\nu}$ iterations, which concludes the proof. □

4.4 Experiment

We demonstrate 5-fold cross validation for our work[1] and LightGBM [49]. LightGBM is one of the state-of-the-art boosting algorithms used for wide range of applications, including classification and regression. Gradient boosting algorithms, such as LightGBM, are known to converge in polynomial iterations if the objective function is smooth [36]. Since our objective function in Eq. (4.1) is not smooth and not supported in LightGBM, we used the setting described in Table 4.1. We use the weak learner that returns a best regression tree of depth 1. We ran Algorithm 5 over the parameters $\nu \in \{0.1m, 0.2m, \ldots, 0.4m\}$. For LightGBM, we used the setting in Table 4.1. Fig. 4.2 shows a ROC-like curve for

Table 4.1: LightGBM setting

Parameter	Value(s)
objective	regression_l1
boosting	gbdt
max_depth	1
min_data_in_leaf	1
num_leaves	4
deterministic	True
lambda_l1	0.1, 10, 50, 100
min_data_in_bin	1
enable_bundle	False
max_bin	# of training examples

diabetes dataset[2], obtained from `sklearn.datasets.load_diabetes()` in Python. Since our formulation regards the loss of sufficiently close points as zero, our algorithm tends to have large errors for the closest points compared to LightGBM. On the other hand, our formulation forces the loss to be small uniformly so that our algorithm wins at some point. For example, the solid red line in Fig. 4.2 corresponds to our algorithm with parameter $\nu = 0.1m$, which wins around $|y - f(\boldsymbol{x})| = 80$. This means that 90% of examples have errors at most $|y - f(\boldsymbol{x})| = 80$. The rest 10% of examples are regarded as outliers.

[1]The code is available at `https://github.com/rmitsuboshi/insensitive_regression_boosting`.

[2]One can obtain the original dataset from `https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html`.

Chapter 5

Extended Formulations via Decision Diagrams

So far, we have considered boosting algorithms via Frank-Wolfe. From this chapter, we consider quite different problems.

The results in this chapter appear in [53].

Large-scale optimization tasks appear in many areas such as machine learning, operations research, and engineering. Time/memory-efficient optimization techniques are more in demand than ever. Various approaches have been proposed to efficiently solve optimization problems over huge data, e.g., stochastic gradient descent methods [23] and concurrent computing techniques using GPUs [74]. Among them, we focus on the "computation on compressed data" approach, where we first compress the given data somehow and then employ an algorithm that works directly on the compressed data (i.e., without decompressing the data) to complete the task, in an attempt to reduce computation time and/or space. Algorithms on compressed data are mainly studied in string processing [34, 43, 56, 57, 79], enumeration of combinatorial objects [61], and combinatorial optimization [5]. In particular, in the work on combinatorial optimization, they compress the set of feasible solutions that satisfy given constraints into a decision diagram so that minimizing a linear objective can be done by finding the shortest path in the decision diagram. Although we can find the optimal solution very efficiently when the size of the decision diagram is small, the method can only be applied to specific types of discrete optimization problems where the feasible solution set is finite, and the objective function is linear.

Whereas, we mainly consider a more general form of discrete/continuous optimization

problems that include linear constraints with integer coefficients:

$$\min_{\boldsymbol{x}\in\mathcal{X}\subset\mathbb{R}^n} f(\boldsymbol{x}) \quad \text{s.t.} \quad A\boldsymbol{x} \geq \boldsymbol{b} \tag{5.1}$$

for some $A \in \mathcal{C}^{m\times n}$ and $\boldsymbol{b} \in \mathcal{C}^m$, where $\mathcal{X}$ denotes the constraints other than $A\boldsymbol{x} \geq \boldsymbol{b}$, and $\mathcal{C}$ is a finite subset of integers. This class of problems includes LP, QP, SDP, and MIP with linear constraints of integer coefficients. So our target problem is fairly general. Without loss of generality, we assume $m > n$, and we are particularly interested in the case where m is huge.

In this paper, we propose a pre-processing method that "rewrites" integer-valued linear constraints with equivalent but more concise ones. More precisely, we propose a general algorithm that, when given an integer-valued constraint matrix $(A, \boldsymbol{b}) \in \mathcal{C}^{m\times n} \times \mathcal{C}^m$ of an optimization problem in Eq. (5.1), produces a matrix $(A', \boldsymbol{b}') \in \mathcal{C}^{m'\times(n+n')} \times \mathcal{C}^{m'}$ that represents its extended formulation, that is, it holds that

$$\exists \boldsymbol{s} \in \mathbb{R}^{n'}, A' \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{s} \end{bmatrix} \geq \boldsymbol{b}' \Leftrightarrow A\boldsymbol{x} \geq \boldsymbol{b}$$

for some n' and m', with the hope that the size of $(A', \boldsymbol{b}')$ is much smaller than that of $(A, \boldsymbol{b})$ even at the cost of adding n' extra variables. Using the extended formulation, we obtain an equivalent optimization problem to Eq. (5.1):

$$\min_{\boldsymbol{x}\in\mathcal{X}\subset\mathbb{R}^n, \boldsymbol{s}\in\mathbb{R}^{n'}} f(\boldsymbol{x}) \quad \text{s.t.} \quad A' \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{s} \end{bmatrix} \geq \boldsymbol{b}'. \tag{5.2}$$

Then, we can apply any existing generic solvers, e.g., MIP/QP/LP solvers if f is linear or quadratic, to Eq. (5.2), combined with our pre-processing method, which may significantly reduce the computation time/space than applying them to the original problem Eq. (5.1).

To obtain a matrix $(A', \boldsymbol{b}')$, we first construct a variant of a decision diagram called a Non-Deterministic Zero-Suppressed Decision Diagram (NZDD, for short) [31] that somehow represents the matrix $(A, \boldsymbol{b})$. Observing that the constraint $A\boldsymbol{z} \geq \boldsymbol{b}$ can be restated in terms of the NZDD constructed as "every path length is lower bounded by 0" for an appropriate edge weighting, we establish the extended formulation $(A', \boldsymbol{b}') \in \mathcal{C}^{m'\times(n+n')} \times \mathcal{C}^{m'}$ with $m' = |E|$ and $n' = |V|$, where V and E are the sets of vertices and edges of the NZDD, respectively. One of the advantages of the result is that the size of the resulting optimization problem depends only on the size of the NZDD and the number n of variables, but *not* on the number m of the constraints in the original problem. Therefore, if

Table 5.1: Characteristics of previous work on optimization with decision diagrams (DDs) and ours.

	Coeff. of lin. consts.	Variables	Objectives	DDs
Prev. work	Any type	Binary/Integer	Linear	Feasible solutions
Ours	Binary/Integer	Any type	Any type	Lin. consts.

the matrix $(A, \boldsymbol{b})$ is well compressed into a small NZDD, then we obtain an equivalent but concise optimization problem in Eq. (5.2).

To clarify the differences between our work and previous work regarding optimization using decision diagrams, we summarize the characteristics of both results in Table 5.1. Notable differences are that (i) ours can treat optimization problems with any types of variables (discrete, or real), any types of objectives (including linear ones) but with integer coefficients on linear constraints, and (ii) ours uses decision diagrams for representing linear constraints while previous work uses them for representing feasible solutions of particular classes of problems. So, for particular classes of discrete optimization problems, the previous approach would work better with specific construction methods for decision diagrams. On the other hand, ours is suitable for continuous optimization problems or/and discrete optimization problems for which efficient construction methods for decision diagrams representing feasible solutions are not known. See the later section for more detailed descriptions of related work.

Then, to realize succinct extended formulations, we propose practical heuristics for constructing NZDDs, which is our third contribution. Since it is not known to construct an NZDD of small size, we first construct a ZDD of minimal size, where the ZDD is a restricted form of the NZDD representation. To this end, we use a ZDD compression software called `zcomp` [92]. Then, we give rewriting rules for NZDDs that reduce both the numbers of vertices and edges, and apply them to obtain NZDDs of smaller size of V and E. Although the rules may increase the size of NZDDs (i.e., the total number of edge labels), the rules seem to work effectively since reducing $|V|$ and $|E|$ is more important for our purpose.

Experimental results on synthetic and real data sets show that our algorithms improve time/space efficiency significantly, especially when (i) $m \gg n$, and (ii) the set $\mathcal{C}$ of integer coefficients is small, e.g., binary, where the datasets tend to have concise NZDD

representations.

5.1 Related work

Various computational tasks over compressed strings or texts are investigated in algorithms and data mining literature, including, e.g., pattern matching over strings and computing edit distances or q-grams [34, 43, 56, 57, 79]. The common assumption is that strings are compressed using the straight-line program, which is a class of context-free grammars generating only one string (e.g., LZ77 and +LZ78). As notable applications of string compression techniques to data mining and machine learning, Nishino et al. [69] and Tabei et al. [90] reduce the space complexity of matrix-based computations. So far, however, string compression-based approaches do not seem to be useful for representing linear constraints.

Decision diagrams are used in the enumeration of combinatorial objects, discrete optimization and so on. In short, a decision diagram is a directed acyclic graph with a root and a leaf, representing a subset family of some finite ground set Σ or, equivalently, a boolean function. Each root-to-leaf path represents a set in the set family. The Binary Decision Diagram (BDD) [10, 52] and its variant, the Zero-Suppressed Binary Decision Diagram (ZDD) [52, 60], are popular in the literature. These support various set operations (such as intersection and union) in efficient ways. Thanks to the DAG structure, linear optimization problems over combinatorial sets $\mathcal{X} \subset \{0,1\}^n$ can be reduced to shortest/longest path problems over the diagrams representing $\mathcal{X}$. This reduction is used to solve the exact optimization of NP-hard combinatorial problems (see, e.g., [5, 6, 14, 44, 68]) and enumeration tasks [61, 62, 63]. Among works on decision diagrams, the work of Fujita et al. [31] would be closest to ours. They propose a variant of ZDD called the Non-deterministic ZDD (NZDD) to represent labeled instances and show how to emulate the boosting algorithm AdaBoost* [76], a variant of AdaBoost [28] that maximizes the margin, over NZDDs. We follow their NZDD-based representation of the data. But our work is different from Fujita et al. in that, they propose specific algorithms running over NZDDs, whereas our work presents extended formulations based on NZDDs, which could be used with various algorithms.

The notion of extended formulation arises in combinatorial optimization (e.g., [20, 103]). The idea is to re-formulate a combinatorial optimization with an equivalent different form, so that the size of the problem is reduced. For example, a typical NP-hard

combinatorial optimization problem has an integer programming formulation of exponential size. Then a good extended formulation should have a smaller size than the exponential. Typical work on extended formulation focuses on some characterization of the problem to obtain succinct formulations (see, e.g., [25]). Our work is different from these in that we focus on the redundancy of the data and try to obtain succinct extended formulations for optimization problems described with data.

5.2 Non-deterministic ZDD

The non-deterministic Zero-suppressed Decision Diagram (NZDD) [31] is a variant of the Zero-suppressed Decision Diagram(ZDD) [52, 60], representing subsets of some finite ground set Σ. More formally, NZDD is defined as follows.

Definition 5.1 (NZDD)**.** *An NZDD G is a quadruple $G = (V, E, \Sigma, \Phi)$, where (V, E) is a directed acyclic graph (V and E are the sets of nodes and edges, respectively) with a single root with no-incoming edges and a leaf with no outgoing edges, Σ is the ground set, and $\Phi : E \to 2^{\Sigma}$ is a function assigning each edge e a subset $\Phi(e)$ of Σ. More precisely, we allow (V, E) to be a multigraph, i.e., two nodes can be connected with more than one edge.*

Furthermore, an NZDD G satisfies the following additional conditions. Let $\mathcal{P}_G$ be the set of paths in G starting from the root to the leaf, where each path $P \in \mathcal{P}_G$ is represented as a subset of E, and for any path $P \in \mathcal{P}_G$, we abuse the notation and let $\Phi(P) = \bigcup_{e \in P} \Phi(e)$.

1. For any path $P \in \mathcal{P}_G$ and any edges $e, e' \in P$, $\Phi(e) \cap \Phi(e') = \emptyset$. That is, for any path P, an element $a \in \Sigma$ appears at most once in P.

2. For any paths $P, P' \in \mathcal{P}_G$, $\Phi(P) \neq \Phi(P')$. Thus, each path P represents a different subset of Σ.

Then, an NZDD G naturally corresponds to a subset family of Σ. Formally, let $L(G) = \{\Phi(P) \mid P \in \mathcal{P}_G\}$. Fig. 5.1 illustrates an NZDD representing a subset family.

A ZDD [60, 52] can be viewed as a special form of NZDD $G = (V, E, \Sigma, \Phi)$ satisfying the following properties: (i) For each edge $e \in E$, $\Phi(e) = \{a\}$ for some $a \in \Sigma$ or $\Phi(e) = \emptyset$. (ii) Each internal node has at most two outgoing edges. If there are two edges, one is labeled with $\{a\}$ for some $a \in \Sigma$ and the other is labeled with $\emptyset$. (iii) There is a total

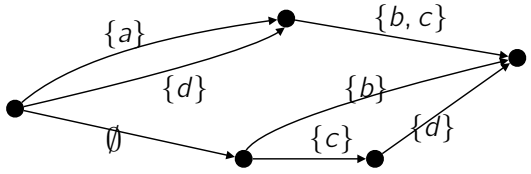

Figure 5.1: An NZDD representing $\{\{a, b, c\}, \{b\}, \{b, c, d\}, \{c, d\}\}$.

order over Σ such that, for any path $P \in \mathcal{P}_G$ and for any $e, e' \in P$ labeled with singletons $\{a\}$ and $\{a'\}$ respectively, if e is an ancestor of e', a precedes a' in the order.

We believe that that constructing a minimal NZDD for a given subset family is NP-hard since closely related problems are NP-hard. For example, constructing a minimal ZDD (over all orderings of Σ) is known to be NP-hard [52], and construction of a minimal NFA which is equivalent to a given DFA is P-space hard [47]. On the other hand, there is a practical construction algorithm of ZDDs given a subset family and a fixed order over Σ using multi-key quicksort [92].

5.3 NZDDs for linear constraints with binary coefficients

In this section, we show an NZDD representation for linear constraints in Eq. (5.1) when linear constraints have $\{0, 1\}$-valued coefficients, that is, $\mathcal{C} = \{0, 1\}$. We will discuss its extensions to integer coefficients in the later section. Let $\boldsymbol{a}_i \in \{0, 1\}^n$ be the vector corresponding to the i-th row of the matrix $A \in \{0, 1\}^{m \times n}$ (for $i \in [m]$). For $\boldsymbol{x} \in \{0, 1\}^n$, let $\mathrm{idx}(\boldsymbol{x}) = \{j \in [n] \mid x_j \neq 0\}$, i.e., the set of indices of nonzero components of $\boldsymbol{x}$. Then, we define $I = \{\mathrm{idx}(\boldsymbol{c}_i) \mid \boldsymbol{c}_i = (\boldsymbol{a}_i, b_i), i \in [m]\}$. Note that I is a subset family of $2^{[n+1]}$. Then we assume that we have some NZDD $G = (V, E, [n + 1], \Phi)$ representing I, that is, $L(G) = I$. We will later show how to construct NZDDs.

The following theorem shows the equivalence between the original problem in Eq. (5.1) and a problem described with the NZDD G.

Theorem 5.1. *Let $G = (V, E, [n + 1], \Phi)$ be an NZDD such that $L(G) = I$. Then the*

following optimization problem is equivalent to problem in Eq. (5.1)*:*

$$\begin{aligned} \min_{\boldsymbol{x}\in X\subset\mathbb{R}^n, \boldsymbol{s}\in\mathbb{R}^{|V|}} \quad & f(\boldsymbol{x}) \\ s.t. \quad & s_{e.u} + \sum_{j\in\Phi(e)} x'_j \geq s_{e.v}, \quad \forall e \in E, \\ & s_{\mathrm{root}} = 0,\ s_{\mathrm{leaf}} = 0, \\ & \boldsymbol{x}' = (\boldsymbol{x}, -1), \end{aligned} \tag{5.3}$$

where e.u and e.v are nodes that the edge e is directed from and to, respectively.

Before going through the proof, let us explain some intuition on problem in Eq. (5.3). Intuitively, each linear constraint in Eq. (5.1) is encoded as a path from the root to the leaf in the NZDD G, and a new variable s_v for each node v represents a lower bound of the length of the shortest path from the root to v. The inequalities in Eq. (5.3) reflect the structure of the standard dynamic programming of Dijkstra, so that all inequalities are satisfied if and only if the length of all paths is larger than zero. In Fig. 5.2, we show an illustration of the extended formulation.

Proof. Let $\boldsymbol{x}_\star$ and $(\hat{\boldsymbol{x}}', \hat{\boldsymbol{s}})$ be the optimal solutions of problems Eq. (5.1) and Eq. (5.3), respectively. It suffices to show that each optimal solution can construct a feasible solution of the other problem.

Let $\hat{\boldsymbol{x}}$ be the vector consisting of the first n components of $\hat{\boldsymbol{x}}'$. For each constraint $\boldsymbol{a}_i^\top \boldsymbol{x} \geq b_i$ ($i \in [m]$) in Eq. (5.1), there exists the corresponding path $P_i \in \mathcal{P}_G$. By repeatedly applying the first constraint in Eq. (5.3) along the path P_i, we have $\sum_{e\in P_i}\sum_{j\in\Phi(e)} \hat{x}'_j \geq \hat{s}_{\mathrm{leaf}} = 0$. Further, since $\Phi(P_i)$ represents the set of indices of nonzero components of $\boldsymbol{c}_i$, $\sum_{e\in P_i}\sum_{j\in\Phi(e)} \hat{x}'_j = \boldsymbol{c}_i^\top \hat{\boldsymbol{x}}' = \boldsymbol{a}_i^\top \hat{\boldsymbol{x}} - b_i$. By combining these inequalities, we have $\boldsymbol{a}_i^\top \hat{\boldsymbol{x}} - b_i \geq 0$. This implies that $\hat{\boldsymbol{x}}$ is a feasible solution of Eq. (5.1) and thus $f(\boldsymbol{x}_\star) \leq f(\hat{\boldsymbol{x}})$.

Let $\boldsymbol{x}'_\star = (\boldsymbol{x}_\star, -1)$. Assuming a topological order on V (from the root to the leaf), we define $s_{\star,\mathrm{root}} = s_{\star,\mathrm{leaf}} = 0$ and $s_{\star,v} = \min_{e\in E, e.v=v} s_{\star,e.u} + \sum_{j\in\Phi(e)} x'_{\star,j}$ for each $v \in V \setminus \{\mathrm{root}, \mathrm{leaf}\}$. Then, we have, for each $e \in E$ s.t. $e.v \neq \mathrm{leaf}$, $s_{\star,e.v} \leq s_{\star,e.u} + \sum_{j\in\Phi(e)} x'_{\star,j}$ by definition. Now, $\min_{e\in E, e.v=\mathrm{leaf}} s_{\star,e.u} + \sum_{j\in\Phi(e)} x'_{\star,j}$ is achieved by a path $P \in \mathcal{P}_G$ corresponding to $\arg\min_{i\in[m]} \boldsymbol{a}_i^\top \boldsymbol{x}_\star - b_i$, which is non-negative since $\boldsymbol{x}_\star$ is feasible w.r.t. Eq. (5.1). Therefore, $s_{\star,e.v} \leq s_{\star,e.u} + \sum_{j\in\Phi(e)} x'_{\star,j}$ for $e \in E$ s.t. $e.v = \mathrm{leaf}$ as well. Thus, $(\boldsymbol{x}'_\star, s_\star)$ is a feasible solution of Eq. (5.3) and $f(\hat{\boldsymbol{x}}) \leq f(\boldsymbol{x}_\star)$. □

Given the NZDD $G = (V, E)$, Eq. (5.3) contains $n + |V| - 2$ variables and $|E|$ linear constraints, where $|V| - 2$ variables are real. The most naive construction, where the re-

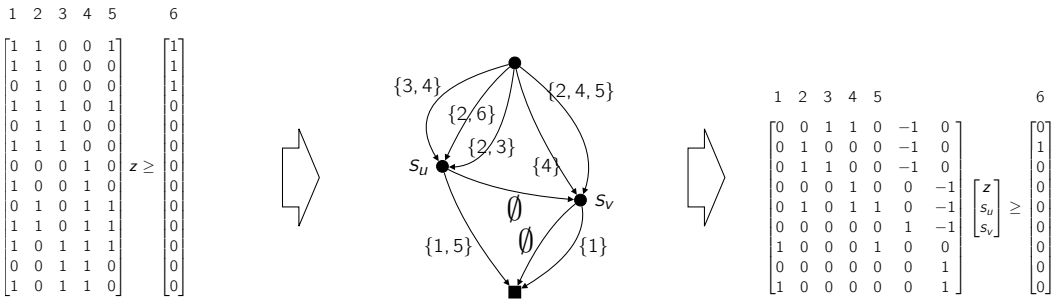

Figure 5.2: An illustration of the extended formulation. Left: Original constraints as in Eq. (5.1). Middle: A NZDD representation of the left constraints. Right: The matrix form Eq. (5.3) of the middle diagram without constant terms. This example reduces the 13 constraints to 9 constraints by adding 2 variables.

sulting NZDD contains two nodes (root and leaf), and every root-to-leaf path corresponds to a constraint, has the same problem size as the original one. Thus, the problem size cannot be worse than the original one.In particular, if Eq. (5.1) is LP or IP, then Eq. (5.3) is LP, or MIP, respectively.

5.4 Extensions to integer coefficients

We briefly discuss how to extend our NZDD representation of linear constraints to the cases where coefficients of linear constraints belong to a finite set $\mathcal{C}$ of integers. There are two ways to do so.

Binary encoding of integers We assume some encoding of integers in $\mathcal{C}$ with $O(\log |\mathcal{C}|)$ bits. Then, each bit can be viewed as a binary-valued variable. Each integer coefficient can be also recovered with its binary representation. Under this attempt, the resulting extended formulation has $O(n \log |\mathcal{C}| + |V|)$ variables and $O(|E|)$ linear constraints.

Extending Σ Another attempt is to extend the domain Σ of an NZDD $G = (V, E, \Sigma, \Phi)$. The extended domain Σ' consists of all pairs of integers in $\mathcal{C}$ and elements in Σ. Again, integer coefficients are recovered through the new domain Σ'. The resulting extended formulation has $O(n|\mathcal{C}| + |V|)$ variables and $O(|E|)$ linear constraints. While the size of the problem is larger than the binary encoding, its implementation is easy in practice and could be effective for $\mathcal{C}$ of small size.

5.5 Construction of NZDDs

We propose heuristics for constructing NZDDs given a subset family $\mathcal{S} \subseteq 2^{\Sigma}$. We use the zcomp [92, 93], developed by Toda, to compress the subset family S to a ZDD. The zcomp is designed based on multikey quicksort [4] for sorting strings. The running time of the zcomp is $O(n \log^2 |\mathcal{S}|)$, where n is an upper bound of the nodes of the output ZDD and $|\mathcal{S}|$ is the sum of cardinalities of sets in $\mathcal{S}$. Since $n \leq |\mathcal{S}|$, the running time is almost linear in the input.

A naive application of the zcomp is, however, not very successful in our experiences. We observe that the zcomp often produces concise ZDDs compared to inputs. But, concise ZDDs do not always imply concise representations of linear constraints. More precisely, the output ZDDs of the zcomp often contains (i) nodes with one incoming edge or (ii) nodes with one outgoing edge. A node v of these types introduces a corresponding variable s_v and linear inequalities. Specifically, in the case of type (ii), we have $s_v \leq \sum_{j \in \Phi(e)} z'_j + s_{e.u}$ for each $e \in E$ s.t. $e.v = v$, and for its child node v' and edge e' between v and v', $s_{v'} \leq \sum_{j \in \Phi(e')} z'_j + s_v$. These inequalities are redundant since we can obtain equivalent inequalities by concatenating them: $s_{v'} \leq \sum_{j \in \Phi(e')} z'_j + \sum_{j \in \Phi(e)} z'_j + s_{e.u}$ for each $e \in E$ s.t. $e.v = v$, where s_v is removed.

Based on the observation above, we propose a simple reduction heuristics removing nodes of type (i) and (ii). More precisely, given an NZDD $G = (V, E)$, the heuristics outputs an NZDD $G' = (V', E')$ such that $L(G) = L(G')$ and G' does not contain nodes of type (i) or (ii). The heuristics can be implemented in $O(|V'| + |E'| + \sum_{e \in E'} |\Phi(e)|)$ time by going through nodes of the input NZDD G in the topological order from the leaf to the root and in the reverse order, respectively.

5.6 ℓ_1-norm regularized soft margin optimization

The ℓ_1-norm regularized soft margin optimization is a standard linear programming formulation of finding a sparse linear classifier with large margin (see, e.g., [21, 97, 99]). We are given a set $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\} \subset \mathcal{X} \times \{-1, 1\}$ of labeled instances, where $\mathcal{X} \subset \mathbb{R}^n$ is the set of instances. For clarity, we assume that the domain is the set of binary vectors, i.e., $\mathcal{X} = \{0, 1\}^n$. This is common in many applications when we employ the bag-of-words representation of instances. Given a parameter $\nu \in (0, 1]$ and a sequence S of

labeled instances, the ℓ_1-norm regularized soft margin optimization is defined as follows[1]:

$$\begin{aligned} \max_{\rho,\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \rho - \frac{1}{\nu m}\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^\top \boldsymbol{x}_i - b) \geq \rho - \xi_i \quad \forall i = 1, 2, \ldots, m, \\ & \sum_j w_j + b = 1, \boldsymbol{w} \in \mathbb{R}^n_+, b \geq 0, \boldsymbol{\xi} \in \mathbb{R}^m_+. \end{aligned} \tag{5.4}$$

For the parameter $\nu \in (0, 1]$ and the optimal solution $(\rho^\star, \boldsymbol{w}^\star, b^\star, \boldsymbol{\xi}^\star)$, by a duality argument, it can be verified that there are at least $(1-\nu)m$ instances that has margin larger than $\rho^\star$, i.e., $y_i(\boldsymbol{w}^{\star\top}\boldsymbol{x}_i - b^\star) \geq \rho^\star$ [21].

We formulate a variant of the ℓ_1-norm reguralized soft margin optimization based on NZDDs and propose efficient algorithms. Our generic re-formulation of Eq. (5.3) can be applied to the soft margin Eq. (5.4) as well. However, a direct application is not successful since Eq. (5.4) contains $O(m)$ slack variables $\boldsymbol{\xi}$ and each ξ_i appears only once in m linear constraints. That implies a resulting NZDD contains $\Omega(m)$ edges. Therefore, we are motivated to formulate a soft margin optimization for which a succinct NZDD representation exists.

Our basic idea is as follows: Suppose that we have an NZDD $G = (V, E, \Phi, \Sigma)$ such that each path P corresponds to a constraint $y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq \rho$. Our idea is to introduce a slack variable β_e on each edge e along the path P, instead of using a slack variable ξ_i for each instance $i \in [m]$.

With these notations, we define a variant of soft margin optimization problems on an NZDD.

$$\begin{aligned} \max_{\rho,b,\boldsymbol{w},\boldsymbol{\beta}} \quad & \rho - \frac{1}{\nu m}\sum_{i=1}^{m}\sum_{e\in P_i}\beta_e \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^\top \boldsymbol{x}_i - b) \geq \rho - \sum_{e\in P_i}\beta_e, \ \forall i = 1, 2, \ldots, m, \\ & \sum_{j=1}^{n} w_j + b = 1, \quad b \geq 0, \ \boldsymbol{w} \in \mathbb{R}^n_+, \ \boldsymbol{\beta} \in \mathbb{R}^E_+. \end{aligned} \tag{5.5}$$

Note that the sum of the slack variables $\sum_{e\in P_i}\beta_e$ for each instance $\boldsymbol{x}_i$ is more restricted than the original slack variable ξ_i. This observation implies the following.

[1]For the sake of simplicity, we show a restricted version of the formulation. We can easily extend the formulation with positive and negative weights by considering positive and negative weights w_j^+ and w_j^- instead of w_j ad replacing w_j with $w_j^+ - w_j^-$.

Proposition 5.1. *An optimal solution of Eq.* (5.5) *is a feasible solution of Eq.* (5.4).

Although problem Eq. (5.5) is a restricted version of Eq. (5.4), we observe that this restriction does not decrease the generalization ability in our experiments, which is shown later.

Now we introduce an equivalent formulation of Eq. (5.5) which is fully described with an NZDD. To do so, we specify how to construct the input NZDD as follows.

NZDD G representing the sample S. Let $I^+ = \{\mathrm{idx}((\boldsymbol{x}_i, 1)) \in 2^{[n+1]} \mid y_i = 1, i \in [m]\}$ and $I^- = \{\mathrm{idx}((\boldsymbol{x}_i, 1)) \in 2^{[n+1]} \mid y_i = -1, i \in [m]\}$ be the set families of indices of nonzero components of positive and negative instances, respectively. For I^+ and I^-, let $G^+ = (V^+, E^+, [n+1], \Phi^+)$ and $G^- = (V^-, E^-, [n+1], \Phi^-)$, be corresponding NZDDs such that $L(G^+) = I^+$ and $L(G^-) = I^-$, respectively. Finally, we connect G^+ and G^- in parallel; We denote the resulting NZDD as $G = (V, E, [n+1], \Phi)$, where $V = V^+ \cup V^- \cup \{\mathrm{root}, \mathrm{leaf}\} \setminus \{\mathrm{leaf}^+, \mathrm{leaf}^-\}$ and $E = E^+ \cup E^- \cup \{(\mathrm{root}, \mathrm{root}^+), (\mathrm{root}, \mathrm{root}^-)\}$, such that (i) two leaf nodes $\mathrm{leaf}^+, \mathrm{leaf}^-$ of G^+ and G^- are merged into a new leaf node leaf, (ii) there are two edges e from a new root node root to root nodes of G^+ and G^- with $\Phi(e) = \emptyset$, and (iii) for other edges $e \in E$, $\Phi(e) = \Phi^+(e)$ if $e \in E^+$ and $\Phi(e) = \Phi^-(e)$ if $e \in E^-$.

$$
\begin{aligned}
\max_{\rho, \boldsymbol{w}, \boldsymbol{\beta}, \boldsymbol{s}} \quad & \rho - \frac{1}{\nu m} \sum_{e \in E} m_e \beta_e \\
\text{s.t.} \quad & s_{e.u} + \mathrm{sgn}(e) \sum_{j \in \Phi(e)} w_j + \beta_e \geq s_{e.v}, \quad \forall e \in E, \\
& s_{\mathrm{root}} = 0, \ s_{\mathrm{leaf}} \geq \rho, \\
& \sum_{j=1}^{n} w_j - w_{n+1} = 1, \\
& w_j \geq 0, \ i \in [n], w_{n+1} \leq 0, \boldsymbol{\beta} \geq \boldsymbol{0},
\end{aligned}
\tag{5.6}
$$

where m_e is the number of paths going thorough the edge e, and $\mathrm{sgn}(e)$ is defined as $\mathrm{sgn}(e) = -1$ if $e \in E^-$ and $\mathrm{sgn}(e) = 1$, otherwise. The constants m_e can be computed in time $O(|E|)$ a priori by a dynamic programming over G. Note that the bias term $-b$ correspond to w_{n+1} for notatinal convenience. Then, by following the same proof argument of Theorem 5.1, we have the following corollary.

Corollary 5.1. *Eq.* (5.6) *is equivalent to Eq.* (5.5).

Algorithm 6 Column Generation

Input: NZDD $G = (V, E, \Sigma, \Phi)$

1: Let $\boldsymbol{d}_1 \in [0,1]^{|E|}$ be any vector satisfying Eqs. (5.9) to (5.11). Let $J_0 = \emptyset$.
2: **for** $t = 1, 2, ...$ **do**
3: Let $j_t = \arg\max_{j \in [n+1]} \operatorname{sgn}(j) \sum_{e: j \in \Phi(e)} \operatorname{sgn}(e) d_e$ and let $\hat{\gamma}_t$ be its objective value.
4: **if** $\hat{\gamma}_t \leq \gamma_t + \epsilon$ **then**
5: Set $T = t - 1$ and **break**.
6: **end if**
7: Let $J_t = J_{t-1} \cup \{j_t\}$ and update $(\boldsymbol{d}_{t+1}, \gamma_{t+1})$ as:

$$\begin{aligned} \min_{\boldsymbol{d}, \gamma} \ & \gamma \\ \text{s.t.} \ & \operatorname{sgn}(j) \sum_{e: j \in \Phi(e)} \operatorname{sgn}(e) d_e \leq \gamma, \forall j \in J_t, \\ & \text{constraints Eqs. (5.9) to (5.11).} \end{aligned} \tag{5.7}$$

8: **end for**

Output: the Lagrangian coefficients $(\boldsymbol{w}_T, \boldsymbol{\beta}_T, \rho_T)$ for the subproblem w.r.t. J_T.

Eq. (5.6) has $O(n + |V| + |E|)$ variables and $O(|E|)$ linear constraints, whereas the original formulation Eq. (5.4) has $O(n + m)$ variables and $O(m)$ linear constraints. So, with a concise NZDD representation of the sample, we obtain an extended formulation whose size is independent of m of the sample size. In later subsections, we propose two efficient solving methods for problem Eq. (5.6).

5.6.1 Column Generation

In this subsection, we propose a column generation-based method for solving the modified ℓ_1-norm soft margin optimization problem in Eq. (5.6). Although the size of the extended formulation Eq. (5.6) does not depend on the size of linear constraints m of the original problem, it still depends on n, the size of variables. The column generation is a standard approach of linear programming that tries to reduce either the size of linear constraints/variables by solving smaller subproblems. In our case, we try to avoid problems depending on the size of variables n.

By a standard dual argument of linear programming, the equivalent dual problem

of Eq. (5.6) is given as follows.

$$\min_{\gamma, \boldsymbol{d}} \gamma \tag{5.8}$$

$$\text{s.t.} \quad \mathrm{sgn}(j) \sum_{e: j \in \Phi(e)} \mathrm{sgn}(e) d_e \leq \gamma, \quad \forall j \in [n+1],$$

$$\sum_{e: e.u = u} d_e = \sum_{e: e.v = u} d_e, \ \forall u \in V \setminus \{\mathrm{root}, \mathrm{leaf}\}, \tag{5.9}$$

$$\sum_{e: e.u = \mathrm{root}} d_e = 1, \ \sum_{e: e.v = \mathrm{leaf}} d_e = 1, \tag{5.10}$$

$$0 \leq d_e \leq \frac{m_e}{\nu m} \quad (e \in E), \tag{5.11}$$

where $\mathrm{sgn}(j) = 1$ for $j \in [n]$ and $sign(j) = -1$ for $j = n+1$. Here, the dual problem Eq. (5.6) has $|E| + 1$ variables and n linear constraints. Roughly speaking, this problem is to find a vector $\boldsymbol{d}$ that represents a "flow" from the root to the leaf in the NZDD optimizing some objective, where the total flow is 1. The objective is γ, the upper bound of $\mathrm{sgn}(j) \sum_{e: j \in \Phi(e)} \mathrm{sgn}(e) d_e$ for each $j \in [n+1]$. The column generation-based algorithm is given in Algorithm 6. The algorithm repeatedly solves the subproblems whose constraints related to γ are only restricted to a subset $J_t \subseteq [n+1]$. Then it adds j_{t+1} to J_t (updated as J_{t+1}), where j_{t+1} corresponds to the constraint that violates condition Eq. (5.9) the most with respect to the current solution $(\gamma_t, \boldsymbol{d}_t)$. It can be shown that the column-generation algorithm finds an ϵ-approximate solution. The details are shown in Appendix. As for its time complexity analysis, similar to other column generation techniques, we do not have non-trivial iteration bounds. In the next section, we propose another algorithm with theoretical guarantee of an iteration bound.

5.7 Performing ERLPBoost over an NZDD

We emulate ERLPBoost [99] on an NZDD. The algorithm is the same as ERLPBoost except for the update rule. Let $d_e^0 = \frac{m_e}{m}$ for all $e \in E$. In each iteration t, the compressed version of ERLPBoost solves the following sub-problem

$$\min_{\gamma, \boldsymbol{d}} \quad \gamma + \frac{1}{\eta} \left[\sum_{e \in E} d_e \ln \frac{d_e}{d_e^0} - d_e + d_e^0 \right] \tag{5.12}$$

$$\text{s.t.} \quad \mathrm{sgn}(j) \sum_{e: j \in \Phi(e)} \mathrm{sgn}(e) d_e \leq \gamma, \quad \forall j \in J_t,$$

constraints Eqs. (5.9) to (5.11).

Algorithm 7 ERLPBoost over an NZDD

Input: NZDD $G = (V, E, \Sigma, \Phi)$

Set $d_e^0 = \frac{m_e}{m}$ for all $e \in E$. Let $J_0 = \emptyset$.
for $t = 1, 2, ...$ **do**
  Find a hypothesis $j_t \in [n+1]$ that maximizes the edge w.r.t. $\boldsymbol{d}^{t-1}$.
  Set $\delta^t := \min_{1 \le q \le t} P^q(\boldsymbol{d}^{q-1}) - P^{t-1}(\boldsymbol{d}^{t-1})$.
  if $\delta^t \le \epsilon/2$ **then**
    Set $T = t - 1$ and **break**.
  end if
  Compute the minimizer $\boldsymbol{d}^t$ of Eq. (5.12).
end for
Solve Eq. (5.8) over $J_T = \{j_t\}_{t=1}^T$ to get the optimal weights $\boldsymbol{w}^T$ on hypotheses.

Output: $f = \sum_{j \in J_T} w_j^T h_j$.

Here, $\eta > 0$ is some parameter. One can rewrite Eq. (5.12) in terms of $\boldsymbol{d}$ by introducing max function. We denote the resulting objective function as $P^t(\boldsymbol{d})$. Algorithm 7 shows ERLPBoost over an NZDD. We can also obtain a similar iteration bound like ERLPBoost.

Theorem 5.2. *If* $\eta = \frac{4}{\epsilon} \operatorname{depth}(G) \max\{1, \ln \frac{1}{\nu}\}$, *then Algorithm 7 outputs an* ϵ*-approximate solution to Eq.* (5.8) *in* $T \le \frac{144}{\epsilon^2} \operatorname{depth}(G)^2 \max\left(1, \ln \frac{1}{\nu}\right)$ *iterations.*

Before proving Theorem 5.2, we look into some properties and prove some Lemmata. By definition, Algorithm 7 solves the following sub-problem;

$$\min_{\boldsymbol{d}} \max_{j \in J_t} \left\{ \operatorname{sgn}(j) \sum_{e \in E} \operatorname{sgn}(e) I_{[j \in \Phi(e)]} d_e \right\} + \frac{1}{\eta} \left[\sum_{e \in E} d_e \ln \frac{d_e}{d_e^0} - d_e + d_e^0 \right] \tag{5.13}$$

$$\text{s.t. } I_{[u=\text{root}]} + I_{[u \neq \text{root}]} \sum_{e=(\cdot,u) \in E} d_e = I_{[u=\text{leaf}]} + I_{[u \neq \text{leaf}]} \sum_{e=(u,\cdot) \in E} d_e, \quad \forall u \in V,$$

$$d_e \le \frac{m_e}{\nu m}, \quad \forall e \in E,$$

where I is the indicator function such that

$$I_{[P]} = \begin{cases} 1 & \text{if } P \text{ is true,} \\ 0 & \text{if } P \text{ is false.} \end{cases}$$

Here, we prove a stronger result in terms of weak learner; In each iteration, weak learner returns a hypothesis $j_t \in \{1, 2, \ldots, n\}$ such that

$$\operatorname{sgn}(j_t) \sum_{e \in E} \operatorname{sgn}(e) d_e^{t-1} \ge g \tag{5.14}$$

for some unknown parameter $g > 0$. This assumption is the same as the one in ERLPBoost [99]. First of all, we give two technical lemmas.

Lemma 5.1. *Let* $\mathrm{depth}(G) = \max_{P \in \mathcal{P}_G \subset 2^E} |P|$. *Let* $\boldsymbol{d} \in \mathbb{R}^{|E|}$ *be any feasible solution of Eq.* (5.13). *Then,* $\sum_{e \in E} d_e \leq \mathrm{depth}(G)$.

Proof. Let G' be a layered NZDD of G obtained by adding some redundant nodes. Since this operation only increases the number of edges, $\sum_{e \in E} d_e \leq \sum_{e \in E'} d_e$ holds, where E' is the set of edges of G'. let E'_k be the set of edges at depth k. Then,

$$\sum_{e \in E} d_e \leq \sum_{e \in E'} d_e = \sum_{k=1}^{K} \sum_{e \in E'_k} d_e = \sum_{k=1}^{K} 1 = \mathrm{depth}(G),$$

where at the second equality, we used the fact that for a feasible $\boldsymbol{d}$, the total flow at any depth equals to 1. □

The following lemma shows an upper bound of the unnormalized relative entropy for a feasible distribution.

Lemma 5.2. *Let* $\boldsymbol{d}$ *be a feasible solution to Eq.* (5.13) *for an NZDD* G. *Then, the following inequality holds.*

$$\sum_{e \in E} d_e \ln \frac{d_e}{d_e^0} - d_e + d_e^0 \leq \mathrm{depth}(G) \ln \frac{1}{\nu}$$

Proof. Without loss of generality, we can assume that the NZDD is layered. Indeed, if the NZDD is not layered, we can add dummy nodes and dummy edges to make the NZDD layered. Further, we can increase the edges on the NZDD to make $m_e = 1$ for all edge $e \in E$. Fig. 5.3 and Fig. 5.4 depict these manipulations. With these conversions, the initial distribution $\boldsymbol{d}^0$ becomes $d_e^0 = 1/m$ for all $e \in E$ and the feasible region becomes Let $E_k \subset E$ be the set of edges at depth k.

$$\begin{aligned}\sum_{e \in E} d_e \ln \frac{d_e}{d_e^0} - d_e + d_e^0 &= \sum_{k=1}^{\mathrm{depth}(G)} \left(\sum_{e \in E_k} d_e \ln \frac{d_e}{d_e^0} - d_e + d_e^0 \right) \\ &= \sum_{k=1}^{\mathrm{depth}(G)} \sum_{e \in E_k} d_e \ln \frac{d_e}{d_e^0}, \end{aligned} \tag{5.15}$$

where the last equality holds since in each layer, the total flow equals to one. For each edge $e \in E$ with $m_e > 1$, define a set $\hat{E}(e)$ of duplicated edges of size $|\hat{E}(e)| = m_e$. Further, define

$$\forall \hat{e} \in \hat{E}(e), \quad \hat{d}_{\hat{e}} = \frac{d_e}{m_e}, \quad \hat{d}^0_{\hat{e}} = \frac{d_e^0}{m_e} = \frac{1}{\nu m}.$$

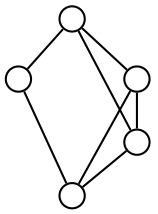

Figure 5.3: A conversion from an NZDD to a layered NZDD. The added nodes and edges are depicted with dotted lines.

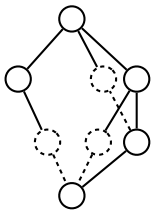

Figure 5.4: Edge duplication for an edge $e \in E$, i.e., duplicating the edge to the parallel edges of size m_e. With this manipulation, $m_{\hat{e}_k}$ becomes one for all duplicated edges $\hat{e}_k \in \hat{E}(e)$.

Note that this manipulation does not change the depth of the original NZDD.

Then,

$$\sum_{\hat{e} \in \hat{E}(e)} \hat{d}_{\hat{e}} \ln \frac{\hat{d}_{\hat{e}}}{\hat{d}^0_{\hat{e}}} = \sum_{\hat{e} \in \hat{E}(e)} \frac{d_e}{m_e} \ln \frac{d_e/m_e}{d^0_e/m_e} = d_e \ln \frac{d_e}{d^0_e}$$

holds. Thus, for any feasible solution $\boldsymbol{d}$, we can realize the same relative entropy value by $\hat{\boldsymbol{d}}$. Therefore, we can rewrite Eq. (5.15) as

$$\sum_{k=1}^{\mathrm{depth}(G)} \sum_{e \in E_k} d_e \ln \frac{d_e}{d^0_e} = \sum_{k=1}^{\mathrm{depth}(G)} \sum_{\hat{e} \in \bigcup_{e \in E} \hat{E}(e)} \hat{d}_{\hat{e}} \ln \frac{\hat{d}_{\hat{e}}}{\hat{d}^0_{\hat{e}}}. \tag{5.16}$$

By construction of $\hat{E}(e)$, $\hat{d}_{\hat{e}} \in [0, 1/\nu m]$. Therefore, we can bound Eq. (5.16) by

$$\sum_{k=1}^{\mathrm{depth}(G)} \sum_{e \in E_k} d_e \ln \frac{d_e}{d^0_e} \leq \sum_{k=1}^{\mathrm{depth}(G)} \ln \frac{m}{\nu m} = \mathrm{depth}(G) \frac{1}{\nu}.$$

□

Let $\delta_t = \min_{k \in [t]} P^k(\boldsymbol{d}^{k-1}) - P^{t-1}(\boldsymbol{d}^{t-1})$ be the optimality gap. The following lemma justificates the stopping criterion for Algorithm 7.

Lemma 5.3. *Let P^T_{LP} be an optimal solution of Eq. (5.8) over $J_T = \{j_1, j_2, \ldots, j_T\}$. If $\eta \geq \frac{2}{\epsilon} \mathrm{depth}(G) \ln \frac{1}{\nu}$, then $\delta^{T+1} \leq \epsilon/2$ implies $g - P^T_{\mathrm{LP}} \leq \epsilon$, where g is the guarantee of the base learner.*

Note that if the algorithm always finds a hypothesis that maximizes the right-hand-side of Eq. (5.14), then this lemma guarantees the ϵ-accuracy to the optimal solution of Eq. (5.8).

Proof. Let $P^T(\boldsymbol{d})$ be the objective function of Eq. (5.13) over $J_T = \{j_1, j_2, \ldots, j_T\}$ obtained by Algorithm 7 and let $\boldsymbol{d}^T$ be the optimal feasible solution of it. By the choice of η and Lemma 5.2, we get

$$\frac{1}{\eta}\sum_{e\in E}\left[d_e \ln\frac{d_e}{d_e^0} - d_e + d_e^0\right] \leq \frac{1}{\eta}\operatorname{depth}(G)\ln\frac{1}{\nu} \leq \frac{\epsilon}{2}.$$

Thus, $P^T(\boldsymbol{d}^T) \leq P_{\mathrm{LP}}^T + \epsilon/2$. On the other hand, by the assumption on the weak learner, for any feasible solution $\boldsymbol{d}^{t-1}$, we obtain a $j_t \in [n+1]$ such that

$$g \leq \operatorname{sgn}(j_t)\sum_{e\in E}\operatorname{sgn}(e)d_e^{t-1}.$$

Using the nonnegativity of the unnormalized relative entropy, we get

$$g \leq \min_{t\in[T+1]}\operatorname{sgn}(j_t)\sum_{e\in E}\operatorname{sgn}(e)d_e^{t-1} \leq \min_{t\in[T+1]} P^t(\boldsymbol{d}^{t-1}).$$

Subtracting $P^T(\boldsymbol{d}^T)$ from both sides, we get

$$g - P^T(\boldsymbol{d}^T) \leq \min_{t\in[T+1]} P^t(\boldsymbol{d}^{t-1}) - P^t(\boldsymbol{d}^t) = \delta^{T+1} \leq \frac{\epsilon}{2}.$$

Thus, $\delta_{T+1} \leq \epsilon/2$ implies $g - P_{LP}^T \leq \epsilon$. □

With Lemma 5.3, We can obtain a similar iteration bound like ERLPBoost. To see that, we need to derive the dual problem of Eq. (5.13). By standard calculation, you can verify that the dual problem becomes:

$$\begin{aligned}
&\max_{\boldsymbol{w},\boldsymbol{s},\boldsymbol{\beta}} \frac{1}{\eta}\sum_{e\in E} d_e^0\left(1 - e^{-\eta A_e^t(\boldsymbol{w},\boldsymbol{s},\boldsymbol{\beta})}\right) + s_{\mathrm{root}} - s_{\mathrm{leaf}} - \frac{1}{\nu m}\sum_{e\in E} m_e\beta_e \qquad (5.17)\\
&\text{sub. to.} \sum_{j\in J_t} w_j = 1, \boldsymbol{w} \geq \boldsymbol{0}, \boldsymbol{\beta} \geq \boldsymbol{0}\\
&\qquad A_e^t(\boldsymbol{w},\boldsymbol{s},\boldsymbol{\beta}) := \operatorname{sgn}(e)\sum_{j\in J_t}\operatorname{sgn}(j) I_{[j\in\Phi(e)]} w_j + s_{e.u} - s_{e.v} + \beta_e, \quad \forall e \in E
\end{aligned}$$

Let $P^t(\boldsymbol{d})$, $D^t(\boldsymbol{w},\boldsymbol{s},\boldsymbol{\beta})$ be the objectives of the optimization sub-problems Eqs. (5.13) and (5.17), respectively. Let $\boldsymbol{d}^t$ be the optimal solution of Eq. (5.13) at round t and similarly, let $(\boldsymbol{w}^t, \boldsymbol{s}^t, \boldsymbol{\beta}^t)$ be the one of Eq. (5.17). Then, by KKT conditions, the following hold.

$$\sum_{j\in J_t}\operatorname{sgn}(j) w_j^t\left[\sum_{e\in E: j\in\Phi(e)}\operatorname{sgn}(e)d_e\right] = \max_{j\in J_t}\operatorname{sgn}(j)\sum_{e\in E: j\in\Phi(e)}\operatorname{sgn}(e)d_e \qquad (5.18)$$

We will prove the following lemma, which corresponds to Lemma 2 in [99].

Lemma 5.4. *If $\eta \geq 1/3$, then*

$$P^t(\boldsymbol{d}^t) - P^{t-1}(\boldsymbol{d}^{t-1}) \geq \frac{1}{18\eta \operatorname{depth}(G)} \left[P^t(\boldsymbol{d}^{t-1}) - P^{t-1}(\boldsymbol{d}^{t-1})\right]^2,$$

where $\operatorname{depth}(G)$ *denotes the max depth of graph* G*.*

Proof. First of all, we examine the right hand side of the inequality. By definition, $P^t(\boldsymbol{d}^{t-1}) \geq P^{t-1}(\boldsymbol{d}^{t-1})$ and

$$\begin{aligned}
&P^t(\boldsymbol{d}^{t-1}) - P^{t-1}(\boldsymbol{d}^{t-1}) \\
&= \operatorname{sgn}(j_t) \sum_{e \in E: j_t \in \Phi(e)} \operatorname{sgn}(e) d_e^{t-1} - \max_{j \in J_{t-1}} \operatorname{sgn}(j) \sum_{e \in E: j \in \Phi(e)} \operatorname{sgn}(e) d_e^{t-1} \\
&= \operatorname{sgn}(j_t) \sum_{e \in E: j_t \in \Phi(e)} \operatorname{sgn}(e) d_e^{t-1} - \sum_{j \in J_{t-1}} \operatorname{sgn}(j) w_j^{t-1} \sum_{e \in E: j \in \Phi(e)} \operatorname{sgn}(e) d_e^{t-1} \\
&= \sum_{e \in E} \operatorname{sgn}(e) d_e^{t-1} \left[\operatorname{sgn}(j_t) I_{[j_t \in \Phi(e)]} - \sum_{j \in J_{t-1}} \operatorname{sgn}(j) I_{[j \in \Phi(e)]} w_j^{t-1} \right] \\
&=: \sum_{e \in E} d_e^{t-1} x_e^t,
\end{aligned}$$

where the second equality holds from Eq. (5.18).

Now, we will bound $P^t(\boldsymbol{d}^t) - P^{t-1}(\boldsymbol{d}^{t-1})$ from below. For $\alpha \in [0, 1]$, let

$$\boldsymbol{w}^t(\alpha) := (1 - \alpha) \begin{bmatrix} \boldsymbol{w}^{t-1} \\ 0 \end{bmatrix} + \alpha \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix}. \tag{5.19}$$

Since $(\boldsymbol{w}^t, \boldsymbol{s}^t, \boldsymbol{\beta}^t)$ is the optimal solution of Eq. (5.13) at round t,

$$D^t(\boldsymbol{w}^t, \boldsymbol{s}^t, \boldsymbol{\beta}^t) \geq D^t(\boldsymbol{w}^t(\alpha), \boldsymbol{s}^{t-1}, \boldsymbol{\beta}^{t-1})$$

holds. By strong duality,

$$\begin{aligned}
&P^t(\boldsymbol{d}^t) - P^{t-1}(\boldsymbol{d}^{t-1}) \\
=&D^t(\boldsymbol{w}^t, \boldsymbol{s}^t, \boldsymbol{\beta}^t) - D^{t-1}(\boldsymbol{w}^{t-1}, \boldsymbol{s}^{t-1}, \boldsymbol{\beta}^{t-1}) \\
\geq&D^t(\boldsymbol{w}^t(\alpha), \boldsymbol{s}^{t-1}, \boldsymbol{\beta}^{t-1}) - D^{t-1}(\boldsymbol{w}^{t-1}, \boldsymbol{s}^{t-1}, \boldsymbol{\beta}^{t-1}) \\
=&\frac{1}{\eta} \sum_{e \in E} d_e^0 \left(1 - e^{-\eta A_e^t(\boldsymbol{w}^t(\alpha), \boldsymbol{s}^{t-1}, \boldsymbol{\beta}^{t-1})}\right) - \frac{1}{\eta} \sum_{e \in E} d_e^0 \left(1 - e^{-\eta A_e^{t-1}(\boldsymbol{w}^{t-1}, \boldsymbol{s}^{t-1}, \boldsymbol{\beta}^{t-1})}\right) \\
=&-\frac{1}{\eta} \sum_{e \in E} d_e^0 \left(e^{-\eta A_e^t(\boldsymbol{w}^t(\alpha), \boldsymbol{s}^{t-1}, \boldsymbol{\beta}^{t-1})} - e^{-\eta A_e^{t-1}(\boldsymbol{w}^{t-1}, \boldsymbol{s}^{t-1}, \boldsymbol{\beta}^{t-1})}\right).
\end{aligned}$$

By KKT conditions, we can write $\boldsymbol{d}^{t-1}$ in terms of the dual variables $\boldsymbol{w}^{t-1}$, $\boldsymbol{s}^{t-1}$, and $\boldsymbol{\beta}^{t-1}$:

$$\begin{aligned} d_e^{t-1} &= d_e^0 \exp\left[-\eta\left(\mathrm{sgn}(e)\sum_{j\in J_{t-1}}\mathrm{sgn}(j)I_{[j\in\Phi(e)]}w_j^{t-1} + s_u^{t-1} - s_v^{t-1} + \beta_e^{t-1}\right)\right] \\ &= d_e^0 e^{-\eta A_e^{t-1}(\boldsymbol{w}^{t-1},\boldsymbol{s}^{t-1},\boldsymbol{\beta}^{t-1})} \end{aligned}$$

Therefore,

$$\begin{aligned} P^t(\boldsymbol{d}^t) - P^{t-1}(\boldsymbol{d}^{t-1}) \geq & -\frac{1}{\eta}\sum_{e\in E} d_e^0 e^{-\eta A_e^{t-1}(\boldsymbol{w}^{t-1},\boldsymbol{s}^{t-1},\boldsymbol{\beta}^{t-1})}\left(e^{-\eta\alpha x_e^t} - 1\right) \\ = & -\frac{1}{\eta}\sum_{e\in E} d_e^{t-1}\left(e^{-\eta\alpha x_e^t} - 1\right). \end{aligned}$$

Since $x_e^t \in [-2,+2]$, $\frac{3\pm x_e^t}{6} \in [0,1]$ and $\frac{3+x_e^t}{6} + \frac{3-x_e^t}{6} = 1$. Thus, using Jensen's inequality, we get

$$\begin{aligned} P^t(\boldsymbol{d}^t) - P^{t-1}(\boldsymbol{d}^{t-1}) &\geq -\frac{1}{\eta}\sum_{e\in E} d_e^{t-1}\left(\exp\left[\frac{3+x_e^t}{6}(-3\eta\alpha) + \frac{3-x_e^t}{6}(3\eta\alpha)\right] - 1\right) \\ &\geq -\frac{1}{\eta}\sum_{e\in E} d_e^{t-1}\left(\frac{3+x_e^t}{6}e^{-3\eta\alpha} + \frac{3-x_e^t}{6}e^{3\eta\alpha} - 1\right) =: R(\alpha). \end{aligned}$$

The above inequality holds for all $\alpha \in [0,1]$. Here, $R(\alpha)$ is a concave function w.r.t. α so that we can choose the optimal $\alpha \in [0,1]$. By standard calculation, we get that the optimal $\alpha \in \mathbb{R}$ is

$$\alpha = \frac{1}{6\eta}\ln\frac{\sum_{e\in E} d_e^{t-1}(3+x_e^t)}{\sum_{e\in E} d_e^{t-1}(3-x_e^t)} \tag{5.20}$$

Since $x_e^t \leq 2$ for all $e \in E$, $\alpha \leq \frac{1}{6\eta}\ln\frac{\ln}{5} < \frac{1}{3\eta}$. Thus, $\alpha \leq 1$ holds. On the other hand, $\sum_{e\in E} d_e^{t-1}x_e^t \geq 0$ so that $\alpha \geq \frac{1}{6\eta}\ln 1 = 0$. Therefore, we can use Eq. (5.20) to lower-bound $P^t(\boldsymbol{d}^t) - P^{t-1}(\boldsymbol{d}^{t-1})$.

$$\begin{aligned} & P^t(\boldsymbol{d}^t) - P^{t-1}(\boldsymbol{d}^{t-1}) \\ \geq & -\frac{1}{\eta}\left[\frac{1}{3}\sqrt{\left(\sum_{e\in E} d_e^{t-1}(3+x_e^t)\right)\left(\sum_{e\in E} d_e^{t-1}(3-x_e^t)\right)} - \sum_{e\in E} d_e^{t-1}\right] \\ = & -\frac{1}{\eta}\left[\frac{1}{3}\sqrt{9\left(\sum_{e\in E} d_e^{t-1}\right)^2 - \left(\sum_{e\in E} d_e^{t-1}x_e^t\right)^2} - \sum_{e\in E} d_e^{t-1}\right] \end{aligned}$$

By using the inequality

$$\forall a, b \geq 0, 3a \geq b \implies \frac{1}{3}\sqrt{9a^2 - b^2} - a \leq -\frac{b^2}{18a},$$

we get

$$\begin{aligned} P^t(\boldsymbol{d}^t) - P^{t-1}(\boldsymbol{d}^{t-1}) \geq & \frac{1}{18\eta} \frac{\left(\sum_{e \in E} d_e^{t-1} x_e^t\right)^2}{\sum_{e \in E} d_e^{t-1}} \\ \geq & \frac{1}{18\eta} \frac{\left(\sum_{e \in E} d_e^{t-1} x_e^t\right)^2}{\operatorname{depth}(G)} \\ = & \frac{1}{18\eta \operatorname{depth}(G)} \left[P^t(\boldsymbol{d}^{t-1}) - P^{t-1}(\boldsymbol{d}^{t-1})\right]^2, \end{aligned}$$

which is the inequality we desire. □

We introduce the following lemma to prove the iteration bound of the compressed ERLPBoost.

Lemma 5.5 ([1])**.** *Let $(\delta^t)_{t \in \mathbb{N}} \subset \mathbb{R}_{\geq 0}$ be a sequence such that*

$$\exists c > 0, \quad \forall t \geq 1, \quad \delta^t - \delta^{t+1} \geq \frac{(\delta^t)^2}{c}.$$

Then, the following inequality holds for all $t \geq 1$.

$$\delta^t \leq \frac{c}{t - 1 + \frac{c}{\delta^1}}$$

Now we are ready to prove Theorem 5.2.

Proof of Theorem 5.2. By definition of η, $\eta \geq 1/3$ holds. Thus, by Lemma 5.4, we get

$$\begin{aligned} P^t(\boldsymbol{d}^t) - P^{t-1}(\boldsymbol{d}^{t-1}) \geq & \frac{1}{18\eta \operatorname{depth}(G)} \left[P^t(\boldsymbol{d}^{t-1}) - P^{t-1}(\boldsymbol{d}^{t-1})\right]^2 \\ \geq & \frac{1}{18\eta \operatorname{depth}(G)} \left[\min_{1 \leq q \leq t} P^q(\boldsymbol{d}^{q-1}) - P^{t-1}(\boldsymbol{d}^{t-1})\right]^2 \\ = & \frac{(\delta^t)^2}{18\eta \operatorname{depth}(G)}. \end{aligned}$$

On the other hand,

$$\begin{aligned} & P^t(\boldsymbol{d}^t) - P^{t-1}(\boldsymbol{d}^{t-1}) \\ = & \left(\min_{1 \leq q \leq t} P^q(\boldsymbol{d}^{q-1}) - P^{t-1}(\boldsymbol{d}^{t-1})\right) - \left(\min_{1 \leq q \leq t} P^q(\boldsymbol{d}^{q-1}) - P^t(\boldsymbol{d}^t)\right) \\ \leq & \left(\min_{1 \leq q \leq t-1} P^q(\boldsymbol{d}^{q-1}) - P^{t-1}(\boldsymbol{d}^{t-1})\right) - \left(\min_{1 \leq q \leq t} P^q(\boldsymbol{d}^{q-1}) - P^t(\boldsymbol{d}^t)\right) \\ = & \delta^{t-1} - \delta^t. \end{aligned}$$

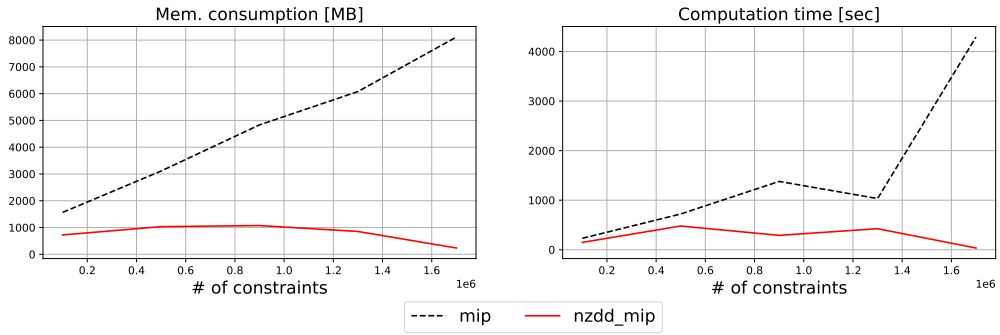

Figure 5.5: The comparison for synthetic datasets of a MIP problem. The horizontal axis represents the number of constraints of the original problem.

Thus, we get the following relation:

$$\delta^{t-1} - \delta^t \geq \frac{(\delta^t)^2}{c\eta}, \quad \text{where} \quad c = 18\,\mathrm{depth}(G).$$

Lemma 5.5 gives us $\delta^t \leq \frac{c\eta}{t-1+\frac{c\eta}{\delta^1}}$. Rearranging this inequality, we get $T \leq \frac{c\eta}{\delta^T} - \frac{c\eta}{\delta^1} + 1$. Since $\delta^1 = \sum_{e\in E: j_1 \in \Phi(e)} \mathrm{sgn}(e) d_e^0 \leq \mathrm{depth}(G)$, we have $\frac{c\eta}{\delta^1} \geq \frac{72}{\epsilon}\,\mathrm{depth}(G) > 1$. Therefore, the above inequality implies that $T \leq \frac{c\eta}{\delta^T}$. As long as the stopping criterion does not satisfy, $\delta^t > \epsilon/2$ hold. Thus,

$$\begin{aligned} T \leq & \frac{1}{\delta^T} \cdot 36\,\mathrm{depth}(G) \frac{2}{\epsilon}\,\mathrm{depth}(G) \cdot \max\left(1, \ln\frac{1}{\nu}\right) \\ \leq & \frac{144}{\epsilon^2}\,\mathrm{depth}(G)^2 \max\left(1, \ln\frac{1}{\nu}\right). \end{aligned}$$

□

5.8 Experiment

We show preliminary experimental results on synthetic and real large data sets[2]. The tasks are, the mixed integer programming, and the ℓ_1-norm regularized soft margin optimization. Our experiments are conducted on a server with 2.60 GHz Intel Xeon Gold 6124 CPUs and 314GB memory. We use Gurobi optimizer 9.01, a state-of-the-art commercial LP solver. To obtain NZDD representations of data sets, we apply the procedure described in the previous section.

[2] `https://bitbucket.org/kohei_hatano/codes_extended_formulation_nzdd/`.

Table 5.2: Computation time (seconds) for constructing NZDDs for synthetic MIP datasets, described in Table 5.3. All datasets have $n = 25$ features and each instance have $k = 10$ nonzero components.

m	ZCOMP	Reducing proc.	Total
4×10^5	0.39	1.02	1.41
8×10^5	0.76	1.38	2.14
12×10^5	1.08	1.41	2.49
16×10^5	1.36	1.10	2.46
20×10^5	1.60	0.33	1.93

Preprocessing of datasets The datasets for the ℓ_1-norm regularized soft margin optimization are obtained from the libsvm datasets. Some of them contain real-valued features. We convert them to binary ones by rounding them using 0.5 as a threshold.

NZDD construction time and summary of datasets Computation times for constructing NZDDs for the MIP synthetic datasets and synthetic and real datasets of the ℓ_1-norm regularized soft margin optimization are summarized in Tables 5.2, 5.4 and 5.6, respectively. Note that the NZDD construction time is not costly and negligible in general because once we construct NZDDs, we can re-use those NZDDs for solving optimization problems with different objective functions or hyperparameters such as ν in the soft margin optimization.

Mixed Integer programming on synthetic datasets First, we apply our extended formulation Eq. (5.1) to mixed integer programming tasks over synthetic data sets. The problems are defined as the linear optimization with n variables and m linear constraints of the form $A\boldsymbol{x} \geq \boldsymbol{b}$, where (i) each row of A has k entries of 1 and others are 0s and nonzero entries are chosen randomly without repetition (ii) coefficients a_i of linear objective $\sum_{i=1}^{n} a_i x_i$ is chosen from $\{1, ..., 100\}$ randomly, and (iii) first l variables take binary values in $\{0, 1\}$ and others take real values in $[0, 1]$. In our experiments, we fix $n = 25, k = 10$, $l = 12$ and $m \in \{4 \times 10^5, 8 \times 10^5 ..., 20 \times 10^5\}$. We apply the Gurobi optimizer directly to the problem denoted as `mip` and the solver with pre-processing the problem by our extended formulation (denoted as `nzdd_mip`, respectively. The results are

Table 5.3: Summary of synthetic datasets for MIP. The term "original" and "extended" mean the original and the extended formulations, respectively.

Data size		NZDD size		Variables		Constraints	
n	m	$\|V\|$	$\|E\|$	Original	Extended	Original	Extended
25	4×10^5	13,321	177,356	25	13,344	400,000	177,356
25	8×10^5	17,771	241,488	25	17,794	800,000	241,488
25	12×10^5	19,348	249,749	25	19,371	1,200,000	249,749
25	16×10^5	17,819	204,723	25	17,842	1,600,000	204,723
25	20×10^5	6,161	55,555	25	6,184	2,000,000	55,555

summarized in Fig. 5.5. Our method consistently improves computation time for these datasets. This makes sense since it can be shown that when $m = O(n^k)$ there exists an NZDD of size $O(nk)$ representing the constraint matrix. In addition, the pre-processing time is within 2 seconds in all cases.

ℓ_1-norm soft margin optimization on real data sets Next, we apply our methods on the task of the ℓ_1-norm soft margin optimization. This problem is a standard optimization problem in the machine learning literature, categorized as LP, for finding sparse linear classifiers given labeled instances. We compare the following methods using a naive LP solver, (i) previous formulation (denoted as `naive`), (ii) our formulation over NZDD (denoted as `nzdd_naive`). Formulations of both (i) and (ii) is placed in previous sections. We measure its computation time (CPU time) and maximum memory consumption, respectively, and compare their averages over parameters. Further, we perform 5-fold cross validation to check the test error rates of our methods on real data sets. In fact, the test error rates are similar between methods for (i) and (ii). This means our extended formulation is comparable to the standard one in terms of generalization performance.

Test error rates Table 5.7 summarizes the test error rates of soft margin optimization algorithms using cross validation. These results imply the extended formulation Eq. (5.6) is comparable to the original problem Eq. (5.4) to obtain generalization ability.

We compare methods on some real data sets in the libsvm datasets [16] to see the effectiveness of our approach in practice. Generally, the datasets contain huge samples

Table 5.4: Computation times (seconds) for NZDD construction for the synthetic datasets, described in Table 5.5. all data have $n = 20$ features.

m	ZCOMP	Reducing proc.	Total
1×10^5	0.08	0.14	0.22
2×10^5	0.15	0.14	0.29
3×10^5	0.25	0.12	0.37
4×10^5	0.31	0.08	0.39
5×10^5	0.39	0.05	0.44
6×10^5	0.46	0.05	0.51
7×10^5	0.53	0.03	0.56
8×10^5	0.60	0.02	0.62
9×10^5	0.68	0.01	0.69
10×10^5	0.72	0.00	0.72

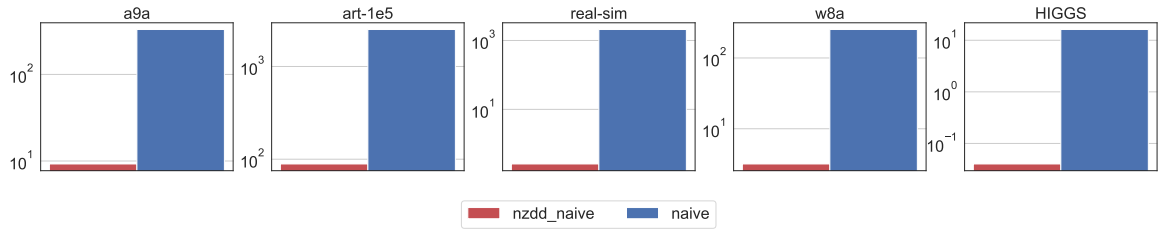

Figure 5.6: Comparison of computation times (sec.) for real data sets of the soft margin optimization problem. The y-axis is plotted in the logarithmic scale.

(m varies from 3×10^4 to 10^7) with a relatively small size of features (n varies from 20 to 10^5). The features of instances of each dataset is transformed into binary values. Results are summarized in Fig. 5.6. Note that these results exclude NZDD construction times since the compression takes around 1 second, except for the HIGGS dataset (around 13 seconds). Furthermore, the construction time of NZDDs can be neglected in the following reason: We often need to try multiple choices of the hyperparameters (ν in our case) and solve the optimization problem for each set of choices. But once we construct an NZDD, we can be re-use it for different values of hyperparameters without reconstructing NZDDs.

Table 5.5: Summary of synthetic datasets for the soft margin optimization. The term "original" and "extended" mean the original and the extended formulations, respectively.

Data size		NZDD size		Variables		Constraints	
n	m	$\|V\|$	$\|E\|$	Original	Extended	Original	Extended
20	1×10^5	4,202	55,147	100,021	59,370	200,001	110,297
20	2×10^5	6,663	82,810	200,021	89,494	400,001	165,623
20	3×10^5	11,022	103,323	300,021	114,366	600,001	206,649
20	4×10^5	13,596	120,994	400,021	134,611	800,001	241,991
20	5×10^5	12,565	128,670	500,021	141,256	1,000,001	257,343
20	6×10^5	13,796	127,139	600,021	140,956	1,200,001	254,281
20	7×10^5	13,513	114,471	700,021	128,005	1,400,001	228,945
20	8×10^5	9,569	95,454	800,021	105,044	1,600,001	190,911
20	9×10^5	6,725	75,318	900,021	82,064	1,800,001	150,639
20	10×10^5	3,914	40,226	1,000,021	44,161	2,000,001	80,455

Table 5.6: Computation times (seconds) for constructing NZDDs for real datasets of the soft margin optimization.

dataset	ZCOMP	Reducing proc.	Total
a9a	0.04	0.20	0.24
art-100000	0.10	0.43	0.53
real-sim	0.24	0.99	1.23
w8a	0.27	4.20	4.47
HIGGS	13.13	0.00	13.13

Table 5.7: Test error rates for real datasets

Data sets	`lpb`	`nzdd_naive`	`nzdd_erlp`
a9a	0.174	0.159	0.157
art-100000	0.000	0.0004	0.004
real-sim	0.179	0.169	0.532
w8a	0.030	0.030	0.029

Chapter 6

Online Combinatorial Linear Optimization via a Frank-Wolfe-based Metarounding Algorithm

Metarounding is an approach to convert an approximation algorithm for linear optimization over some combinatorial classes to an online linear optimization algorithm for the same class. We propose a new metarounding algorithm under a natural assumption that a relax-based approximation algorithm exists for the combinatorial class. Our algorithm is much more efficient in both theoretical and practical aspects.

The results in this chapter appear in [65].

6.1 Problem setting

Online decision-making problems using combinatorial objects arise in many applications, e.g., routing, advertisements, and resource allocation tasks. Examples of combinatorial objects include permutations [41], k-sets [100], paths [37], matching [15, 41], spanning trees [73] and so on.

Let $\mathcal{C} \subset \mathbb{N}^n$ be a combinatorial class, i.e., a set of combinatorial objects, where each combinatorial object is represented as a positive vector. More formally, online combinatorial linear optimization over a combinatorial class $\mathcal{C}$ is defined as a repeated game between a player and an environment. The protocol is the following: For each round $t = 1, 2, \ldots,$

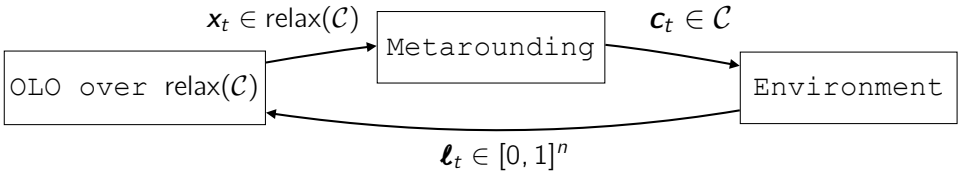

Figure 6.1: A conceptual diagram for online combinatorial linear optimization via metarounding. In each round t, metarounding converts a vector $\boldsymbol{x}_t$ into a combinatorial vector $\boldsymbol{c}_t \in \mathcal{C}$.

(i) the player chooses $\boldsymbol{c}_t \in \mathcal{C}$. Then, (ii) the environment reveals a loss vector $\boldsymbol{\ell}_t \in [0,1]^n$ and (iii) the player incurs the loss $\boldsymbol{c}_t \cdot \boldsymbol{\ell}_t$.

The player's goal is to minimize the α-regret defined by

$$R_T(\alpha) = \sum_{t=1}^{T} \boldsymbol{c}_t \cdot \boldsymbol{\ell}_t - \alpha \min_{\boldsymbol{c} \in \mathcal{C}} \sum_{t=1}^{T} \boldsymbol{c} \cdot \boldsymbol{\ell}_t,$$

for as small $\alpha \geq 1$ as possible.

There are two approaches for online combinatorial linear optimization over $\mathcal{C}$. The first approach is to design algorithms for individual classes [15, 41, 100, 104, 105]. The second approach is so called offline-to-online conversion, which uses an offline approximation algorithm over $\mathcal{C}$ to construct an online algorithm over $\mathcal{C}$ [30, 33, 48]. An offline approximation algorithm takes $\boldsymbol{\ell}$ as input and then outputs a vector $\boldsymbol{c} \in \mathcal{C}$ satisfying $\boldsymbol{c} \cdot \boldsymbol{\ell} \leq \alpha \min_{\boldsymbol{c}' \in \mathcal{C}} \boldsymbol{c}' \cdot \boldsymbol{\ell}$. The offline-to-online conversion approach is generic in that once we design an efficient conversion algorithm, we can just use known approximation algorithms for different combinatorial classes. For example, Garber proposed an algorithm that calls the sub-routine $O(\ln T)$ times for each round t and achieves $R_T(\alpha) = O(T^{2/3})$ [33]. The paper also proposed an algorithm that calls the sub-routine $O(\sqrt{T} \ln T)$ per round and achieves $R_T(\alpha) = O(\sqrt{T})$. The algorithm needs to know the number of total rounds T and the approximation ratio α a priori. In addition, the number of sub-routine calls depends on T, so the conversion via the sub-routine may slow for a huge T.

Another offline-to-online conversion approach is to use metarounding [12], which uses a relaxation-based approximation algorithm. In fact, for many combinatorial classes $\mathcal{C}$, there are relaxation-based approximation algorithms. Thus, the requirement is natural. Metarounding converts a vector $\boldsymbol{x} \in \mathrm{relax}(\mathcal{C})$ into a combinatorial vector $\boldsymbol{c} \in \mathcal{C}$, where $\mathrm{relax}(\mathcal{C})$ is a convex set satisfying $\mathcal{C} \subset \mathrm{relax}(\mathcal{C})$. The relax-based approximation algorithms takes $\boldsymbol{\ell}$ as input and then outputs a vector $\boldsymbol{c} \in \mathcal{C}$ satisfying $\boldsymbol{c} \cdot \boldsymbol{\ell} \leq \alpha \min_{\boldsymbol{p} \in \mathrm{relax}(\mathcal{C})} \boldsymbol{p} \cdot \boldsymbol{\ell}$.

The only difference between relaxation-based approximation and the approximation algorithm is the right-hand-side of the condition. Using the relaxation-based approximation algorithm as sub-routine, metarounding algorithms aim to output a vector $\boldsymbol{c} \in \mathcal{C}$ randomly to satisfy $\mathbb{E}_{\boldsymbol{c}}[\boldsymbol{c}] \cdot \boldsymbol{\ell} \leq \alpha \boldsymbol{x} \cdot \boldsymbol{\ell}$ for all $\boldsymbol{\ell}$. Thus, combining the metarounding algorithm with an arbitrary online linear optimization algorithm over relax($\mathcal{C}$), one can achieve $R_T(\alpha) = \alpha$(1-regret of the OLO algorithm) [30]. Thus, once an efficient metarounding algorithm is obtained, we can fuse it to an OLO algorithm to obtain a better regret bound. Fig. 6.1 shows the concept of metarounding for online combinatorial linear optimization over $\mathcal{C}$. Assuming the existence of such relax-based approximation algorithm, Fujita et al. [30] proposes a metarounding algorithm that finds a distribution $\boldsymbol{\lambda} \in \Delta_{\mathcal{C}}$ satisfying $\mathbb{E}_{\boldsymbol{c}}[\boldsymbol{c}] \cdot \boldsymbol{\ell} \leq (\alpha + \epsilon)\boldsymbol{x} \cdot \boldsymbol{\ell}$ for all $\boldsymbol{\ell}$ in $O(M^2 \mathrm{diam}_\infty(\mathcal{C})^2 n^2 \ln(n)/\epsilon^2)$ iterations, where $\mathrm{diam}_\infty(\mathcal{C}) = \max\{\|\boldsymbol{c}\|_\infty \mid \boldsymbol{c} \in \mathcal{C}\}$ is the maximal value of a combinatorial vector $\boldsymbol{c} \in \mathcal{C}$ and $M = \max\{1/x_i \mid x_i \neq 0, i \in [n]\}$ is the input-dependent constant.

This paper proposes a new metarounding algorithm that finds a distribution $\boldsymbol{\lambda} \in \Delta_{\mathcal{C}}$ over $\mathcal{C}$ that satisfies $\mathbb{E}_{\boldsymbol{c} \sim \boldsymbol{\lambda}}[\boldsymbol{c}] \cdot \boldsymbol{\ell} \leq (\alpha + \epsilon)\boldsymbol{x} \cdot \boldsymbol{\ell}$ for all $\boldsymbol{\ell} \in [0, 1]^n$ in $O(\mathrm{diam}_\infty(\mathcal{C})^2 \ln(n)/\epsilon^2)$ rounds. This guarantee is significantly better compared to the one by Fujita et al. [30]. Our algorithm is designed based on Frank-Wolfe algorithms [26, 46]. Table 6.1 summarizes the previous works and our contribution.

Our technical contribution is two-fold. The first contribution is a re-formulation of the optimization problem for metarounding, based on a new observation (Proposition 1). This formulation enables us to derive algorithms with better iteration bounds. The second contribution is a new boosting-based metarounding algorithm. Our algorithm is similar to ERLPBoost[99], but ours employs a modified regularizer which is better suited for metarounding. In fact, it can be viewed that our algorithm is a generalization of ERLPBoost for the approximation factor $\alpha > 1$. Furthermore, our analyses are based on those for Frank-Wolfe, which are completely different from those for ERLPBoost. As a result, our result significantly improves Fujita et al's on the number of approximation oracle calls per trial from $O(\mathrm{diam}_\infty(\mathcal{C})^2 n^2 \ln(n)/\epsilon^2)$ to $O(\mathrm{diam}_\infty(\mathcal{C})^2 \ln(n)/\epsilon^2)$. We also observe its significant improvements in the preliminary experiments.

6.2 Related Work

Fujita et al. [30] try to solve the problem with a boosting approach based on SoftBoost [97]. The resulting algorithm named `MbB` (the shorthand for "Metarounding-by-Boosting") is

Table 6.1: Comparison of the previous works. Our assumption is the same as the one in Fujita et al. [30]. Note that the work by Fujita et al. and ours aim to optimize $(\alpha+\epsilon)$-regret $R_T(\alpha+\epsilon)$ for arbitrarily small $\epsilon > 0$.

Algorithm	Approx. alg. oracle	# of oracle calls per trial	Regret
Kalai et al. [48]	Any	$O(T)$	$O(T^{1/2})$
Garber [33]	Any	$O(n^2 \ln T)$	$O(T^{2/3})$
Garber [33]	Any	$O(n^2\sqrt{T} \ln T)$	$O(T^{1/2})$
Fujita et al. [30]	Relax-based	$O(\mathrm{diam}_\infty(\mathcal{C})^2 n^2 \ln(n)/\epsilon^2)$	$O(T^{1/2})$
This work	Relax-based	$O(\mathrm{diam}_\infty(\mathcal{C})^2 \ln(n)/\epsilon^2)$	$O(T^{1/2})$

guaranteed to terminate in $O\left(M^2 n^2 \ln(n)/\epsilon^2\right)$ iterations, where M is the input-dependent constant. They showed the efficiency of `MbB` by numerical experiments against the ellipsoid method [13]. There are two main advantages for `MbB`. The first is that `MbB` does not need to know the approximation ratio α a priori. The other is that the iteration bound for `MbB` does not depend on the iteration T of online algorithms. Previous works take $O(\ln T)$ [33] to $O(T)$ [13] iterations for every round of online combinatorial optimization, where T is the number of total rounds for the online algorithm.

6.2.1 Margin optimization boosting algorithms

Boosting is a common machine learning technique that combines multiple low-accuracy predictors to yield a highly accurate one. Since the predictors to be combined are chosen from a set of exponentially many predictors, one cannot compute the combined predictor directory. The main idea of boosting is to collect the most promising predictors one by one until some condition is satisfied.

Gradient Boosting Machines [29], such as XGBoost [17] and LightGBM [49], are popular algorithms that minimize some loss functions. For binary classification, it is well known that a predictor that maximizes the quantity so-called "margin" instead of minimizing loss functions is known to have smaller losses for unseen data [81]. Thus, some algorithms, such as LPBoost [21], SoftBoost [97], and ERLPBoost [99], are proposed to optimize the margin.

Algorithm 8 Metarounding by ERLPBoost

Input: $\boldsymbol{x} \in \mathrm{relax}(\mathcal{C})$, accuracy $\epsilon > 0$, and a relax-based α-approx. algorithm $\mathcal{B}$.

1: Initialize $\bar{\boldsymbol{\ell}}_0 = \frac{1}{n}\boldsymbol{x}, \boldsymbol{\ell}_0 = \frac{1}{n}\mathbf{1}$ and set $H^\star(\bar{C}^{(0)}\boldsymbol{\lambda}_0) = \infty$, $\eta = 2\ln(n)/\epsilon$.
2: **for** $k = 0, 1, 2, \ldots, K$ **do**
3: Call the approximation algorithm to obtain $\boldsymbol{c}_{k+1} \in \mathcal{B}(\boldsymbol{\ell}_k)$.
Let $\bar{\boldsymbol{c}}_{k+1} = (c_1/M, c_2/M, \ldots, c_n/M)$.
4: **if** $\epsilon_k := H^\star(\bar{C}^{(k)}\boldsymbol{\lambda}_k) - \max_{j \le k} \bar{\boldsymbol{c}}_{j+1} \cdot \bar{\boldsymbol{\ell}}_j \le \epsilon/2$ **then**
5: Set $K = k$. **break.**
6: **end if**
7: Set $\boldsymbol{\lambda}_{k+1} \leftarrow \arg\min_{\boldsymbol{\lambda} \in \Delta_{k+1}} H^\star(\bar{C}^{(k+1)}\boldsymbol{\lambda})$, where $\bar{C}^{(k+1)} = \begin{bmatrix} \bar{\boldsymbol{c}}_1 & \bar{\boldsymbol{c}}_2 & \ldots & \bar{\boldsymbol{c}}_{k+1} \end{bmatrix}$.
8: Set $\bar{\boldsymbol{\ell}}_{k+1} = \nabla H^\star(\bar{C}^{(k+1)}\boldsymbol{\lambda}_{k+1}) = \arg\max_{\bar{\boldsymbol{\ell}} \in \Delta_n} \bar{\boldsymbol{\ell}}^\top \bar{C}^{(k+1)}\boldsymbol{\lambda}_{k+1} - \frac{1}{\eta}\Delta(\bar{\boldsymbol{\ell}})$ and $\boldsymbol{\ell}_{k+1} = (\bar{\ell}_{k+1,1}/x_1, \bar{\ell}_{k+1,2}/x_2, \ldots, \bar{\ell}_{k+1,n}/x_n)$
9: **end for**
10: Solve Eq. (6.1) over $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_{K+1}$ and let the optimal solution be $\boldsymbol{\lambda}^\star$.

Output: $\boldsymbol{\lambda}^\star \in \Delta_{\mathcal{C}}$

6.3 The Frank-Wolfe-based algorithm

First, we revisit the approach by Fujita et al. [30]. In `MbB`, they formulate an α-metarounding as an algorithm that solves the following optimization problem.

$$\min_{\beta, \boldsymbol{\lambda}} \ \beta \qquad \text{s.t.} \quad \sum_{\boldsymbol{c} \in \mathcal{C}} \lambda_{\boldsymbol{c}} c_i \le \beta x_i, \quad \forall i \in [n], \qquad \boldsymbol{\lambda} \in \Delta_{\mathcal{C}}. \tag{6.1}$$

The Lagrange dual problem for Eq. (6.1) is

$$\max_{\gamma, \boldsymbol{\ell}} \ \gamma \qquad \text{s.t.} \quad \boldsymbol{\ell} \cdot \boldsymbol{c} \ge \gamma, \quad \forall \boldsymbol{c} \in \mathcal{C}, \qquad \boldsymbol{\ell} \cdot \boldsymbol{x} = 1, \quad \boldsymbol{\ell} \ge \mathbf{0}. \tag{6.2}$$

Eq. (6.2) has $O(|\mathcal{C}|)$ constraints, so `MbB` uses the column-generation approach like soft margin boosting algorithms; That is, starting from $\tilde{\mathcal{C}} = \emptyset$, they solve a problem under the constraints in Eq. (6.2) over $\tilde{\mathcal{C}}$ and then append a new combinatorial vector $\boldsymbol{c} \in \mathcal{C}$ to $\tilde{\mathcal{C}}$ one by one until some conditions are met. One can see that Eq. (6.2) can be seen as so-called "Edge minimization" in boosting (See, e.g., [21, 97]). In fact, setting $\boldsymbol{x} = \mathbf{1}$ in Problem Eq. (6.2) corresponds to margin optimization. With this insight, `MbB` is proposed

based on a boosting algorithm named SoftBoost [97], but the resulting guarantee is not as good as the one in SoftBoost.

We want to use the state-of-the-art soft margin optimization named ERLPBoost [99] to solve Eq. (6.2), which uses a normalized relative entropy regularizer, but there are some issues. First, constraint $\boldsymbol{\ell} \cdot \boldsymbol{x} = 1$ makes the direct application hard. ERLPBoost succeeded because its feasible region is probability simplex, while Eq. (6.1) is not. One naive idea is to use unnormalized relative entropy instead of the normalized one. This approach results in a similar result to Fujita et al. since the variable $\boldsymbol{\ell}$ is unbounded. With these issues, we use a different approach, a Frank-Wolfe-based approach similar to the one in Mitsuboshi et al. [64]. First, we convert Eq. (6.2) to an equivalent bounded problem and then use an ERLPBoost-like problem. The resulting algorithm is a generalization of the soft margin boosting in the following sense: Our algorithm becomes ERLPBoost if we set $\boldsymbol{x} = \mathbf{1}$.

6.3.1 Bounding the dual variable

Before getting into our main result, we prove the following proposition.

Proposition 6.1. *Let* $M := \max\{\frac{1}{x_i} \mid x_i \neq 0, i \in [n]\}$. *There is an optimal solution* $(\gamma^\star, \boldsymbol{\ell}^\star)$ *to Eq.* (6.2) *satisfying* $\ell_i^\star \leq M$ *for all* $i \in [n]$.

Proof. Let $(\beta^\star, \boldsymbol{\lambda}^\star)$, $(\gamma^\star, \boldsymbol{\ell}^\star)$ be optimal solutions to Eqs. (6.1) and (6.2), respectively. Without loss of generality, we can restrict the combinatorial set to $\mathcal{C}^\star := \{\boldsymbol{c} \in \mathcal{C} \mid \lambda_{\boldsymbol{c}} > 0\}$. We consider the following two cases.

1. $\forall i \in [n], x_i \neq 0$. This case,

$$\boldsymbol{\ell}^\star \cdot \boldsymbol{x} > \sum_{j \neq i} \ell_j^\star x_j + M x_i > \sum_{j \neq i} \ell_j^\star x_j + M \min_{k \in [n]} x_k \geq 1,$$

   which contradicts to the constraint $\boldsymbol{\ell}^\star \cdot \boldsymbol{x} = 1$, so that $\ell_i^\star \leq M$ for all $i \in [n]$.

2. $\exists i \in [n], x_i = 0$. Since $(\beta^\star, \boldsymbol{\lambda}^\star)$ is a feasible solution, $\sum_{\boldsymbol{c} \in \mathcal{C}^\star} \lambda_{\boldsymbol{c}}^\star c_i \leq \beta^\star x_i = 0$, which implies $\lambda_{\boldsymbol{c}}^\star = 0$ or $\boldsymbol{c}_i = 0$ for all $\boldsymbol{c} \in \mathcal{C}^\star$. By definition of $\mathcal{C}^\star$, $\lambda_{\boldsymbol{c}}^\star > 0$ for all $\boldsymbol{c} \in \mathcal{C}^\star$, so we have $c_i = 0$ for all $\boldsymbol{c} \in \mathcal{C}^\star$. Thus, $\ell_i^\star c_i = 0$ does not affect to the objective value $\gamma^\star$. So we can set $\ell_i^\star = 0$ without losing the optimality.

□

As discussed in the above proof, the variables $\{\ell_i \mid x_i = 0, i \in [n]\}$ do not contribute to the optimal value, so we assume that $x_i > 0$ for all $i \in [n]$ without loss of generality. Furthermore, restricting $\boldsymbol{\ell} \in [0, M]^n$ does not affect the optimal solution to Eq. (6.2), so we consider the following problem

$$\max_{\gamma, \boldsymbol{\ell}} \ \gamma \qquad \text{s.t.} \quad \boldsymbol{\ell} \cdot \boldsymbol{x} = 1, \quad \boldsymbol{\ell} \cdot \boldsymbol{c} \geq \gamma, \quad \forall \boldsymbol{c} \in \mathcal{C}, \quad \mathbf{0} \leq \boldsymbol{\ell} \leq \boldsymbol{M}, \tag{6.3}$$

where M is the value defined in Proposition Proposition 6.1. To clarify the analysis, we rewrite the above problem by introducing $\bar{\boldsymbol{\ell}} = (\ell_1 x_1, \ell_2 x_2, \ldots, \ell_n x_n)$ and $\bar{\mathcal{C}} = \{\bar{\boldsymbol{c}} \mid \bar{\boldsymbol{c}} = (c_1/x_1, c_2/x_2, \ldots, c_n/x_n), \boldsymbol{c} = (c_1, c_2, \ldots, c_n) \in \mathcal{C}\}$.

$$\max_{\gamma, \bar{\boldsymbol{\ell}}} \ \gamma \qquad \text{s.t.} \quad \bar{\boldsymbol{\ell}} \cdot \bar{\boldsymbol{c}} \geq \gamma, \quad \forall \bar{\boldsymbol{c}} \in \bar{\mathcal{C}}, \quad \bar{\boldsymbol{\ell}} \in \Delta_n. \tag{6.4}$$

Obviously, Eq. (6.4) is equivalent to Eq. (6.3). Thus, the following sections focus on solving Eq. (6.4).

6.3.2 The regularization

Regularizing a smooth function to a convex objective function does not make the objective smooth while regularizing a strongly convex function makes it strongly convex. So, we use a strongly convex regularizer $R(\bar{\boldsymbol{\ell}}) = \sum_{i=1}^n \bar{\ell}_i \ln \frac{\bar{\ell}_i}{1/n}$ to the objective function of (6.4) then consider the primal problem. We solve Eq. (6.4) by ERLPBoost-based algorithm, shown in 8. Our algorithm aims to find an $\epsilon/2$-approximate solution to the following problem.

$$\max_{\gamma, \boldsymbol{\ell}} \ \gamma - \frac{1}{\eta} R(\bar{\boldsymbol{\ell}}) \qquad \text{s.t.} \quad \bar{\boldsymbol{\ell}} \cdot \bar{\boldsymbol{c}} \geq \gamma, \quad \forall \bar{\boldsymbol{c}} \in \bar{\mathcal{C}}, \quad \bar{\boldsymbol{\ell}} \in \Delta_n. \tag{6.5}$$

Note that, by definition, $R(\bar{\boldsymbol{\ell}}) \leq \ln n$ holds for all $\bar{\boldsymbol{\ell}} \in \Delta_n$.

Let $\mathbb{I}_{\Delta_n}$ be the indicator function for Δ_n, $J(\boldsymbol{\theta}) = \max_{\boldsymbol{c} \in \bar{\mathcal{C}}} \theta_{\boldsymbol{c}}$ be the max function over $\mathbb{R}^{\bar{\mathcal{C}}}$, and let $H = \mathbb{I}_{\Delta_n} + \frac{1}{\eta} R$. With these notations, one can rewrite Eq. (6.5) as

$$\max_{\bar{\boldsymbol{\ell}}} \ -J(-\bar{\boldsymbol{\ell}}^\top \bar{C}) - \mathbb{I}_{\Delta_n}(\bar{\boldsymbol{\ell}}) - \frac{1}{\eta} R(\bar{\boldsymbol{\ell}}) =: \max_{\bar{\boldsymbol{\ell}}} \ -J(-\bar{\boldsymbol{\ell}}^\top \bar{C}) - H(\bar{\boldsymbol{\ell}}), \tag{6.6}$$

where $\bar{C} = \begin{bmatrix} \bar{\boldsymbol{c}}_1 & \bar{\boldsymbol{c}}_2 & \ldots \end{bmatrix} \in \mathbb{R}^{n \times \bar{\mathcal{C}}}$ is the matrix whose column vectors are elements of $\bar{\mathcal{C}}$. By Theorem 2.1, the dual problem of Eq. (6.6) is

$$\min_{\boldsymbol{\lambda}} \ J^\star(\boldsymbol{\lambda}) + H^\star(\bar{C}\boldsymbol{\lambda}) = \min_{\boldsymbol{\lambda} \in \Delta_{\bar{\mathcal{C}}}} \ H^\star(\bar{C}\boldsymbol{\lambda}). \tag{6.7}$$

Since $\operatorname{dom} J = \mathbb{R}^{\mathcal{C}}$ and $\operatorname{dom} \mathbb{I}_{\Delta_n} + (1/\eta)R = \Delta_n$, with a similar argument in Chapter 3, we have $\mathbf{0} \in \operatorname{core}\left(\operatorname{dom} J - C^\top \operatorname{dom} \mathbb{I}_{\Delta_n} + (1/\eta)R\right)$ so that the strong duality holds. Since

H is $1/\eta$-strongly convex w.r.t. ℓ_1-norm, its conjugate $H^\star$ is η-smooth w.r.t. ℓ_∞-norm. The objective function $H^\star$ is a smoothified version of the one in Eq. (6.1).

The smoothness of the objective function is a key property for our analysis, so we aim to solve Eq. (6.7) instead of Eq. (6.1). Algorithm 8 summarizes our method, which maintains the quantity ϵ_k defined in Line 4.

Lemma 6.1. *Let $\boldsymbol{x} \in \mathrm{relax}(\mathcal{C})$ be an input to Algorithm 8 and let $\epsilon_k = H^\star(\bar{C}^{(k)}\boldsymbol{\lambda}) - \max_{j\in[k]} \bar{\boldsymbol{c}}_j \cdot \bar{\boldsymbol{\ell}}_{j-1}$ be the optimality gap defined in Line 4 of Algorithm 8. If $\eta \geq 2\ln(n)/\epsilon$, $\epsilon_k \leq \epsilon/2$ implies $\mathbb{E}_{\boldsymbol{c}\sim\boldsymbol{\lambda}_k}[\boldsymbol{c}] \cdot \boldsymbol{\ell} \leq (\alpha+\epsilon)\boldsymbol{x}\cdot\boldsymbol{\ell}$ for all $\boldsymbol{\ell} \in [0,M]^n$.*

Proof. Recall that $\bar{\boldsymbol{\ell}}_k = \nabla H^\star(\bar{C}^{(k)}\boldsymbol{\lambda}_k) = \arg\max_{\bar{\boldsymbol{\ell}}\in\Delta_n} \bar{\boldsymbol{\ell}}^\top \bar{C}^{(k)}\boldsymbol{\lambda}_k - \frac{1}{\eta}R(\bar{\boldsymbol{\ell}})$. By definition of $H^\star$, the following holds for all $\boldsymbol{\ell} \in [0,M]^n$.

$$\begin{aligned} H^\star(\bar{C}^{(k)}\boldsymbol{\lambda}_k) &= \bar{\boldsymbol{\ell}}_k^\top \bar{C}^{(k)}\boldsymbol{\lambda}_k - \frac{1}{\eta}R(\bar{\boldsymbol{\ell}}_k) \\ &\geq \bar{\boldsymbol{\ell}}^\top \bar{C}^{(k)}\boldsymbol{\lambda}_k - \frac{1}{\eta}R(\bar{\boldsymbol{\ell}}) \geq \boldsymbol{\ell}^\top C^{(k)}\boldsymbol{\lambda}_k - \frac{1}{\eta}\ln n \geq \mathbb{E}_{\boldsymbol{c}\sim\boldsymbol{\lambda}_k}[\boldsymbol{\ell}\cdot\boldsymbol{c}] - \frac{\epsilon}{2}. \end{aligned}$$

On the other hand, by the assumption of the $\mathcal{B}$, we have

$$\begin{aligned} \max_{j\in[k]} \bar{\boldsymbol{\ell}}_j \cdot \bar{\boldsymbol{c}}_{j+1} = \max_{j\in[k]} \boldsymbol{\ell}_j \cdot \boldsymbol{c}_{j+1} &\leq \max_{j\in[k]} \alpha \min_{\boldsymbol{p}\in\mathrm{relax}(\mathcal{C})} \boldsymbol{\ell}_j \cdot \boldsymbol{p} \\ &\leq \alpha \max_{j\in[k]} \boldsymbol{\ell}_j \cdot \boldsymbol{x} = \alpha \max_{j\in[k]} \|\bar{\boldsymbol{\ell}}_j\|_1 = \alpha, \end{aligned}$$

where the second inequality comes from the fact that $\boldsymbol{x} \in \mathrm{relax}(\mathcal{C})$. Combining the above relations, we have

$$\epsilon_k = H^\star(\bar{C}^{(k)}\boldsymbol{\lambda}_k) - \max_{j\in[k]} \bar{\boldsymbol{c}}_{j+1}\cdot\bar{\boldsymbol{\ell}}_j \geq \mathbb{E}_{\boldsymbol{c}\sim\boldsymbol{\lambda}_k}[\boldsymbol{\ell}\cdot\boldsymbol{c}] - \frac{\epsilon}{2} - \alpha, \qquad \forall \boldsymbol{\ell} \in [0,M]^n.$$

Thus, $\epsilon_k \leq \epsilon/2$ implies $\mathbb{E}_{\boldsymbol{c}\sim\boldsymbol{\lambda}_k}[\boldsymbol{\ell}\cdot\boldsymbol{c}] \leq \alpha+\epsilon = (\alpha+\epsilon)\|\bar{\boldsymbol{\ell}}\|_1 = (\alpha+\epsilon)\boldsymbol{x}\cdot\boldsymbol{\ell}$ for all $\boldsymbol{\ell} \in [0,M]^n$, which is the relation we desired. □

Lemma 6.1 assures that the output $\boldsymbol{\lambda}^\star$ of Algorithm 8 is an ϵ-approximate solution for Eq. (6.1). Thus, we can focus on proving how quickly the optimality gap converges. The following lemma shows the convergence rate.

Lemma 6.2. *For all $\{\mu_k\}_{k\geq 1} \subset [0,1]$, the sequence $\{\epsilon_k\}_{k\geq 1}$ generated by Algorithm 8 satisfies the following relation.*

$$\forall k \geq 1, \quad \epsilon_{k+1} \leq (1-\mu_k)\,\epsilon_k + 2\eta\mu_k^2 M^2 \mathrm{diam}_\infty(\mathcal{C})^2. \tag{6.8}$$

Proof. First of all, $\boldsymbol{\lambda}_{k+1}$ is chosen to optimize

$$\min_{\boldsymbol{\lambda}\in\Delta_{k+1}} H^\star(\bar{C}^{(k+1)}\boldsymbol{\lambda}), \qquad \bar{C}^{(k+1)} = \begin{bmatrix}\bar{\boldsymbol{c}}_1 & \bar{\boldsymbol{c}}_2 & \dots & \bar{\boldsymbol{c}}_{k+1}\end{bmatrix} \in [0, M\mathrm{diam}_\infty(\mathcal{C})]^{n\times(k+1)},$$

so that $H^\star(\bar{C}^{(k+1)}\boldsymbol{\lambda}_{k+1}) \le H^\star(\bar{C}^{(k+1)}\boldsymbol{\lambda})$ holds for all interior points $\boldsymbol{\lambda} = \boldsymbol{\lambda}_k + \mu_k\left(\boldsymbol{e}_{\bar{\boldsymbol{c}}_{k+1}} - \boldsymbol{\lambda}_k\right)$ with $\mu_k \in [0,1]$. Thus, using the η-smoothness of $H^\star$,

$$\begin{aligned}
\epsilon_k - \epsilon_{k+1} &\ge H^\star(\bar{C}^{(k)}\boldsymbol{\lambda}_k) - H^\star(\bar{C}^{(k+1)}\boldsymbol{\lambda}) \\
&\ge -\mu_k \nabla H^\star(\bar{C}^{(k+1)}\boldsymbol{\lambda}_k)^\top \bar{C}^{(k+1)}\left(\boldsymbol{e}_{\bar{\boldsymbol{c}}_{k+1}} - \boldsymbol{\lambda}_k\right) \\
&\quad - \frac{\eta}{2}\mu_k^2 \|\bar{C}^{(k+1)}(\boldsymbol{e}_{\bar{\boldsymbol{c}}_{k+1}} - \boldsymbol{\lambda}_k)\|_\infty^2 \\
&\ge -\mu_k \bar{\boldsymbol{\ell}}_k \bar{C}^{(k+1)}\left(\boldsymbol{e}_{\bar{\boldsymbol{c}}_{k+1}} - \boldsymbol{\lambda}_k\right) - 2\eta\mu_k^2 M^2 \mathrm{diam}_\infty(\mathcal{C})^2 \\
&= -\mu_k \bar{\boldsymbol{\ell}}_k \cdot \bar{\boldsymbol{c}}_{k+1} + \mu_k \bar{\boldsymbol{\ell}}_k^\top \bar{C}^{(k)}\boldsymbol{\lambda}_k - 2\eta\mu_k^2 M^2 \mathrm{diam}_\infty(\mathcal{C})^2 \\
&\ge -\mu_k \bar{\boldsymbol{\ell}}_k \cdot \bar{\boldsymbol{c}}_{k+1} + \mu_k\left[\bar{\boldsymbol{\ell}}_k^\top \bar{C}^{(k)}\boldsymbol{\lambda}_k - \frac{1}{\eta}R(\bar{\boldsymbol{\ell}}_k)\right] - 2\eta\mu_k^2 M^2 \mathrm{diam}_\infty(\mathcal{C})^2 \\
&= -\mu_k \bar{\boldsymbol{\ell}}_k \cdot \bar{\boldsymbol{c}}_{k+1} + \mu_k H^\star(\bar{C}^{(k)}\boldsymbol{\lambda}_k) - 2\eta\mu_k^2 M^2 \mathrm{diam}_\infty(\mathcal{C})^2 \\
&\ge \mu_k\epsilon_k - 2\eta\mu_k^2 M^2 \mathrm{diam}_\infty(\mathcal{C})^2.
\end{aligned}$$

Rearraging terms, we get the recurrence relation. □

Theorem 6.1. *Algorithm 8 terminates in $K = O\left(M^2\mathrm{diam}_\infty(\mathcal{C})^2 \ln(n)/\epsilon^2\right)$ iterations.*

Proof. Since Lemma 6.2 does not specify the sequence $\{\mu_k\}_{k\ge1} \subset [0,1]$, we choose $\mu_k = \frac{2}{k+2}$. Applying Lemma 2.1 with $c = 2\eta M^2\mathrm{diam}_\infty(\mathcal{C})^2$, we have

$$\epsilon_k \le \frac{8\eta M^2 D_\infty^2}{k+2}. \tag{6.9}$$

Setting the right-hand-side of Eq. (6.9) to be at most $\epsilon/2$, we $\epsilon_K \le \epsilon/2$ is achieved after $K \ge 16\eta M^2\mathrm{diam}_\infty(\mathcal{C})^2/\epsilon \ge 32M^2\mathrm{diam}_\infty(\mathcal{C})^2\ln(n)/\epsilon^2$ iterations. □

By Theorem 6.1, $\epsilon_K \le \frac{\epsilon}{2}$ holds after $K = O\left(M^2\mathrm{diam}_\infty(\mathcal{C})^2\ln(n)/\epsilon^2\right)$ iterations. Further, Lemma 6.1 guarantees that the output $\boldsymbol{\lambda}_K \in \Delta_{\mathcal{C}}$ satisfies $\mathbb{E}_{\boldsymbol{c}\sim\boldsymbol{\lambda}}[\boldsymbol{c}]\cdot\boldsymbol{\ell} \le (\alpha+\epsilon)\boldsymbol{x}\cdot\boldsymbol{\ell}$ for all $\boldsymbol{\ell}\in[0,M]^n$. Therefore, Algorithm 8 is a matarounding algorithm. We remark that if $\boldsymbol{x} = \mathbf{1}$, the convergence rate is the same as in ERLPBoost. In this sense, our algorithm can be seen as a generalized version of ERLPBoost. Furthermore, our analysis uses the Frank-Wolfe-like analysis, significantly simplifying the one ERLPBoost.

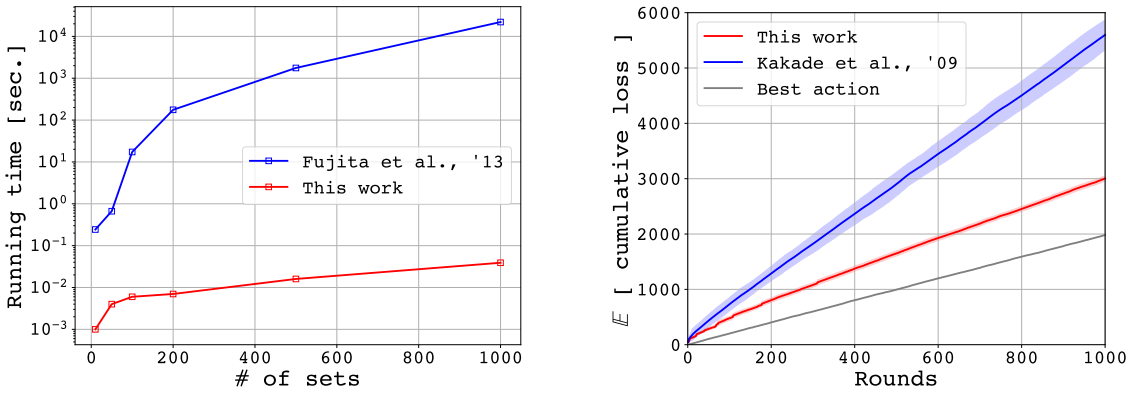

Figure 6.2: Comparison of previous works and our work. Both figure uses combinatorial vector of dimension $n = 20$. Left: comparison of running time. Right: comparison of cumulative losses. KKL '09 is the algorithm proposed by Kakade et al., 2009 [48]. In this experiment, we set $|\mathcal{C}| = 100$.

6.4 Experiment

The algorithm proposed by Garber [33] requires solving large-scale quadratic programs with $O(n^2\sqrt{T}\ln T)$ variables for $O(\sqrt{T})$-regret and $O(n^2 \ln T)$ variables for $O(T^{2/3})$-regret. Therefore, one cannot compute a combinatorial vector even though the number of oracle calls is sub-linear in T. For this reason, we compared our algorithm against to the algorithm by Kakade et al. [48] that achieves $O(\sqrt{T})$-regret[1]. We performed our experiment on Intel Xeon Gold 6124 CPU 2.60GHz processors.

Our experiment was demonstrated on a set cover instance, generated with the same procedure by Fujita et al. [30]. We choose a fixed item size $m = 10$ and measure the running time by varying the number of sets $n \in \{10, 50, 100, 200, 500, 1000\}$. The input for metarounding is a solution of relaxed set cover problem with a random cost vector $\boldsymbol{x} \in [0, 1]^n$. Here, the relaxation means that the variable takes value in $[0, 1]$ instead of $\{0, 1\}$.

Running time. We compared the algorithm proposed by Fujita et al. [30] and our algorithm. We use the Gurobi optimizer 9.0.1[2] to solve the sub-problems. To solve the entropy minimization, we used the sequential quadratic minimization approach; it first minimizes the quadratic approximation of the objective, then the objective function is updated to the approximation around the optimal solution. This procedure repeats until

[1]The program is available at `https://github.com/rmitsuboshi/metarounding_mitsuboshi`.
[2]`https://www.gurobi.com/`

convergent. As shown in the left side of Fig. 6.2, our algorithm is extremely fast compared to the others.

Cumulative loss. We compared the algorithm proposed by Kakade et al. [48] and our algorithm. In each round, the loss vector is generated randomly. Our algorithm uses the classic online gradient descent algorithm [38] over $[0,1]^n$ as the online linear optimization. The algorithm achieves $R_T(1) = O(\sqrt{T})$ over relax($\mathcal{C}$), so combining it with our metarounding achieves $R_T(\alpha) = O(\sqrt{T})$. As in the right side of Fig. 6.2, our algorithm is empirically superior to the other.

Chapter 7

Conclusion

This thesis studied the Frank-Wolfe-based boosting algorithms and their extention to other fields. Most of the proposed algorithms have a convergence guarantee that is at least the same as the one for the previous works. We designed the algorithms to be applicable to real-world datasets since their running times per round are fast.

This thesis started with developing a unified view for soft margin boosting algorithms, such as LPBoost, ERLPBoost, and the Corrective ERLPBoost. The view comes from Frank-Wolfe algorithms via the Fenchel duality. With this unified view, we proposed a general boosting algorithm that can be combined with arbitrary secondary algorithm. Using LPBoost as the secondary algorithm, the proposed boosting algorithm converges quickly without losing its convergence rate. The algorithm's convergence rate is the same as the ones for the previous works. Furthermore, the analysis simplifies the ones for ERLPBoost and the Corrective ERLPBoost.

Then, we used the technique of the Frank-Wolfe-based boosting to the regression boosting that handles a soft version of the insensitive loss. Applying the Frank-Wolfe view for boosting algorithms again, we proposed a boosting algorithm with a polynomial convergence guarantee even though the objective function is not smooth.

After that, we considered to apply the technique to other fields. The first one was for extended formulations. Extended formulations are the general approach for optimization problems that reduce the constraints by slightly increasing the variables without changing the optimal value and solutions. We proposed an extended formulation for optimization problems with linear constraints via Non-deterministic ZDD that is applicable not only to convex programs but also to mixed integer programs. Furthermore, we adopted the extended formulation to soft margin optimization and proposed a boosting scheme and

some algorithms. We also showed a convergence rate for one of the algorithms that depends on a parameter of the compressed data structure.

Finally, we applied the technique to Metarounding, a tool for the online combinatorial linear optimization problem. The existence of Metarounding implies an online algorithm whose α-regret is sub-linear in T. We proposed a Metarounding algorithm that is also designed based on the Frank-Wolfe-based boosting algorithm. The algorithm has a convergence guarantee, which is significantly better than the previous work under the same setting.

So far, we have diverted the technique of the Frank-Wolfe-based boosting algorithm to other fields, such as online learning. Some natural questions arises.

- Can we use the technique in other fields?
- Can we bring the technique back to the Frank-Wolfe algorithms?

Furthermore, the resulting algorithms have the same iteration bound to the ones in the Frank-Wolfe algorithms. Can we improve the convergence rate by specializing the analysis to the objective functions? There is a known lower bound for the hard margin boosting [59]. The lower bound is in terms of the number of hypotheses. The paper also showed an optimal boosting algorithm named SparsiBoost that runs AdaBoost* first to obtain a combined hypothesis and then sparsifies it to obtain one with consists of fewer hypotheses. AdaBoost* collects a hypothesis for every iteration, so the number of hypotheses to be combined matches the number of iterations. Since the iteration bound for AdaBoost* does not match the lower bound, SparsiBoost is not optimal regarding the number of rounds. There are some gaps between our bound and the lower bound. So, filling this gap is one of the future works.

Lastly, there is a known conjecture by Warmuth [98]: Can a perturbed version of LPBoost be ERLPBoost? We still believe one can design a perturbed LPBoost via the Frank-Wolfe-based approach with a polynomial convergence rate in expectation or with high probability.

Bibliography

[1] N. Abe, J. Takeuchi, and M. K. Warmuth. Polynomial Learnability of Stochastic Rules with Respect to the KL-Divergence and Quadratic Distance. *IEICE Transactions on Information and Systems*, E84-D(3):299–316, 2001.

[2] N. Alon, A. Gonen, E. Hazan, and S. Moran. Boosting Simple Learners. *TheoretiCS*, 2, 2023.

[3] A. Belloni. Introduction to bundle methods. Technical report, Technical report, Operation Research Center, MIT, 2005.

[4] J. L. Bentley and R. Sedgewick. Fast algorithms for sorting and searching strings. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms, (SODA 1997)*, pages 360–369, 1997.

[5] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. Hooker. *Decision Diagrams for Optimization*. Springer Cham, 2016.

[6] D. Bergman, W. J. van Hoeve, and J. N. Hooker. Manipulating MDD Relaxations for Combinatorial Optimization. In *Proceedings of the 8th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, (CPAIOR 2011)*, volume 6697 of *LNCS*, pages 20–35, 2011.

[7] J. M. Borwein and A. S. Lewis. *Convex Analysis*, pages 65–96. Springer New York, 2006.

[8] S. P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2014.

[9] L. Breiman. Prediction Games and Arcing Algorithms. *Neural Computation*, 11(7):1493–1517, 10 1999.

[10] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, aug 1986.

[11] M. D. Canon and C. D. Cullum. A Tight Upper Bound on the Rate of Convergence of Frank-Wolfe Algorithm. *SIAM Journal on Control*, 6(4):509–516, 1968.

[12] D. R. Carr and S. Vempala. Randomized metarounding. *Random Structure and Algorithms*, 20(1):343–352, 2002.

[13] R. D. Carr and S. S. Vempala. Randomized metarounding. *Random Struct. Algorithms*, 20(3):343–352, 2002.

[14] M. P. Castro, A. A. Cire, and J. C. Beck. An MDD-Based Lagrangian Approach to the Multicommodity Pickup-and-Delivery TSP. *INFORMS Journal on Computing*, 32(2):263–278, sep 2019.

[15] N. Cesa-Bianchi and G. Lugosi. Combinatorial Bandits. *Journal of Computer and System Sciences*, 78(5):1404–1422, 2012.

[16] C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(27):1–27, 2011.

[17] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (KDD 2016)*, page 785–794. Association for Computing Machinery, 2016.

[18] K. L. Clarkson. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. *ACM Trans. Algorithms*, 6(4):63:1–63:30, 2010.

[19] C. W. Combettes and S. Pokutta. Boosting Frank-Wolfe by Chasing Gradients. In *Proceedings of the 37th International Conference on Machine Learning, (ICML 2020)*, volume 119 of *Proceedings of Machine Learning Research*, pages 2111–2121. PMLR, 2020.

[20] M. Conforti, G. Cornuéjols, and G. Zambelli. Extended formulations in combinatorial optimization. *4OR*, 8(1):1–48, 2010.

[21] A. Demiriz, K. P. Bennett, and J. Shawe-Taylor. Linear Programming Boosting via Column Generation. *Machine Learning*, 46(1-3):225–254, 2002.

[22] C. Domingo and O. Watanabe. MadaBoost: A Modification of AdaBoost. In N. Cesa-Bianchi and S. A. Goldman, editors, *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory (COLT 2000)*, pages 180–189. Morgan Kaufmann, 2000.

[23] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7):2121–2159, 2011.

[24] N. Duffy and D. P. Helmbold. Boosting Methods for Regression. *Mach. Learn.*, 47(2-3):153–200, 2002.

[25] S. Fiorini, T. Huynh, and S. Weltge. Strengthening convex relaxations of 0/1-sets using Boolean formulas. *Mathematical Programming*, 190(1):467–482, 2021.

[26] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

[27] Y. Freund, R. D. Iyer, R. E. Schapire, and Y. Singer. An Efficient Boosting Algorithm for Combining Preferences. *J. Mach. Learn. Res.*, 4:933–969, 2003.

[28] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

[29] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5), 2001.

[30] T. Fujita, K. Hatano, and E. Takimoto. Combinatorial Online Prediction via Metarounding. In *Proceedings of 24th Annual Conference on Algorithmic Learning Theory (ALT 2013)*, volume 8139 of *LNCS*, pages 68–82, 2013.

[31] T. Fujita, K. Hatano, and E. Takimoto. Boosting over non-deterministic ZDDs. *Theoretical Computer Science*, 806:81–89, 2020.

[32] W. Gao and Z. Zhou. On the doubt about margin explanation of boosting. *Artif. Intell.*, 203:1–18, 2013.

[33] D. Garber. Efficient Online Linear Optimization with Approximation Algorithms. *Math. Oper. Res.*, 46(1):204–220, 2021.

[34] K. Goto, H. Bannai, S. Inenaga, and M. Takeda. Fast q-gram mining on SLP compressed strings. *Journal of Discrete Algorithms*, 18:89–99, 2013.

[35] A. Grønlund, L. Kamma, K. G. Larsen, A. Mathiasen, and J. Nelson. Margin-Based Generalization Lower Bounds for Boosted Classifiers. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, (NeurIPS 2019)*, pages 11940–11949, 2019.

[36] A. Grubb and D. Bagnell. Generalized Boosting Algorithms for Convex Optimization. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, (ICML 2011)*, pages 1209–1216. Omnipress, 2011.

[37] A. György, T. Linder, G. Lugosi, and G. Ottucsák. The On-Line Shortest Path Problem Under Partial Monitoring. *Journal of Machine Learning Research*, 8:2369–2403, 2007.

[38] E. Hazan. Introduction to Online Convex Optimization. *CoRR*, abs/1909.05207, 2019.

[39] E. Hazan and S. Kale. Projection-free Online Learning. In *Proceedings of the 29th International Conference on Machine Learning, (ICML 2012)*. icml.cc / Omnipress, 2012.

[40] E. Hazan and E. Minasyan. Faster Projection-free Online Learning. In J. D. Abernethy and S. Agarwal, editors, *Conference on Learning Theory, (COLT 2020)*, volume 125 of *Proceedings of Machine Learning Research*, pages 1877–1893. PMLR, 2020.

[41] D. P. Helmbold and M. K. Warmuth. Learning Permutations with Exponential Weights. *Journal of Machine Learning Research*, 10:1705–1736, 2009.

[42] M. Herbster and M. K. Warmuth. Tracking the Best Linear Predictor. *J. Mach. Learn. Res.*, 1:281–309, sep 2001.

[43] D. Hermelin, G. M. Landau, S. Landau, and O. Weimann. A Unified Algorithm for Accelerating Edit-Distance Computation via Text-Compression. In *Proceedings*

of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009), volume LIPIcs, pages 529–540, 2009.

[44] T. Inoue, K. Takano, T. Watanabe, J. Kawahara, R. Yoshinaka, A. Kishimoto, K. Tsuda, S.-i. Minato, and Y. Hayashi. Distribution Loss Minimization With Guaranteed Error Bound. *IEEE Transactions on Smart Grid*, 5(1):102–111, 2014.

[45] Jacques Guélat and Patrice Marcotte. Some comments on Wolfe's 'away step'. *Mathematical Programming*, 35:110–119, 1986.

[46] M. Jaggi. Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 427–435. JMLR.org, 2013.

[47] T. Jiang and B. Ravikumar. Minimal NFA Problems are Hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.

[48] S. Kakade, A. T. Kalai, and L. Ligett. Playing games with approximation algorithms. *SIAM Journal on Computing*, 39(3):1018–1106, 2009.

[49] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, (NIPS 2017)*, pages 3146–3154, 2017.

[50] J. E. Kelley, Jr. The Cutting-Plane Method for Solving Convex Programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

[51] K. C. Kiwiel. Proximity Control in Bundle Methods for Convex Nondifferentiable Minimization. *Math. Program.*, 46:105–122, 1990.

[52] D. E. Knuth. *The Art of Computer Programming, Volume 4A, Combinatorial Algorithms,Part 1.* Addison-Wesley Professional, 2011.

[53] Y. Kurokawa, R. Mitsuboshi, H. Hamasaki, K. Hatano, E. Takimoto, and H. Rahmanian. Extended Formulations via Decision Diagrams. In W. Wu and G. Tong, editors, *Computing and Combinatorics - 29th International Conference, COCOON*

2023, Hawaii, HI, USA, December 15-17, 2023, Proceedings, Part II, volume 14423 of *Lecture Notes in Computer Science*, pages 17–28. Springer, 2023.

[54] S. Lacoste-Julien and M. Jaggi. On the Global Linear Convergence of Frank-Wolfe Optimization Variants. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, (NIPS 2015)*, pages 496–504, 2015.

[55] C. Lemaréchal, A. Nemirovskii, and Y. E. Nesterov. New variants of bundle methods. *Math. Program.*, 69:111–147, 1995.

[56] Y. Lifshits. Processing Compressed Texts: A Tractability Border. In *Proceedings of the18th Annual Symposium on Combinatorial Pattern Matching (CPM 2007)*, volume 4580 of *LNCS*, pages 228–240, 2007.

[57] M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups - Complexity - Cryptology*, 4(2):241–299, 2012.

[58] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean. Boosting Algorithms as Gradient Descent. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12, (NIPS 1999)*, pages 512–518. The MIT Press, 1999.

[59] A. Mathiasen, K. G. Larsen, and A. Grønlund. Optimal Minimal Margin Maximization with Boosting. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, (ICML 2019)*, volume 97 of *Proceedings of Machine Learning Research*, pages 4392–4401. PMLR, 2019.

[60] S.-i. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Proceedings of the 30th international Design Automation Conference, (DAC 1993)*, pages 272–277, 1993.

[61] S.-i. Minato. Power of Enumeration — Recent Topics on BDD/ZDD-Based Techniques for Discrete Structure Manipulation. *IEICE Transactions on Information and Systems*, E100.D(8):1556–1562, 2017.

[62] S.-i. Minato and T. Uno. Frequentness-Transition Queries for Distinctive Pattern Mining from Time-Segmented Databases. In *Proceedings of the 2010 SIAM International Conference on Data Mining, (SDM 2010)*, pages 339–349, 2010.

[63] S.-i. Minato, T. Uno, and H. Arimura. LCM over ZBDDs: Fast Generation of Very Large-Scale Frequent Itemsets Using a Compact Graph-Based Representation. In *Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2008)*, volume 5012 of *LNAI*, pages 234–246, 2008.

[64] R. Mitsuboshi, K. Hatano, and E. Takimoto. Boosting as Frank-Wolfe. *arXiv*, 2209.10831, 2022.

[65] R. Mitsuboshi, K. Hatano, and E. Takimoto. Online combinatorial linear optimization via a Frank-Wolfe-based metarounding algorithm. *IEICE Transactions on Information and Systems*, advpub:2024FCP0006, 2024.

[66] R. Mitsuboshi, K. Hatano, and E. Takimoto. Solving Linear Regression with Insensitive Loss by Boosting. *IEICE Trans. Inf. Syst.*, 107(3):294–300, 2024.

[67] M. Mohri, A. Rostamizadeh, and A. Talwalker. *Foundation of Machine Learning*. The MIT Press, second edition, 2018.

[68] D. R. Morrison, E. C. Sewell, and S. H. Jacobson. Solving the Pricing Problem in a Branch-and-Price Algorithm for Graph Coloring Using Zero-Suppressed Binary Decision Diagrams. *INFORMS Journal on Computing*, 28(1):67–82, jan 2016.

[69] M. Nishino, N. Yasuda, S.-i. Minato, and M. Nagata. Accelerating Graph Adjacency Matrix Multiplications with Adjacency Forest. In *Proceedings of the 2014 SIAM International Conference on Data Mining, (SDM 2014)*, pages 1073–1081, 2014.

[70] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 1999.

[71] F. Pedregosa, G. Négiar, A. Askari, and M. Jaggi. Linearly Convergent Frank-Wolfe without Line-Search. In *The 23rd International Conference on Artificial Intelligence and Statistics, (AISTATS 2020)*, Proceedings of Machine Learning Research. PMLR, 2020.

[72] L. O. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. CatBoost: unbiased boosting with categorical features. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, (NeurIPS 2018)*, pages 6639–6649, 2018.

[73] J. G. Propp and D. B. Wilson. How to Get a Perfectly Random Sample from a Generic Markov Chain and Generate a Random Spanning Tree of a Directed Graph. *Journal of Algorithms*, 27(2):170–217, may 1998.

[74] R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning, (ICML 2009)*, pages 873–880, 2009.

[75] G. Rätsch, T. Onoda, and K. Müller. Soft Margins for AdaBoost. *Mach. Learn.*, 42(3):287–320, 2001.

[76] G. Rätsch and M. K. Warmuth. Efficient Margin Maximizing with Boosting. *Journal of Machine Learning Research*, 6:2131–2152, 2005.

[77] S. N. Ravi, M. D. Collins, and V. Singh. A Deterministic Nonsmooth Frank Wolfe Algorithm with Coreset Guarantees. *INFORMS Journal on Optimization*, 1(2):120–142, 2019.

[78] C. Rudin, I. Daubechies, and R. E. Schapire. The Dynamics of AdaBoost: Cyclic Behavior and Convergence of Margins. *J. Mach. Learn. Res.*, 5:1557–1595, 2004.

[79] W. Rytter. Grammar Compression, LZ-Encodings, and String Algorithms with Implicit Input. In *Proceedings of the 31st International Colloquium of Automata, Languages and Programming (ICALP 2004)*, volume 3142 of *LNCS*, pages 15–27, 2004.

[80] R. E. Schapire and Y. Freund. *Boosting: Foundations and Algorithms*. The MIT Press, 05 2012.

[81] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[82] R. E. Schapire and Y. Singer. Improved Boosting Algorithms Using Confidence-rated Predictions. *Mach. Learn.*, 37(3):297–336, 1999.

[83] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New Support Vector Algorithms. *Neural Comput.*, 12(5):1207–1245, 2000.

[84] H. Schramm and J. Zowe. A Version of the Bundle Idea for Minimizing a Nonsmooth Function: Conceptual Idea, Convergence Analysis, Numerical Results. *SIAM Journal on Optimization*, 2(1):121–152, 1992.

[85] R. A. Servedio. Smooth boosting and learning with malicious noise. *J. Mach. Learn. Res.*, 4:633–648, 2003.

[86] S. Shalev-Shwartz. *Online learning: theory, algorithms and applications*. PhD thesis, Hebrew University of Jerusalem, Israel, 2007.

[87] S. Shalev-Shwartz. Online Learning and Online Convex Optimization. *Found. Trends Mach. Learn.*, 4(2):107–194, 2012.

[88] S. Shalev-Shwartz and Y. Singer. On the equivalence of weak learnability and linear separability: new relaxations and efficient boosting algorithms. *Mach. Learn.*, 80(2-3), 2010.

[89] J. Shawe-Taylor and N. Cristianini. On the generalization of soft margin algorithms. *IEEE Trans. Inf. Theory*, 48(10):2721–2735, 2002.

[90] Y. Tabei, H. Saigo, Y. Yamanishi, and S. J. Puglisi. Scalable Partial Least Squares Regression on Grammar-Compressed Data Matrices. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (KDD 2016)*, pages 1875–1884, 2016.

[91] C. H. Teo, S. V. N. Vishwanathan, A. J. Smola, and Q. V. Le. Bundle Methods for Regularized Risk Minimization. *J. Mach. Learn. Res.*, 11:311–365, 2010.

[92] T. Toda. Fast Compression of Large-Scale Hypergraphs for Solving Combinatorial Problems. In *Proceedings of 16th International Conference on Discovery Science, (DS 2013)*, volume 8140 of *LNAI*, pages 281–293. Springer Berlin Heidelberg, 2013.

[93] T. Toda. ZCOMP: Fast Compression of Hypergraphs into ZDDs, 2015.

[94] K. Tsuji, K. Tanaka, and S. Pokutta. Pairwise Conditional Gradients without Swap Steps and Sparser Kernel Herding. In *International Conference on Machine Learning, (ICML 2022)*, volume 162 of *Proceedings of Machine Learning Research*, 2022.

[95] S. P. Vadhan and C. J. Zheng. A Uniform Min-Max Theorem with Applications in Cryptography. In *Proceedings of the 33rd Annual Cryptology Conference (CRYPTO 2013)*, volume 8042 of *Lecture Notes in Computer Science*, pages 93–110. Springer, 2013.

[96] C. Wang, Y. Wang, E. Weinan, and R. E. Schapire. Functional Frank-Wolfe Boosting for General Loss Functions. *arXiv*, 1510.02558, 2015.

[97] M. Warmuth, K. Glocer, and G. Rätsch. Boosting Algorithms for Maximizing the Soft Margin. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, pages 1585–1592, 2007.

[98] M. K. Warmuth. A perturbation that makes "Follow the leader" equivalent to "Randomized weighted majority", 2005.

[99] M. K. Warmuth, K. A. Glocer, and S. V. N. Vishwanathan. Entropy Regularized LPBoost. In *Proceedings of the 19th International Conference on Algorithmic Learning Theory, (ALT 2008)*, volume 5254, pages 256–271. Springer, 2008.

[100] M. K. Warmuth and D. Kuzmin. Randomized Online PCA Algorithms with Regret Bounds that are Logarithmic in the Dimension. *Journal of Machine Learning Research*, 9:2287–2320, 2008.

[101] M. K. Warmuth, J. Liao, and G. Rätsch. Totally corrective boosting algorithms that maximize the margin. In *Proceedings of the 23rd international conference on Machine learning (ICML 2006)*, pages 1001–1008, 2006.

[102] P. Wolfe. Convergence theory in nonlinear programming. *Integer and nonlinear programming*, pages 1–36, 1970.

[103] M. Yannakakis. Expressing combinatorial optimization problems by Linear Programs. *Journal of Computer and System Sciences*, 43(3):441–466, 1991.

[104] S. Yasutake, K. Hatano, S. Kijima, E. Takimoto, and M. Takeda. Online Linear Optimization over Permutations. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC 2011)*, volume 7074 of *LNCS*, pages 534–543, 2011.

[105] S. Yasutake, K. Hatano, E. Takimoto, and M. Takeda. Online Rank Aggregation. In S. C. H. Hoi and W. L. Buntine, editors, *Proceedings of the 4th Asian Conference on Machine Learning, (ACML 2012)*, volume 25 of *JMLR Proceedings*, pages 539–553. JMLR.org, 2012.