OPT2021: 13th Annual Workshop on Optimization for Machine Learning

A New Scheme for Boosting with an Average Margin Distribution Oracle

Ryotaro Mitsuboshi RYOTARO.MITSUBOSHI@INF.KYUSHU-U.AC.JP
Kohei Hatano HATANO@INF.KYUSHU-U.AC.JP
Eiji Takimoto EIJI@INF.KYUSHU-U.AC.JP
Kyushu University, Japan

Abstract

Boosting is a standard framework for learning a large-margin sparse ensemble of base hypotheses, where the algorithm assumes an oracle called the base learner, which returns a base hypothesis h with maximum edge $E_i[y_i h(\boldsymbol{x}_i)]$ with respect to a given distribution over the sample $((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m))$. In particular, for the ℓ_1-norm regularized soft margin optimization problem, several boosting algorithms have theoretical iteration bound for finding ϵ-approximate solutions. They are not as fast as classical LPBoost by Demiriz et al., which has no non-trivial iteration bound. In this paper, we propose a new boosting scheme. We assume a special base learner, which returns the average margin distribution vector $E_h[(y_i h(\boldsymbol{x}_i))_{i=1}^m]$ with respect to a certain distribution over the base hypothesis class H. Under this scheme, we propose a boosting algorithm whose iteration bound is $O((r \ln m)/\epsilon^2)$ and running time is $O(m \ln m)$ per iteration, where r is the VC dimension of H. Moreover, we also propose an efficient implementation for the new base learner, given that a relevant sub-class $H_{|S}$ of H is represented by a non-deterministic version of the Zero-suppressed Decision Diagram (NZDD), where NZDD is a data structure for representing a family of sets succinctly.

1. Introduction

Theory and algorithms for large-margin classifiers with sparse weights have been studied extensively. Those classifiers are guaranteed to have low generalization error when they have a large margin on the training samples (e.g., [9, 10]) and are helpful for feature selection purposes. A standard formulation for learning a sparse linear classifier is the 1-norm regularized soft margin optimization problem (defined later), a linear program (LP). Off-the-shelf LP solvers can solve this problem, but they are still not efficient enough for massive data. Boosting is a way to overcome this issue.

Some boosting algorithms optimize the soft margin problem, but these algorithms have theoretical or practical issues. For example, LPBoost [1] solves sub-problems of the soft margin problem iteratively (the column generation approach in the optimization literature). LPBoost works well experimentally, but there is no non-trivial iteration bound proven so far. Moreover, it needs to run $\Omega(m)$ iterations in the worst case [13], where m is the number of training examples. ERLPBoost [15], a variant of LPBoost, solves sub-problems with an entropic regularizer iteratively. It often works slower than LPBoost per iteration, but its iteration bound is shown to be $O(\log(m)/\epsilon^2)$ to obtain an ϵ-approximate solution. Thus, we are motivated to investigate provable boosting algorithms with both theoretical iteration guarantees and practical performances.

Table 1: The classical boostings assume that the base learner returns a max edge hypothesis for given distribution. Our work assumes a base learner that returns the average margin distribution. $H_{|S}$ is introduced in section 3, whose size is bounded by the VC dimension of H.

	LPBoost [1]	ERLPBoost [15]	SS algorithm [11]	Our work
Iteration bound	$\Omega(m)$	$O(\ln(m/\nu)/\epsilon^2)$	$O(\ln(m)/\epsilon^2)$	$O(\ln(\lvert H_{\lvert S}\rvert)/\epsilon^2)$
Problem per iter.	Linear Program	Ent. min.	Sorting	Sorting
Base Learner	Max edge	Max edge	Max edge	Average vector

This paper shows a new boosting scheme that reduces the 1-norm regularized soft margin optimization to strongly smooth function optimization. An advantage of this scheme is that we can apply Frank-Wolfe-based methods, for which only linear optimization is required at each iteration. In fact, by employing an entropic regularizer in the scheme, our boosting algorithm runs in $O((r \ln m)/\epsilon^2)$ iterations to obtain an ϵ-approximate solution, where r is the VC dimension of H. For the class of decision stumps over $\mathbb{R}^d$, we have a tighter iteration bound $O((\ln d + \ln m)/\epsilon^2)$, which is competitive to previous bounds. Furthermore, the linear optimization is reduced to a sorting problem for each iteration and can be solved in $O(m \log m)$ time. On the other hand, our scheme requires a base learner, which is different from the standard one. The standard base learner for the soft margin optimization returns a hypothesis with the maximum edge w.r.t. the given distribution. Instead, our base learner is supposed to return an "average margin distribution" over the sample w.r.t. a given distribution. For example, the average can be viewed as a soft-max of edges of hypotheses with the entropic regularizer. We show that our base learner runs efficiently, given that the restriction of hypotheses over the sample is represented by a data structure called Non-deterministic Zero-suppressed Decision Diagram (NZDD) [3], where NZDD is a variant of Zero Suppressed Decision Diagram (ZDD) [5, 7]. The time complexity for computing the average margin distribution depends linearly on the size of NZDD. Further, we also propose an efficient method to construct an NZDD when the hypotheses are decision stumps over ^{d}, whose time and space complexity are $O(dm \ln m)$ and $O(dm)$, respectively. Table 1 compares our work and others. Finally, we validate the effectiveness of the 1-norm soft margin formulation and the computational advantages of our proposed algorithm for real data sets with the base hypotheses of decision stumps.

2. Preliminaries

For notational simplicity, let $[k] = \{1, \ldots, k\}$ for $k \in \mathbb{N}$. We write m-dimensional probability simplex capped by $\nu \in [1, m]$ by $\Delta_{m,\nu} = \{\boldsymbol{d} \in [0, 1/\nu]^m \mid \|\boldsymbol{d}\|_1 = 1\}$. If $\nu = 1$, the capped simplex becomes the standard simplex so that we write Δ_m for shorthand. A function f is called β-strongly smooth w.r.t. ℓ_p-norm if for all $\boldsymbol{x}$ and $\boldsymbol{y}$, $f(\boldsymbol{y}) \leq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x}) \cdot (\boldsymbol{y} - \boldsymbol{x}) + \frac{\beta}{2}\|\boldsymbol{y} - \boldsymbol{x}\|_p^2$. We also write the indicator function for an event A as $\mathbb{I}_{[A]}$, that is, $\mathbb{I}_{[A]} = 1$ if A is true and 0 otherwise.

Soft margin optimization. Let $X \subseteq \mathbb{R}^d$ be a domain of interest. The soft margin maximization problem is a standard formulation for classification tasks [9]. Let $H \subseteq \{-1, +1\}^X$ be a set of hypotheses. Given a sequence of training examples $S = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)) \in (X \times \{-1, +1\})^m$ and a capping parameter $\nu \in [1, m]$, the objective of the soft margin maximization is

to find a combined hypothesis $f_{\boldsymbol{w}} = \sum_{h \in H} w_h h$ whose weight vector $\boldsymbol{w}$ is a solution of

$$\max_{\boldsymbol{w} \in \Delta_H, \boldsymbol{\xi} \geq \mathbf{0}} \min_{i \in [m]} \left[\sum_{w_h \in H} w_h u_i^h + \xi_i \right] - \frac{1}{\nu} \sum_{i=1}^m \xi_i, \tag{1}$$

where we use the notation $u_i^h = y_i h(\boldsymbol{x}_i)$ for shorthand. If H is a huge or infinitely large set, problem (1) cannot be solved directly. Boosting is a framework that overcomes this issue.

NZDD. Non-deterministic ZDD (NZDD, for shorthand) is a variant of the Zero-Suppressed Decision Diagram [5, 7], which is a data structure that represents a family of sets proposed in [3]. An NZDD G is a quadruple $G = (V, E, \Sigma, \psi)$, where (V, E) is a Directed Acyclic Graph (DAG) with a root and a leaf, and Σ is a ground set, $\psi : E \to 2^\Sigma$ is a function that assigns a set of labels to each edge $e \in E$. Let $\mathcal{P}_G$ be the set of all root-to-leaf path of G. For each path $P \in \mathcal{P}_G$, let $L(P) = \bigcup_{e \in P} \psi(e)$. Since $\psi(e) \subset \Sigma$, each path P represent a subset $L(P) \subset \Sigma$. Therefore, we can see that an NZDD is a data structure that represents a family of sets $L(G) := \{L(P) \mid P \in \mathcal{P}_G\}$.

3. A new scheme

In this section, we first present a new scheme that solves the soft margin maximization problem. Before we get to the main point, we first discuss the hypothesis class. Let S be the sequence of training examples defined in section 2, and let H be the hypothesis class the base learner picks from. We introduce an equivalence relation over H defined as $h_1 \equiv h_2 \iff \forall i \in [m], h_1(\boldsymbol{x}_i) = h_2(\boldsymbol{x}_i)$. Then, H can be partitioned into the equivalence classes $H_1 \cup H_2 \cup \cdots \cup H_p$ for some $p \in \mathbb{N}$. We define a set of representatives as $H_{|S} = \{h_1, \ldots, h_p\}$, where h_i is a representative of the equivalence class H_i. An optimal solution of problem (1) with respect to $H_{|S}$ is also an optimal solution of H. Therefore it is sufficient to consider $H_{|S}$ instead of H. Although the size of $H_{|S}$ is smaller than H, it may have exponentially many hypotheses.

3.1. Formulation

Our main idea is first to add a regularizer to the objective of (1), then take the dual form, and finally, solve the dual problem by the boosting-like method. We consider the regularized problem of (1),

$$\max_{\boldsymbol{w} \in \Delta_{H_{|S}}, \boldsymbol{\xi} \geq \mathbf{0}} \min_{i \in [m]} \left[\sum_{w_h \in H_{|S}} w_h u_i^h + \xi_i \right] - \frac{1}{\nu} \sum_{i=1}^m \xi_i - \frac{1}{\eta} \sum_{h \in H_{|S}} w_h \ln w_h \tag{2}$$

where $\eta > 0$ is a parameter. Now, we take the dual form of (2). The resulting objective function of the dual problem has a favorable property for its gradient.

Theorem 3.1 *The dual problem of (2) is*

$$\min_{\boldsymbol{d}} \frac{1}{\eta} \sum_{h \in H_{|S}} e^{\eta \boldsymbol{d} \cdot \boldsymbol{u}_h} \quad \textit{subject to} \quad \boldsymbol{d} \in \Delta_{m,\nu}. \tag{3}$$

We denote the objective function of (2) by J_η. Here, a notable difference between our algorithm and ERLPBoost is that ERLPBoost first takes the dual form of the soft margin optimization problem then solves its entropy regularized problem. In contrast, our algorithm solves the dual problem of the

entropy regularized soft margin optimization problem. The main difference between the classical boosting settings and ours is the role of the base learner. In our framework, the base learner returns the average margin distribution while the base learner in the classical settings returns a max edge hypothesis. If we choose $\eta > 0$ appropriately, we can obtain an ϵ-approximate solution of the dual problem of (1).

Lemma 3.1 *If $\eta \geq \frac{2}{\epsilon} \ln |H_{|S}|$, then an $\epsilon/2$-approximate solution of problem (3) is also an ϵ-approximate solution of the dual problem of the original problem (1).*

In general, the computation of $|H_{|S}|$ is hard. In such a case, we can use the upper bound of $|H_{|S}|$. Note that by Sauer's lemma (e.g., [9]), the size of $H_{|S}$ can be bounded by $(em/r)^r$, where r is the VC dimension of H so that $\eta = (2r/\epsilon)\ln(em/r)$ is a reasonable choice that satisfies the assumption of lemma 3.1. Furthermore, if the hypothesis class is the decision stump class, then the size of $|H_{|S}|$ is upper bounded by $2d(m+1)$, thus we can use $\eta = 2\ln(2d(m+1))/\epsilon$ as a parameter. We propose a new scheme to solve (3) by some convex optimization algorithm; for each iteration $t = 1, \ldots, T$, the booster sends the distribution $\boldsymbol{d}^t \in \Delta_{m,\nu}$ to the base learner. Then the base learner returns an average margin distribution $\nabla J_\eta(\boldsymbol{d}^t) = E_h[\boldsymbol{u}^h \mid \boldsymbol{d}^t]$, where $\Pr[h \mid \boldsymbol{d}^t] \propto e^{\eta \boldsymbol{d}^t \cdot \boldsymbol{u}^h}$ for all $h \in H_{|S}$. After T rounds, the algorithm outputs a combined hypothesis $f_{\boldsymbol{w}^T} = E_h[h \mid \boldsymbol{d}^T]$. This weighting is based on the primal-dual relation for the optimal solution of (3). Unfortunately, this conversion might not guarantee an ϵ-approximate solution of (1), but as $\boldsymbol{d}^T$ is close to the optimal solution of (3), $\boldsymbol{w}^T$ is also close to the optimal solution of (2). We give an efficient way to compute this average margin distribution $\nabla J_\eta(\boldsymbol{d}^t)$ in section 4. For now, we assume that there exists an efficient algorithm that computes the margin distributions. Under this assumption, we can use the convex optimization techniques for a strongly smooth function.

4. Algorithm for our scheme

In order to achieve fast computation per iteration, we apply the Frank-Wolfe algorithm [2]. This algorithm optimizes the convex function over a closed convex set by solving LP iteratively. Roughly speaking, the FW algorithm uses the gradient vector over the feasible region and the LP optimizer over the feasible region. We apply this algorithm to our optimization problem (3). In our case, the LP step can be achieved by sorting. The gradient computation step corresponds to the base learner call, and the rest steps correspond to updating the distribution step for boosting. We first show the algorithm's convergence rate for our case, and then we show how to compute the gradient vector and the linear programming step. The convergence rate of the FW algorithm is as follows:

Theorem 4.1 (Jaggi [4]) *Consider an optimization problem $\min_{\boldsymbol{x} \in \mathcal{D}} f(\boldsymbol{x})$, where D is a closed convex set, and f is β-strongly smooth function w.r.t. a norm $\|\cdot\|$. Let D be the diameter of $\mathcal{D}$ with respect to norm $\|\cdot\|$. Then, the FW algorithm converges in $O\left(\frac{\beta D}{t}\right)$ iterations.*

Since $\mathrm{diam}_{\|\cdot\|_1}(\Delta_{m,\nu}) \leq 2$ and $J_\eta(\boldsymbol{d})$ is η-strongly smooth w.r.t. ℓ_1-norm, we get the following corollary: the convergence rate for our case.

Corollary 4.1 *Applying the FW algorithm to problem (3), it finds an ϵ-approximate solution of the dual problem of (1) in $O\left(\frac{1}{\epsilon^2} \ln |H_{|S}|\right)$ iterations.*

4.1. Computation of the gradient

We now describe how to compute the base learner in the new boosting scheme. Our main idea is to construct an NZDD for $H_{|S}$ whose each root-to-leaf path corresponds to a vector in $H_{|S}$, and then we compute the required average margin distribution over the NZDD.

Construct an NZDD for $H_{|S}$. The first step depends on the structure of H. For the general $H_{|S}$, we enumerate the elements of $H_{|S}$ then use ZCOMP [1] [12]. However, this approach tends to generate a large NZDD. So it is better to construct a specialized NZDD for each $H_{|S}$. If the H is the decision stump class over $\mathbb{R}^d$, then we can construct the NZDD in $O(dm)$ time and $O(dm)$ space.

Computation of the average margin distribution. Let $G = (V, E, [m], \psi)$ be the NZDD the base learner has. We now describe how to compute the average margin distribution for any distribution $\boldsymbol{d} \in \Delta_{m,\nu}$ given to the base learner. This computation consists of two steps; First, we compute normalized weights over edges by the weight pushing algorithm [8]. Then we compute the weights on hypotheses by dynamic programming over the weighted NZDD. For the decision stump class, the computation over the NZDD is faster than the naive computation. The overall computation time of a gradient vector is $O(|G|)$, where $|G| = \sum_{e \in E} |\psi(e)|$.

5. Experiments

We use Gurobi optimizer 9.0.1 with an academic license to solve the sub-problems (LPs and QPs) of LPBoost and ERLPBoost. To solve the entropy minimization of ERLPBoost, we adopt the sequential quadratic programming as in [14]. We evaluate the computation times of LPBoost, ERLPBoost, SS algorithm, and our algorithm on some LIBSVM data sets [2]. We set the accuracy parameter $\epsilon = 0.01$ and the capping parameter $\nu = 0.8m$ and run each algorithm until meeting its stopping criterion. We set the ERLPBoost parameter η as $\eta = (2/\epsilon) \ln m$ and set the parameter η of our algorithm as the number of paths of the NZDD for $H_{|S}$. Table 2 shows that their computation times (seconds) for the data sets. In most cases, our algorithm terminates earlier than others, except for the RCV1 data set. Also, for the real sim data set, ours terminates much faster. A possible reason for these behaviors would be that the underlying problems are close to the classical Frank-Wolfe algorithm's best case or worst case. For instance, if the optimal solution is an extreme point in $\Delta_{m,\nu}$, the algorithm could reach it quickly. On the other hand, if the solution is not an extreme point but on the boundary of $\Delta_{m,\nu}$, the algorithm would require the worst-case bound iterations. More sophisticated variants of the Frank-Wolfe algorithm, e.g., [6], might avoid such worst situations.

Table 2: Comparison of the computation times (seconds) of boosting algorithms.

	Madelon	Covtype	Gissete	Real sim	RCV1
LPBoost	**6.49**	20.53	23.95	789.47	**1059.85**
ERLPBoost	33.68	55.52	36.23	591.43	1251.37
SS algo.	8.71	970.87	647.87	23.03	9893.03
Our work	9.56	**3.39**	**0.99**	**10.19**	3630.69

1. http://www.sd.is.uec.ac.jp/toda/code/zcomp.html
2. The LIBSVM data sets are from https://www.csie.ntu.edu.tw/~cjlin/libsvm/.

References

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

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

[3] Takahiro Fujita, Kohei Hatano, and Eiji Takimoto. Boosting over non-deterministic zdds. Theor. Comput. Sci., 806:81–89, 2020.

[4] Martin 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.

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

[6] Simon Lacoste-Julien and Martin 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, December 7-12, 2015, Montreal, Quebec, Canada, pages 496–504, 2015.

[7] Shin-ichi Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In Proceedings of the 30th international Design Automation Conference (DAC'93), pages 272–277, 1993.

[8] Mehryar Mohri and Michael Riley. A weight pushing algorithm for large vocabulary speech recognition. In EUROSPEECH 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event, pages 1603–1606. ISCA, 2001.

[9] Mehryar Mohri, Afshin Rostamizadeh, and Amee Talwalker. Foundation of Machine Learning. The MIT Press, second edi edition, 2018.

[10] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wen Sun Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. The Annals of Statistics, 26(5): 1651–1686, 1998.

[11] Shai Shalev-Shwartz and Yoram Singer. On the Equivalence of Weak Learnability and Linear Separability: New Relaxations and Efficient Boosting Algorithms. In Proceedings of the 21st Conference on Learning Theory (COLT '08), 2008.

[12] Takahisa Toda. Fast compression of large-scale hypergraphs for solving combinatorial problems. In Johannes Fürnkranz, Eyke Hüllermeier, and Tomoyuki Higuchi, editors, Discovery Science - 16th International Conference, DS 2013, Singapore, October 6-9, 2013. Proceedings, volume 8140 of Lecture Notes in Computer Science, pages 281–293. Springer, 2013.

[13] 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.

[14] Manfred K Warmuth, Jun Liao, and Gunnar Rätsch. Totally corrective boosting algorithms that maximize the margin. In Proceedings of the 23rd international conference on Machine learning (ICML '06), pages 1001–1008, 2006.

[15] Manfred K. Warmuth, Karen A. Glocer, and S. V. N. Vishwanathan. Entropy regularized lpboost. In Algorithmic Learning Theory, 19th International Conference, ALT 2008, Budapest, Hungary, October 13-16, 2008. Proceedings, volume 5254 of Lecture Notes in Computer Science, pages 256–271. Springer, 2008.

Appendix A. Algorithms and proofs

In this Appendix, we show the main algorithm and NZDDs.

A.1. Proof of lemma 3.1

Since the dual problem is derived by simple calculation, we only prove the strong smoothness of $J_\eta(\boldsymbol{d})$. The hessian of $J_\eta(\boldsymbol{d})$ is

$$\nabla^2 J_\eta(\boldsymbol{d}) = \eta E_{h\in H_{|S}}\left[\boldsymbol{u}^h {\boldsymbol{u}^h}^\top \middle| \boldsymbol{d}\right] - \eta E_{h\in H_{|S}}\left[\boldsymbol{u}^h \mid \boldsymbol{d}\right] E_{h\in H_{|S}}\left[\boldsymbol{u}^h \mid \boldsymbol{d}\right]^\top. \tag{4}$$

Using the fact that $\boldsymbol{u}^h \in \{-1,+1\}^m$ for all $h \in H_{|S}$, we get

$$\begin{aligned}\boldsymbol{w}^\top \nabla^2 J_\eta(\boldsymbol{d})\boldsymbol{w} =& \eta E_{h\in H_{|S}}\left[(\boldsymbol{w}\cdot\boldsymbol{u}^h)^2\right] - \eta\left(\boldsymbol{w}\cdot E_{h\in H_{|S}}[\boldsymbol{u}^h]\right)^2 \\ \le& \eta E_{h\in H_{|S}}\left[(\boldsymbol{w}\cdot\boldsymbol{u}^h)^2\right] \le \eta E_{h\in H_{|S}}\left[\|\boldsymbol{w}\|_1^2 \mid \boldsymbol{d}\right] = \eta\|\boldsymbol{w}\|_1^2\end{aligned}$$

for all $\boldsymbol{w}$ and $\boldsymbol{d} \in \Delta_{m,\nu}$. Therefore, for all $\boldsymbol{d}, \boldsymbol{d}^0 \in \Delta_{m,\nu}$, there exists a $\boldsymbol{z} \in \Delta_{m,\nu}$ such that

$$\begin{aligned}J_\eta(\boldsymbol{d}) =& J_\eta(\boldsymbol{d}^0) + (\boldsymbol{d}-\boldsymbol{d}^0)\cdot\nabla J_\eta(\boldsymbol{d}^0) + \frac{1}{2}(\boldsymbol{d}-\boldsymbol{d}^0)^\top \nabla^2 J_\eta(\boldsymbol{z})(\boldsymbol{d}-\boldsymbol{d}^0) \\ \le& J_\eta(\boldsymbol{d}^0) + (\boldsymbol{d}-\boldsymbol{d}^0)\cdot\nabla J_\eta(\boldsymbol{d}^0) + \frac{\eta}{2}\|\boldsymbol{d}-\boldsymbol{d}^0\|_1^2,\end{aligned}$$

where the first equality follows from Taylor's theorem.

A.2. Main algorithm

Algorithm 1 shows our boosting algorithm. We can regard this algorithm as the classical Frank-Wolfe algorithm that minimizes J_η over $\Delta_{m,\nu}$. Since the feasible region is $\Delta_{m,\nu}$, the linear programming step can be solved by sorting. We also terminate the for loop if the stopping criterion is satisfied. This stopping criterion comes from

$$J_\eta(\boldsymbol{d}^t) - J_\eta(\boldsymbol{d}^\star) \le (\boldsymbol{d}^t - \boldsymbol{d}^\star)\cdot J_\eta(\boldsymbol{d}^t) \le (\boldsymbol{d}^t - \boldsymbol{s}^t)\cdot J_\eta(\boldsymbol{d}^t),$$

where the first inequality follows from the convexity of J_η and the second inequality follows from the definition of $\boldsymbol{s}^t$. Therefore, if the stopping criterion is satisfied, it implies that $\boldsymbol{d}^t$ is an $\epsilon/2$-approximate solution.

Appendix B. Computation of average margin vector

Let $G = (V, E, [m], \psi)$ be the NZDD the base learner has. We now describe how to compute the average margin distribution for any distribution $\boldsymbol{d} \in \Delta_{m,\nu}$ given to the base learner. This computation consists of 2 steps; We first assign a weight to each edge $e \in E$ as $a(e) = \prod_{i\in\psi(e)} e^{2\eta d_i}$. After that, apply the weight pushing algorithm [8] to this weighted DAG (V, E, a). The input of the weight pushing algorithm is a DAG (V, E) and a function $a : E \to \mathbb{R}_+$ that assigns weight to each edge in E. Then, the weight pushing algorithm returns a function $b : E \to \mathbb{R}_+$ that satisfies

$$\forall P \in \mathcal{P}_G,\ \prod_{e\in P} b(e) = \frac{\prod_{e\in P} a(e)}{\sum_{P'\in\mathcal{P}_G}\prod_{e\in P'} a(e)}.$$

Algorithm 1: New boosting scheme with Frank-Wolfe

input : Training examples S, accuracy $\epsilon > 0$, capping parameter $\nu \in [1, m]$, and access to a base learner.

Initialize $\boldsymbol{d}^0 = (1/m)\mathbf{1}$, $\eta = \frac{2}{\epsilon} \ln |H_{|S}|$.

for $t = 0$ **to** T **do**

Send $\boldsymbol{d}^t$ to the base learner.

Receive the average margin distribution $\nabla J_\eta(\boldsymbol{d}) = E_{h \in H_{|S}} \left[\boldsymbol{u}^h \mid \boldsymbol{d} \right]$ from the base learner.

Let $\pi^t(1), \ldots, \pi^t(m)$ be the indices that satisfies
$\pi^t(i) \leq \pi^t(j) \iff \nabla J_\eta(\boldsymbol{d}^t)_i \leq \nabla J_\eta(\boldsymbol{d}^t)_j$.

Set $\forall i \in [m], s^t_{\pi^t(i)} = \begin{cases} 1/\nu & i \in \lfloor \nu \rfloor \\ 1 - (\lfloor \nu \rfloor / \nu) & i = \lfloor \nu \rfloor + 1 \\ 0 & \text{otherwise} \end{cases}$.

`// This corresponds to` $\boldsymbol{s}^t \in \arg\min_{\boldsymbol{s} \in \Delta_{m,\nu}} \boldsymbol{s} \cdot \nabla J_\eta(\boldsymbol{d}^t)$ `in FW algorithm.`

if $(\boldsymbol{d}^t - \boldsymbol{s}^t) \cdot \nabla J_\eta(\boldsymbol{d}^t) \leq \epsilon/2$ **then**
Set $T = t$, **break**.
end

Set $\lambda_t = \frac{2}{t+2}$ and update distribution $\boldsymbol{d}^{t+1} = \boldsymbol{d}^t + \lambda_t(\boldsymbol{s}^t - \boldsymbol{d}^t)$.

end

Compute $w_h^T = \frac{e^{\eta \boldsymbol{d}^T \cdot \boldsymbol{u}^h}}{\sum_{h' \in H} e^{\eta \boldsymbol{d}^T \cdot \boldsymbol{u}^{h'}}}, \quad \forall h \in H_{|S}$.

output: Combined hypothesis $f_{\boldsymbol{w}^T} = \sum_{h \in H_{|S}} w_h^T h$.

Since we assigned weight as above, after using the weight pushing algorithm, we can get a weighting function that satisfies By definition of $a : E \to \mathbb{R}$, the output function b satisfies

$$\forall P \in \mathcal{P}_G,\ \prod_{e \in P} b(e) = \frac{\prod_{e \in P} \prod_{i \in \psi(e)} e^{2\eta d_i}}{\sum_{P' \in \mathcal{P}_G} \prod_{e \in P'} \prod_{i \in \psi(e)} e^{2\eta d_i}}. \tag{5}$$

Let $b : E \to \mathbb{R}$ be the function returned by the weight pushing algorithm whose input is a weighting function $a : E \to \mathbb{R}$ defined above. Let $\mathcal{P}_G$ be the corresponding path from the root to the leaf. Then, we can express

$$\begin{aligned} \forall i \in [m],\ E_{\boldsymbol{u} \in U}[\boldsymbol{u} \mid \boldsymbol{d}] &= \sum_{\boldsymbol{u} \in U} \frac{e^{2\eta \boldsymbol{d} \cdot \boldsymbol{u}}}{\sum_{\boldsymbol{u}' \in U} e^{2\eta \boldsymbol{d} \cdot \boldsymbol{u}'}} u_i \\ &= \sum_{P \in \mathcal{P}_G} \left(\prod_{e \in P} \hat{w}(e) \right) \mathbb{I}_{[i \in \bigcup_{e' \in P} \psi(e')]} \\ &= \sum_{e \in E} \mathbb{I}_{[i \in \psi(e)]} \left(\sum_{P \in \mathcal{P}_G} \mathbb{I}_{[e' \in P]} \prod_{e \in P} \hat{w}(e) \right) =: \sum_{e \in E} \mathbb{I}_{[i \in \psi(e)]} q(e) \end{aligned}$$

where the third equality follows the fact that each label appears at most once for each path. Let $\mathcal{P}_G(u, v)$ be the set of paths from node u to v, and let $r, \ell \in V$ be the root node and leaf node,

Algorithm 2: Compute the average margin distribution

input : Distribution $\boldsymbol{d} \in \Delta_{m,\nu}$ over S and NZDD (V, E), where V is topologically sorted.
Assign weight to each edge $e \in E$ as $a(e) = \prod_{i \in \psi(e)} e^{2\eta d_i}$.
Let $b : E \to [0, 1]$ be the function returned by the weight pushing algorithm with weighting a.
Set $z(v_1) = 1$.
for $i = 2$ **to** $|V|$ **do**
  $z(v_i) = \sum_{e=(u,v_i) \in E} z(u)b(e)$.
end
For all $e = (u, v) \in E$, compute $q(e) = z(u)b(e)$.
Compute $r_i = \sum_{e \in E} \mathbb{I}_{[i \in \psi(e)]} q(e) \in [0, 1]$ for all $i \in [m]$.
output: $2\boldsymbol{r} - \mathbf{1}$.

respectively. By simple observation, we can see that for each edge $e = (u, v)$, $q(e)$ can be written as $q(e) = b(e) \sum_{P \in \mathcal{P}_G(r,u)} \prod_{e' \in P} b(e')$. Therefore, once we topologically sorted the vertices in V, we can compute $q(e)$ for all $e \in E$ by dynamic programming. The overall computation time of an average margin distribution for given distribution is $O(|G|)$. Algorithm 2 summarizes the procedures above.