UNIVERSITÀ DEGLI STUDI DI PADOVA

OPTIMIZATION FOR DATA SCIENCE

FRANK-WOLFE APPROACH TO BOOSTING ALGORITHMS

University of Padua

Department of Mathematics

Academic Year 2022/2023

Laura Legrottaglie

mat. 2073222

DIPARTIMENTO MATEMATICA

Abstract

Boosting is a powerful ensemble algorithm designed to create a strong learner by combining multiple weak learners. It is particularly useful for tackling problems involving the optimization of the $\ell 1$-norm regularized soft margin. Various boosting algorithms have been developed to address this problem, including LPBoost (Linear Programming Boosting) and ERLPBoost (Entropy Regularized LPBoost). LPBoost exhibits a fast $\mathcal{E}$-convergence rate, but can be computationally expensive in the worst-case scenario, requiring $\Omega(m)$, where m is the number of samples. On the other hand, ERLPBoost offers a $\mathcal{E}$-convergence rate of $O\left(\frac{1}{\mathcal{E}^2} ln\frac{m}{\nu}\right)$, which is theoretically efficient but may be slower in practical applications compared to LPBoost.

To address this efficiency challenge, Mitsuboshi et al. in 2022 demonstrated that LPBoost and ERLPBoost are specific instances of the Frank-Wolfe algorithm. They introduced a novel boosting framework that combines Frank-Wolfe with any secondary algorithm while maintaining the same convergence rate as ERLPBoost. In this new approach, they selected LPBoost as the secondary algorithm and coined the resulting algorithm MLPBoost (Modified LP-Boost). When tested on real-world datasets MLPBoost exhibited performance like LPBoost. After presenting the algorithms, this project aims to replicate them and test on a real dataset following the guidelines in [1] and [2].

1-Introduction

1.1-Boosting: general scheme

Boosting is a general framework that can be used to solve a binary classification problem. Suppose to have a set of m labeled training examples (x_i, y_i) where $x_i \in X$ and $y_i \in \pm 1$. Boosting algorithms give to each sample a weight d_i, resulting in a distribution of these weighted samples d residing within the m dimensional probability simplex P^m. The scheme starts by assigning the samples a uniform distribution and at each iteration, the algorithm "sends" the current distribution to an oracle, which gives back a new hypothesis (weak learner) h : X → [−1, 1] with a certain unknown assurance of performance g. We can measure the effectiveness of a weak learner by its *edge*, which we can think of as a weighted accuracy of the hypothesis.

$$\gamma_h = \sum_{i=1}^{m} d_i y_i h(x_i) = \sum_{i=1}^{m} d_i u_i = d \cdot u$$

where $u\ is\ a\ m - dimensional\ vector$ of ± 1 whose elements are the products of the true labels of the samples y_i and the predicted ones $h(x_i)$.

When we are in a binary classification problem, the edge is just a transformation of the weighted error ϵ_h.

$$\epsilon_h(d) = \frac{1}{2} - \frac{1}{2}\gamma_h$$

A hypothesis that classifies all the samples perfectly has edge $\gamma_h = 1$, while the worst hypothesis that always predicts wrong has edge $\gamma_h = -1$. A random hypothesis has $\gamma_h \approx 0$. The greater the edge, the more valuable the hypothesis becomes in classifying the training examples. The edge of a set of learners is the maximum edge of the hypotheses.
After obtaining a new hypothesis, the algorithm re-weights the training examples, putting more weight on examples that were poorly classified. For this reason, after the update, all the edges of the previous hypotheses go down.
The final output of the boosting algorithm is a convex combination of base hypotheses:

$$f_w(x_i) = \sum_{t=1}^{T} w_t h_t(x_i)$$

where each learner h_t has its own weight w_t. To "evaluate" a sample, it's possible to define the margin of an instance:

$$\rho_m = y_i f_w(x_i) = \sum_{t=1}^{T} y_i w_t h_t(x_i) = \sum_{t=1}^{T} w_t u_i^t$$

We can think of it as a weighted accuracy of the labeled example, showing as the sample is "behaving" in the overall frame. The margin of a set of examples is taken to be the minimum margin of the set, i.e. the margin of the worst sample. [2]

For the upcoming analysis, it's helpful to introduce the following notation:
$A = \left(y_i h_j(\boldsymbol{x_i})\right)_{i,j} \in [-1,+1]^{m\times n}$ is a matrix of size $m \times n$ containing the products of the true labels and the predicted ones.

1.2- Basic definitions and theorems

1.2.1- The Fenchel conjugate

The Fenchel conjugate f^* of a function $f\colon R^m \to [-\infty,+\infty]$ is defined as

$$f^\star(\theta) = \sup_{d\in R^m}\left(d^\top\theta - f(d)\right)$$

Roughly speaking, in R^2 given a slope θ, the conjugate function defines the offset needed for an affine function with that slope to be a supporting affine minorizer.

Moreover, if f is a strongly convex function, the gradient vector of f^* is

$$\nabla f^\star(\theta) = \arg\sup_{d\in R^m}\left(d^\top\theta - f(d)\right)$$

as shown in [3].

1.2.2- The Duality Theorem (Borwein and Lewis) [4]

Let $f\colon R^m \to (-\infty,+\infty]$ and

$g\colon R^n \to (-\infty,+\infty]$ be convex functions, and a linear map $A\colon R^m \to R^n$. The Fenchel problems

$$\gamma = \inf_{\boldsymbol{d}} f(d) + g(A^T d)$$
$$\rho = \sup_{\boldsymbol{w}} -f^*(-Aw) - g^*(w)$$

satisfy weak duality $\gamma \geq \rho$.

If $\mathbf{0} \in int(dom\, g - A^T dom\, f)$ then strong duality holds $\gamma = \rho$.

2-Duality in ℓ1-norm soft margin problem

The problem of ℓ1-norm margin optimization represents a way to discover classifiers with ample margins by formulating it as a linear problem (LP). To deal with the linearly inseparable cases, the problem relaxes the hard margin constraint and allows to the misclassification of some samples but penalizing them using slack variables ψ_i. In this case we are searching for the "soft" margin. To motivate this relaxation, let's make an example and consider a matrix *A* whose *i*th row is the negation of its *j* th row. In other words, our training dataset includes an instance that appears twice with contradictory labels. It's evident that this training set isn't linearly separable, even though the rest of the training set can be perfectly separated with a significant margin. To address this issue, we can create a probability distribution **d** with $d_i = d_j = \frac{1}{2}$ and the weights of all other samples equal to zero and note that $d^T A = \mathbf{0}$. The matrix A is not weak learnable (i.e. for all possible distributions *d* there exists a *j* s.t $|(d^T A)_j| \geq \gamma$, where $\gamma = \max_{\boldsymbol{w}} \min_{\boldsymbol{i\in[m]}} (Aw)_i$).
In the above example, the weak learnability assumption fails because we place excessive weights on the problematic examples *i, j.*

The ℓ1-norm soft margin primal problem can be defined as

$$\rho_t^*(\nu) = max_{w,\rho,\psi}(\rho - \frac{1}{\nu}\sum_{i=1}^{m}\psi_i)$$
$$s.t. \sum_{n=1}^{t} u_i^t w_n \geq \rho - \psi_i,\ for\ 1 \leq i \leq m,$$
$$w \in P^t, \psi \geq 0$$

In the primal problem, our objective is to maximize the minimum margin, which essentially entails identifying the most challenging example, the one with the narrowest margin, and then determining the optimal set of hypothesis weights that will improve its classification (maximization process).

The dual problem instead is formulated as follows:

$$\gamma_t^*(\nu) = min_{d,\gamma}\, \gamma \qquad (1)$$

$$s.t.\ d \cdot u^n \le \gamma,\ for\ 1 \le n \le t,$$
$$d \in P^m, d \le \frac{1}{\nu}\mathbf{1}$$

The dual problem aims to minimize the maximum edge when the distribution of the samples is capped with $\frac{1}{\nu}$, where $\nu \in \{1, \dots, m\}$. In fact, it is possible to prove that in the primal problem when the margin is ρ, $m - \nu$ samples have at least a margin of ρ and this corresponds to the capped distribution's constraint $\frac{1}{\nu}\mathbf{1}$ in the dual problem. The hard margin case is equivalent to do not have any capping constraints and can be considered setting $\nu = 1$. [5]

3.- LPBoost

We now examine one of the practical algorithms for solving the LP problem, that is called LPBoost. In their approach to tackling this problem, Demiriz et al. recommended employing, as the base learner within the algorithm, the one that exhibits the highest edge among all potential base learners. This strategy is aimed at accelerating the convergence process. [6] The hypothesis h that should be chosen is $h \in \underset{h\in H}{argmax} \sum_{i=1}^{m} d_i y_i h(x_i)$.

The LPBoost algorithm suggested by Wartmuth et al. in [2] is represented in figure 1.

The stopping condition of the algorithm guarantees that the error of the final classifier is less than the accuracy parameter δ. In fact, by weak-learnability assumption we know that at each iteration the oracle returns a hypothesis with an edge at least equal to a certain guarantee g, i.e $(d^{t-1}u^t) \ge g$.

3.1- Boosting from a game theory prospective

From a game theory point of view, we can represent boosting using a payoff matrix (a matrix that contains the results of players' choices in a game).

	Column player			
		w_1	w_2	w_3
Row player	d_1	0	1	-1
	d_2	-1	0	1
	d_3	1	-1	0

We can think that the rows are the examples, and the columns are the weak hypotheses. The sum of a row weighted according to the sample's weight d_i is the margin of an example, while the sum of a column weighted according to the hypothesis's weight w_j is the edge of a hypothesis. Therefore, the row player minimizes, and the column player maximizes. The payoff of the game (i.e. the outcome of the game that depends of the selected strategies of the players) is

$$payoff = d^T A w = \sum_{i,j} d_i A_{i,j} w_j$$

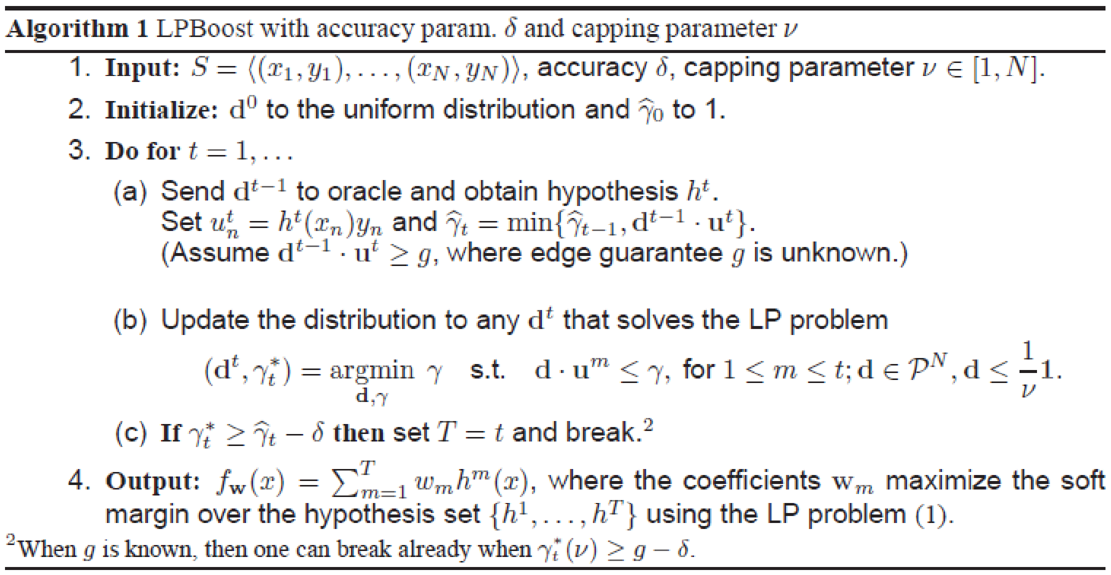
We can conceptualize the LPBoost algorithm as a two-step process. First, it solves a linear programming (LP) problem to determine an updated distribution d of the training samples. Subsequently, an oracle generates a new hypothesis, which effectively corresponds to adding a new column to the matrix A. This new column represents an additional weak learner in the boosting ensemble.

By Van Neumann's Min Max theorem, we can write the value of the game (i.e. the payoff of the game if all the players follow their optimal strategy) as:

$$value\ of\ the\ game = \underset{\boldsymbol{d}}{min}\ \underset{\boldsymbol{w}}{max}\ d^T A w$$
$$= \underset{\boldsymbol{w}}{maw}\ \underset{\boldsymbol{d}}{min}\ d^T A w$$

This means that the minimization of the maximum edge is equal to the maximization of the minimum margin.

Figure 1



Algorithm 1 LPBoost with accuracy param. δ and capping parameter ν

```
1. Input: $S = \langle(x_1, y_1), \dots, (x_N, y_N)\rangle$, accuracy $\delta$, capping parameter $\nu \in [1, N]$.
2. Initialize: $\mathbf{d}^0$ to the uniform distribution and $\widehat{\gamma}_0$ to 1.
3. Do for $t = 1, \dots$
    (a) Send $\mathbf{d}^{t-1}$ to oracle and obtain hypothesis $h^t$.
        Set $u_n^t = h^t(x_n)y_n$ and $\widehat{\gamma}_t = \min\{\widehat{\gamma}_{t-1}, \mathbf{d}^{t-1} \cdot \mathbf{u}^t\}$.
        (Assume $\mathbf{d}^{t-1} \cdot \mathbf{u}^t \geq g$, where edge guarantee $g$ is unknown.)
    (b) Update the distribution to any $\mathbf{d}^t$ that solves the LP problem
            $(\mathbf{d}^t, \gamma_t^*) = \operatorname*{argmin}_{\mathbf{d},\gamma} \gamma$  s.t.  $\mathbf{d} \cdot \mathbf{u}^m \leq \gamma$, for $1 \leq m \leq t$; $\mathbf{d} \in \mathcal{P}^N$, $\mathbf{d} \leq \frac{1}{\nu}\mathbf{1}$.
    (c) If $\gamma_t^* \geq \widehat{\gamma}_t - \delta$ then set $T = t$ and break.[2]
4. Output: $f_{\mathbf{w}}(x) = \sum_{m=1}^{T} w_m h^m(x)$, where the coefficients $\mathbf{w}_m$ maximize the soft
   margin over the hypothesis set $\{h^1, \dots, h^T\}$ using the LP problem (1).
```

[2]When g is known, then one can break already when $\gamma_t^*(\nu) \geq g - \delta$.

3.2- Edge minimization and its dual problem by Fenchel duality

Let $\mathcal{P}_\nu^m := \{\, d \in [0,1/\nu]^m \mid |\boldsymbol{d}|_1 = 1 \,\}$, where $\nu \in [1, m]$ *the* m-dimensional capped probability simplex, we can define the edge minimization problem as:

$$\min_{\boldsymbol{d}} \max_{j \in [n]} (\boldsymbol{d}^\top A)_j + f(\boldsymbol{d}),$$

$$\text{where} \quad f(\boldsymbol{d}) = \begin{cases} 0 & \boldsymbol{d} \in \mathcal{P}_\nu^m \\ +\infty & \boldsymbol{d} \notin \mathcal{P}_\nu^m \end{cases}.$$

Intuitively, we can think on the problem as the minimum of the maximal edge plus a regularization term *f (*d*)* that does not penalize distributions inside the capped probability simplex and places an infinite penalty for the rest of the distributions.
By Fenchel's duality theorem, if we consider the convex functions:

$$f(\boldsymbol{d}) = \begin{cases} 0 & \boldsymbol{d} \in \mathcal{P}_\nu^m \\ +\infty & \boldsymbol{d} \notin \mathcal{P}_\nu^m \end{cases}, \quad g(\boldsymbol{\theta}) = \max_{j \in [n]} \theta_j.$$

the dual problem of the edge minimization problem

$$\min_{\boldsymbol{d}} g(A^T d) + f(d)$$
$$= \min_{\boldsymbol{d}} \max_{\boldsymbol{j} \in [\boldsymbol{n}]} (d^T A)_j + \; f(d)$$

is the soft margin optimization problem:

$$\max_{\boldsymbol{w} \in \mathcal{P}^n} -f^\star(-A\boldsymbol{w})$$

and the duality gap between the two is zero as showed by Shalev-Shwartz and Singer in [3].

4- Limitations of LPBoost and ERLPBoost

Although the LPBoost algorithm is fast in practical applications, it has a significant limitation: it tends to assign all weight to the samples with the minimum margin. Consequently, the algorithm concentrates its efforts on challenging examples, giving them the utmost importance, and potentially failing to generalize effectively across the entire dataset. To address this limitation, Warmuth et al. introduced an enhanced variant of LPBoost known as Entropy Regularized LPBoost (ERLPBoost) [7]. ERLPBoost introduces a regularization term in the *d* domain and updates the distribution on the samples by solving the problem:

$$\min_{\boldsymbol{d}} \max_{\boldsymbol{j} \in [\boldsymbol{n}]} (d^T A)_j + \; f(d) \; + \frac{1}{\eta} \Delta(d)$$

where

$$\Delta(d) \; = \Delta(d, \tfrac{1}{m}\mathbf{1}) =$$

$$= \sum_{i=1}^{m} d_i \ln\left(\frac{d_i}{\frac{1}{m}}\right) =$$

$$= \sum_{i=1}^{m} d_i \ln(d_i) + d_i \ln(m)$$

is called the relative entropy from the uniform distribution $\frac{1}{m}\mathbf{1} \in \mathcal{P}_v^m$.

The relative entropy, also known as Kullback-Leibler divergence, is a measure of how a probability distribution diverges from another. Since we are considering the uniform distribution $\frac{1}{m}$, the relative entropy in this case is used to measure how far the probability distribution represented by vector *d* is different from the uniform distribution over the capped probability simplex.

In the subsequent discussion, we refer to $\tilde{f}(d) = f(d) + \frac{1}{\eta}\Delta(d)$ and we will show that ERLPBoost is an instance of Frank-Wolfe algorithm.

5- Frank-Wolfe algorithms

The Frank-Wolfe algorithm is a first-order iterative algorithm that solves problems of the form $\min_{x \in C} f(x)$ where $f: R^n \to R$ is a convex function with Lipschitz continuous gradient with constant $L > 0$ and $C \subseteq R^n$ is a convex compact set. At each iteration *t*, the algorithm finds a descent direction by solving the minimization problem:

$$\hat{x}_t = \min_{x \in C} \nabla f(x_t)^T (x - x_t)$$

and updates $\hat{x}_{t+1} = x_t + \lambda_t (\hat{x}_t - x)$ for some $\lambda_t \in (0,1]$.

5.1-Classical stepsize and Short-Step strategy

The classical strategy is to consider at each iteration as step size $\lambda_t = \frac{2}{t+2}$. Nevertheless, it's possible to choose different step sizes. For example, one can choose the *short-step* strategy which consists of choosing at each iteration λ_t as:

$$\lambda_t = \max\left\{0, \min\left\{1, \frac{(x_t - x_{t+1})^T \nabla f(x_t)}{\eta \| x_{t+1} - x_t \|^2}\right\}\right\}$$

5.2-Limitations of Classical Frank-Wolfe

The classical Frank-Wolfe algorithm exhibits sublinear convergence in general scenarios. However, when some specific conditions are satisfied (the feasible set *C* possesses specific structural properties, the objective function *f* is *σ*-strongly convex with a Lipschitz continuous gradient and the optimal solution x^* resides within the interior of the set C) it is possible to achieve a linear convergence rate.

Unfortunately, in many practical constrained optimization problems, the optimal solution tends to lie on the boundary of the feasible set. In such cases, the classical Frank-Wolfe algorithm becomes inefficient due to a phenomenon known as "zig-zagging." Specifically, when we take a polytope as the feasible set, the algorithm starts oscillating or "zig-zagging" between the vertices defining the face of the polytope that contains the solution x^*. [8]

This zig-zagging behavior hinders the convergence of the algorithm because it keeps oscillating along the edges of the polytope, making slow progress towards the optimal solution.

5.3-Away-steps Frank-Wolfe

Various modifications and extensions of the classical Frank-Wolfe algorithm have been proposed to improve the convergence. These methods aim to mitigate the zig-zagging phenomenon and accelerate convergence even in the presence of boundary solutions.

One of the improved variants of classical Frank-Wolfe algorithm is the Away-Step Frank-Wolfe algorithm.

To address the zig-zagging issue, Wolfe introduced a modification to achieve linear convergence in the case of strongly convex functions when dealing with a polyhedral feasible set *C*. Taking into account that each iterate *t* can be represented as a sparse convex combination of at most *t* vertices, the algorithm adds the possibility to move away from the "worst" vertex between the ones that represent the current iterate. It corresponds to the one with the highest value of the linearized function. Therefore, at each iteration, the algorithm computes two directions: the conventional Frank-Wolfe direction that goes towards the best vertex (i.e. the vertex that minimizes the linearized approximation of the objective function) and the away-step direction that points away from the worst vertex. These directions, then, are compared to finding the one that minimizes the most the objective function and the new starting point is chosen by moving according to the chosen direction by a suitable step size.

5.4- ERLPBoost as instance of Frank-Wolfe algorithm

As we have already mentioned ERLPBoost aims to solve the edge minimization problem

$$\min_{\boldsymbol{d}} \max_{\boldsymbol{j}\in[\boldsymbol{n}]} (d^T A)_j + \tilde{f}(d)$$

or its dual soft margin optimization problem:

$$\max_{\boldsymbol{w}\in\mathcal{P}^n} -\tilde{f}^\star(-A\boldsymbol{w})$$

By the definition of the Fenchel conjugate and its gradient we can write

$$\tilde{f}^*(-Aw) = \sup_{\boldsymbol{d}\in\boldsymbol{R}^m} (-d^T Aw - \tilde{f}(d))$$

$$\nabla\tilde{f}^*(-Aw) = \operatorname*{argsup}_{\boldsymbol{d}\in\boldsymbol{R}^m} (-d^T Aw - \tilde{f}(d)) = \operatorname*{argmin}_{\boldsymbol{d}\in\boldsymbol{R}^m} (d^T Aw + \tilde{f}(d))$$

Therefore, at each iteration *t* computing the gradient of the function $\tilde{f}^*$ is equivalent to find the distribution $d_t \in \mathcal{P}_\nu^m$ that minimizes $(d^T Aw + \tilde{f}(d))$. After we have found the distribution d_t, we want to obtain a new hypothesis $h_{j_{t+1}}$ that maximizes the edge, i.e $j_{t+1} \in \operatorname*{argmax}_{\boldsymbol{j}\in[\boldsymbol{n}]} (d^T A)_j$. Obtaining a new hypothesis $h_{j_{t+1}}$ is equivalent to find the basis vector $e_{j_{t+1}} \in P^n$ since the maximization problem is over the probability simplex P^n. Thus, we can write:

$$\arg\max_{\boldsymbol{e}_j : j\in[n]} \boldsymbol{d}_t^\top A\boldsymbol{e}_j = \arg\min_{\boldsymbol{e}_j : j\in[n]} (-A\boldsymbol{e}_j)^\top \nabla\tilde{f}^\star(\boldsymbol{\theta}_t)$$

This shows that finding a new hypothesis that maximizes the edge corresponds to solve a linear programming problem with Frank-Wolfe algorithm.

6. Modified LPBoost (MLPBoost)

The novel algorithm presented by Mitsuboshi et al. in [1] is a FW-inspired boosting scheme. It utilizes two distinct update rules: a primary FW update rule denoted as *F* and a secondary update rule referred to as *B*. Both rules generate a weight vector $w \in P^n$ for the base learners.

It's important to notice that the primary FW update rule serves as a safety mechanism to ensure convergence guarantees, offering the flexibility to incorporate any update rule for B. In the MLPBoost the secondary algorithm chosen is the LPBoost.

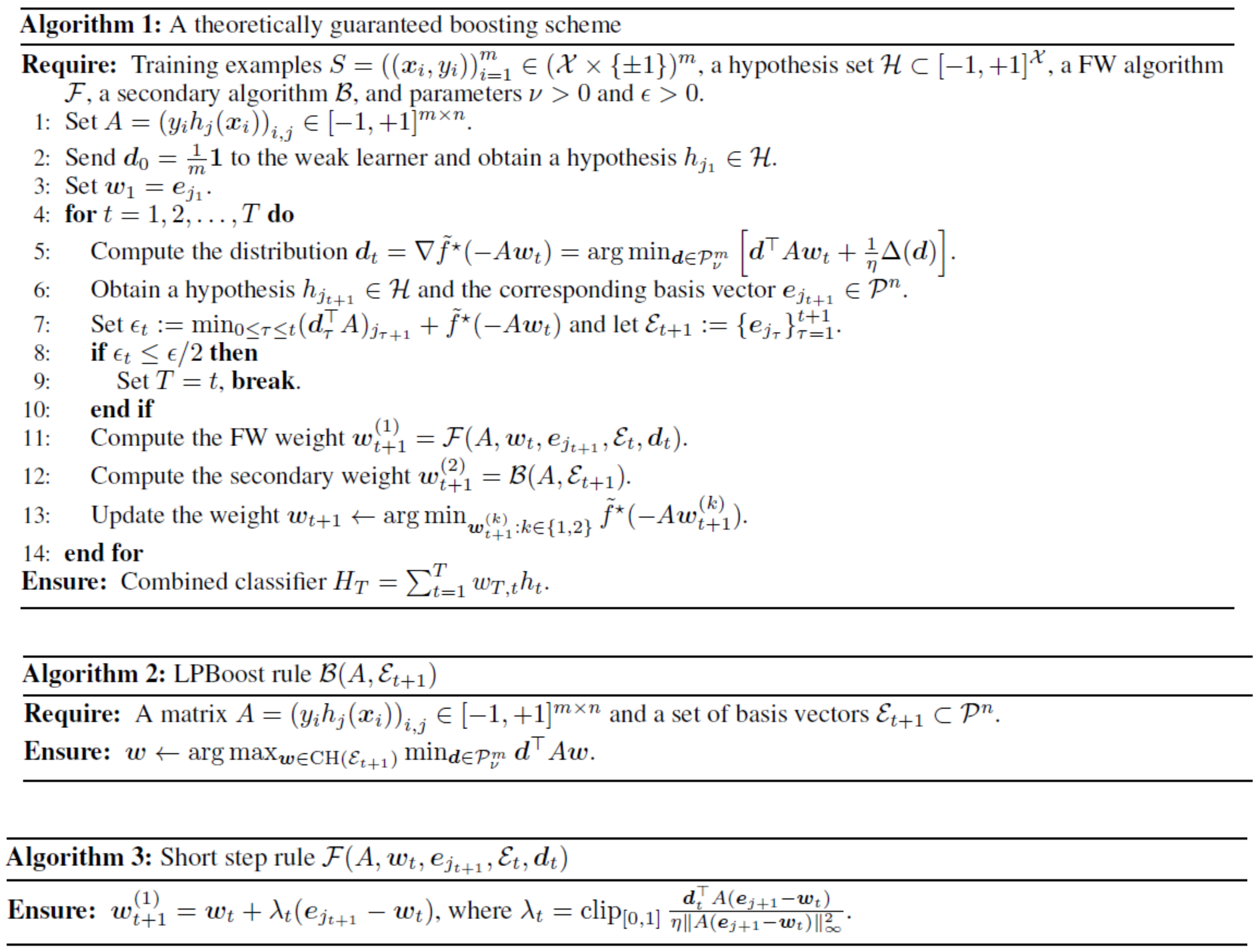
The MLPBoost algorithm is represented in figure 2.

Let's highlight some key aspects of the algorithm.

At line 5, as suggested in [1], the computation of the distribution d_t at each iteration can be done by a sorting-based algorithm proposed by Shalev-Shwartz and Singer in [5] as we explain in the following section.

At lines 7 and 8, the stopping condition is specified where ϵ_t is the optimality gap. In fact, by weak-learnability assumption we know that at each iteration the oracle returns a hypothesis with an edge at least equal to *g*, a

Figure 2



```
Algorithm 1: A theoretically guaranteed boosting scheme
Require: Training examples $S = ((x_i, y_i))_{i=1}^m \in (\mathcal{X} \times \{\pm 1\})^m$, a hypothesis set $\mathcal{H} \subset [-1,+1]^{\mathcal{X}}$, a FW algorithm
    $\mathcal{F}$, a secondary algorithm $\mathcal{B}$, and parameters $\nu > 0$ and $\epsilon > 0$.
 1: Set $A = (y_i h_j(x_i))_{i,j} \in [-1,+1]^{m \times n}$.
 2: Send $d_0 = \frac{1}{m}\mathbf{1}$ to the weak learner and obtain a hypothesis $h_{j_1} \in \mathcal{H}$.
 3: Set $w_1 = e_{j_1}$.
 4: for $t = 1, 2, \ldots, T$ do
 5:     Compute the distribution $d_t = \nabla \tilde{f}^\star(-Aw_t) = \arg\min_{d \in \mathcal{P}_\nu^m} \left[ d^\top A w_t + \frac{1}{\eta}\Delta(d) \right]$.
 6:     Obtain a hypothesis $h_{j_{t+1}} \in \mathcal{H}$ and the corresponding basis vector $e_{j_{t+1}} \in \mathcal{P}^n$.
 7:     Set $\epsilon_t := \min_{0 \le \tau \le t} (d_\tau^\top A)_{j_{\tau+1}} + \tilde{f}^\star(-Aw_t)$ and let $\mathcal{E}_{t+1} := \{e_{j_\tau}\}_{\tau=1}^{t+1}$.
 8:     if $\epsilon_t \le \epsilon/2$ then
 9:         Set $T = t$, break.
10:     end if
11:     Compute the FW weight $w_{t+1}^{(1)} = \mathcal{F}(A, w_t, e_{j_{t+1}}, \mathcal{E}_t, d_t)$.
12:     Compute the secondary weight $w_{t+1}^{(2)} = \mathcal{B}(A, \mathcal{E}_{t+1})$.
13:     Update the weight $w_{t+1} \leftarrow \arg\min_{w_{t+1}^{(k)} : k \in \{1,2\}} \tilde{f}^\star(-Aw_{t+1}^{(k)})$.
14: end for
Ensure: Combined classifier $H_T = \sum_{t=1}^T w_{T,t} h_t$.
```

```
Algorithm 2: LPBoost rule $\mathcal{B}(A, \mathcal{E}_{t+1})$
Require: A matrix $A = (y_i h_j(x_i))_{i,j} \in [-1,+1]^{m \times n}$ and a set of basis vectors $\mathcal{E}_{t+1} \subset \mathcal{P}^n$.
Ensure: $w \leftarrow \arg\max_{w \in \mathrm{CH}(\mathcal{E}_{t+1})} \min_{d \in \mathcal{P}_\nu^m} d^\top A w$.
```

```
Algorithm 3: Short step rule $\mathcal{F}(A, w_t, e_{j_{t+1}}, \mathcal{E}_t, d_t)$
Ensure: $w_{t+1}^{(1)} = w_t + \lambda_t(e_{j_{t+1}} - w_t)$, where $\lambda_t = \mathrm{clip}_{[0,1]} \frac{d_t^\top A(e_{j+1} - w_t)}{\eta \|A(e_{j+1} - w_t)\|_\infty^2}$.
```

certain guarantee, i.e $(d^T A)_{j_{t+1}} \geq g$. By setting $\eta = \frac{2}{\epsilon} ln(\frac{m}{\nu})$, we can write:

$$\epsilon_t = min_{0 \le \tau \le t} (d^T A)_{j_{t+1}} + \tilde{f}^*(-Aw_t) \geq g + \tilde{f}^*(-Aw_t) \leq \frac{\epsilon}{2}$$

By last inequality:

$$-\tilde{f}^*(-Aw_t) \geq g - \epsilon$$

At line 13, after we have computed the weight according to the FW rule and the LPBoost rule, we check which weight vector is the best for the update. This evaluation is done by the comparison of the objective function that we want to maximize $-\tilde{f}^*(-Aw)$ or by the edges of the result classifier obtained with the different updates.

6.1- Computation of d_t on the capped probability simplex $\mathcal{P}_\nu^m$

It's possible to prove that when *f(d)* is defined on the probability simplex P^m then the distribution d_t is the gradient of the Fenchel conjugate of the function $\frac{1}{\eta}$ Δ(d).

The Fenchel conjugate of the relative entropy is equal to

$$h^\star(\boldsymbol{\theta}) = \log\left(\frac{1}{m}\sum_{i=1}^m e^{\theta_i}\right)$$

and considering the property $(ch)^*(\theta) = c\, h^*(\theta/c)$ where c is a constant, we obtain:

$$d_{t,i} \propto e^{-\eta(Aw_t)_i}$$

This shows that the log of the sample weight is negative proportional to the margin of the example $(Aw_t)_i$ according to the current vector of the hypotheses' weights w_t. Therefore, a sample with a larger margin will have a weight smaller than an example with a smaller margin (i.e. the algorithm assigns more weight to the erroneous samples).

The problem now is to find the distribution d_t on the capped probability simplex $\mathcal{P}_\nu^m$.

The method proposed in [5] is based on Bregman divergences and their properties. The Bregman divergence with respect to a convex function h between two vectors d and d_0 is defined as:

$$B_h(\mathbf{d}\|\mathbf{d}_0) = h(\mathbf{d}) - h(\mathbf{d}_0) - \langle \nabla h(\mathbf{d}_0), \mathbf{d} - \mathbf{d}_0 \rangle$$

Since the distribution d_t *is* equal to the gradient of the Fenchel conjugate of the function $\tilde{f}$ we can write:

$$d_t = \nabla(h+f)^*(\theta) =$$

$$\underset{\boldsymbol{d}\in \mathcal{P}_\nu^m}{argmin}\,(B_h(d||\,\nabla h*(\theta)) =$$

$$\underset{\boldsymbol{d}\in \mathcal{P}_\nu^m}{argmin}\,(B_h(d||\,\nabla h*(\theta)) =$$

$$\underset{\boldsymbol{d}\in \mathcal{P}_\nu^m}{argmin}\,(B_h(d||\,e^{-\eta(Aw_t)_i}\,) =$$

$$\underset{\boldsymbol{d}\in \mathcal{P}_\nu^m}{argmin}\,(B_h(d||\,d_0) \qquad (2)$$

where $\nabla h*(\theta) = e^{-\eta(Aw_t)_i}$ with

$$\theta = -Aw_t.$$

Thus, to find the distribution d_t we start by defining the probability vector $d_0 \propto e^{-\eta(Aw_t)_i}$ and then we use entropic projection to find the distribution d that is closer to d_0.

6.2-Entropic projection

It can be demonstrated that the optimal solution of the problem (2) is of the form $d_t = (\nu, \nu, \dots, \nu, d_{t,i}, \dots, d_{t,m})$ where $\nu \geq d_{r,j} \geq 0$ with $r \in \{i, \dots, m\}$ *and* $\nu \in (0,1)$. In order to find the index *i* and the values of the weights from index *i* to index *m* we need to point out an important property of the entropic projection, that is the entropic projection preserves the relative order of components of the projected vector (i.e. if $d_{0,i} \geq d_{0,j}$ *then* $d_{t,i} \geq d_{t,j}$).

If d_0 is sorted in not increasing order, then for all $r \in \{i, \dots, m\}$ we have:

$$d_{t,r} = \xi\, d_{0,r} \; where\; \xi = \frac{1 - \nu(i-1)}{\sum_{r=i}^m d_{0,r}}$$

The following is the sorting-based algorithm in [5] that solves the entropic projection problem (2).

```
INPUT: A vector $\mathbf{d}_0 \in \mathbb{S}^m$ and a scalar $\nu \in (0,1)$
    Sort $\mathbf{d}_0$ in non-increasing order $\Rightarrow \mathbf{u}$
INITIALIZE: $Z = \sum_{r=1}^m u_r$
FOR $i = 1, \dots, m$
    $\xi = \frac{1 - \nu(i-1)}{Z}$
    IF $\xi u_i \leq \nu$
        BREAK
    ENDIF
    $Z \leftarrow Z - u_i$
ENDFOR
OUTPUT: $\mathbf{d}_t$ s.t. $d_{t,r} = \min\{\nu, \xi d_{0,r}\}$
```

7-Experiments:

In my experiments, I compared the performances of LPBoost and MLPBoost algorithms on the real sklearn dataset *breast_cancer* with 569 examples and 31 features. At each iteration, the oracle provides as weak learner the decision stump that maximizes the edge on the dataset for the given samples' distribution. The tolerance parameter chosen is $\epsilon = 0.01$ and the capping parameter $\nu = \{pm \mid p = 0.1,0.2,0.3,0.4,0.5\}$ where m is the number of samples in the dataset, is selected by means of a 5-folds cross validation.

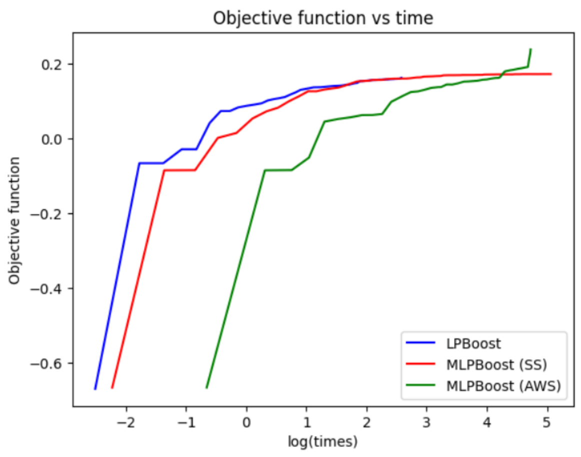
First, the dataset is divided into training and testing sets. Next, 5-fold cross-validation is performed on the training dataset, adjusting the capping parameter within the range of values to identify the optimal choice. Lastly, the algorithm is trained using the entire training set with the selected best parameter, that in this case results to be $\nu = 0.1m$ and its performance is evaluated by measuring the test error on the test dataset.

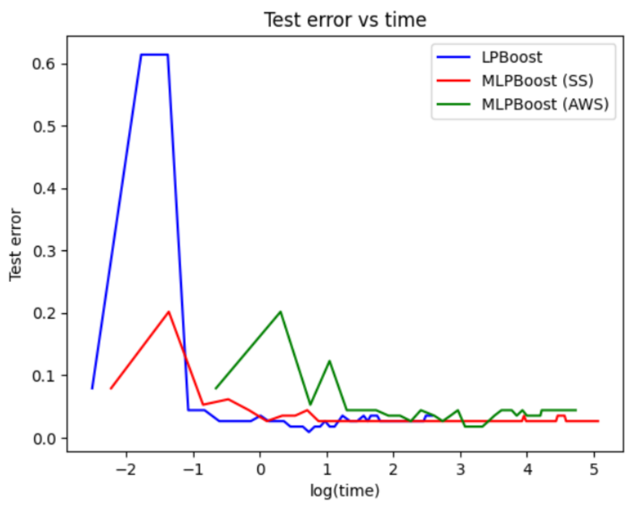
The algorithms are run for $max\_iter = 100$. Some of them, as MLPBoost with Away-Step meet the stopping conditions and achieve the convergence guarantee before the maximum number of iterations.

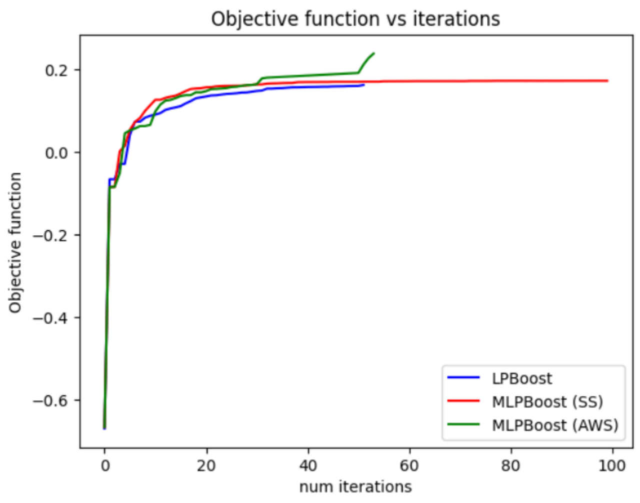
7.1-Results:

In the plots we compare the performance of LPBoost, MLPBoost with short-step strategy (SS) and MLPBoost with away-step strategy (AWS). All the graphs are plotted considering the log of the times on the x-axis to have a better visualization.

By examining the plots, it becomes evident that while LPBoost algorithm is slightly faster than MLPBoost in the maximization of the objective function, it's noteworthy that MLPBoost performs competitively. Also, when comparing test errors, MLPBoost has a similar behavior to LPBoost by achieving low test error in a short time frame. The initial very small test error points out the importance of selecting a strong base learner, in this case the one that is focused on maximizing the dataset's edge, to speed up the convergence.



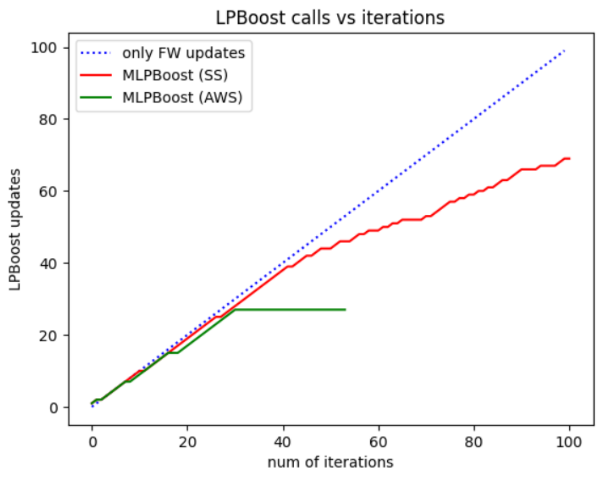




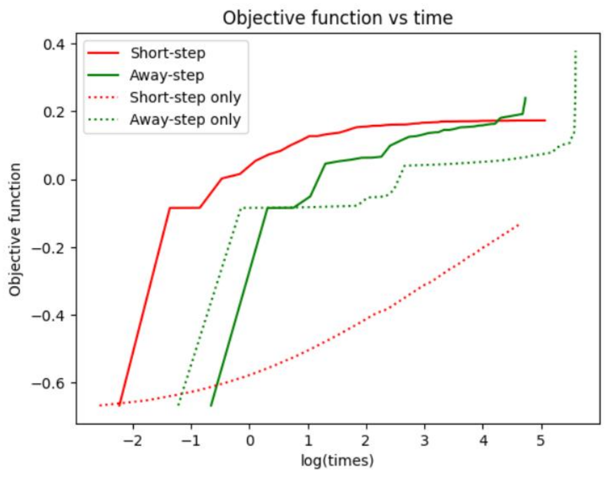
In relation to the number of iterations, MLPBoost exhibits better progress concerning the objective function, but it's important to acknowledge that each iteration of MLPBoost is more computationally intensive than one of LPBoost.

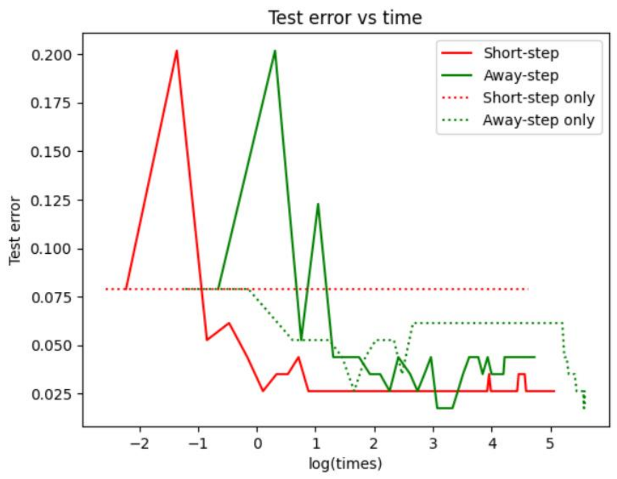
When considering MLPBoost with the Short-step strategy versus the Away-step strategy for Frank-Wolfe, the former prevails.

It's also interesting to observe how the number of updates of the second algorithm (LPBoost in this case) varies throughout the iterations. In the early iterations, LPBoost updates lead to noticeable advancements, but as the number of iterations grows FW-updates are more efficient.



Moreover, for assessing the secondary algorithm's effectiveness, a comparison can be made between MLPBoost's performance with both primary (FW) and secondary (LPBoost) updates and with only FW updates.





Remarkably, MLPBoost with LPBoost as the secondary algorithm yields superior results in terms of both maximizing the objective function and minimizing test errors.

8-Conclusion

The results are compatible with the ones obtained by Mitsuboshi et al. in [1]. While MLPBoost exhibits the same theoretical convergence rate as ERLPBoost $O\left(\frac{1}{\varepsilon^2} ln \frac{m}{\nu}\right)$, in practice it behaves as LPBoost. As illustrated by the plots, LPBoost remains the fastest algorithm. However, it's important to note that LPBoost has significant limitations, as previously mentioned. This highlights the ongoing challenge of finding an algorithm that not only exceeds LPBoost in practice but also has a good theoretical guarantee. The quest for such an algorithm remains an open problem.

References

[1] Ryotaro Mitsuboshi, Kohei Hatano, Eiji Takimoto. Boosting as Frank-Wolfe. October 3, 2022

[2] 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.

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

[4] Jonathan M. Borwein and Adrian S. Lewis. Convex Analysis, pages 65–96. Springer New York, 2006.

[5] Y. Censor and S. A. Zenios. Parallel Optimization. Oxford, New York, 1997.

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

[7] 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.

[8] 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.