Optimization for Data Science

Gradient Boosting with Frank-Wolfe

Department of Mathematics

Università degli Studi di Padova

Fairouz Baz Radwan and Sofia Pope Trogu

Supervisor: Prof. Francesco Rinaldi

June 20th, 2024

Table of Contents

1 Introduction

In this report, we will discuss and implement a boosting algorithm proposed by Mitsuboshi, et al. 2022 in "Boosting as Frank-Wolfe" that seeks to maximize the soft margin of a linear combination of hypotheses from weak learners [3]. Before introducing the boosting scheme, we would like to review the basic framework and assumptions of boosting.

In the setting where boosting is achieved through sampling, we work with a binary dataset comprising of positively or negatively labeled samples and access a weak learner capable of generating basic hypotheses, i.e. a Decision Tree Classifier. These hypotheses are generally suboptimal on their own. However, a boosting algorithm can effectively combine these hypotheses linearly, resulting in a classifier superior to any individual hypothesis [5]. This is accomplished by an algorithm that adjusts a distribution across the examples. In each iteration, the modified distribution is provided to the weak learner, which then produces a basic hypothesis with new constraints to the current sample distribution. Subsequently, the algorithm modifies the sample weights, prioritizing more challenging examples, or examples that are more commonly misclassified. The process repeats until the soft margin objective is reached or iterations have maxed out. These hypotheses are then set as the ultimate weights of the classifiers, and the final output of the algorithm is the cumulative linear combination of all the basic hypotheses generated throughout the process.

When dealing with linearly separable data, optimizing the margin has been demonstrated as an effective strategy for minimizing generalization error. Conversely, in cases where data is not linearly separable, focusing on maximizing the soft margin proves to be a more resilient approach [5]. This maximization can be directly achieved through linear programming, a method employed by LPBoost [1]. LPBoost is an effective boosting technique for solving a linear programming (LP) problem to learn an updated distribution of the training sample weights. The boosting scheme also involves generating new hypotheses that minimize the previous maximum edge. LPBoost converges to the soft margin objective rapidly, taking $\Omega(m)$ in the worst case where m are the number of examples. The algorithm's termination condition ensures that the error of the final classifier is guaranteed to be less than the accuracy parameter δ. This guarantee is reinforced by the assumption of weak-learnability, which assures us that during each iteration, the oracle provides a hypothesis with an edge of at least g, denoted as $(d_t^T A)_{j_{t+1}} \geq g$, where d_t is the current distribution, A is a matrix that combines the base hypothesis with the labels of all the samples, j is the index that indicates the current week learner, and t+1 identifies the iteration for the updated hypothesis. However, LPBoost has been shown to perform poorly in sparse conditions. To account for this, entropy regulated LPBoost (ERLPBoost) was introduced that adds a relative entropy term to the problem [6]. ERLPBoost ensures a convergence rate of $\mathcal{O}\left(\frac{1}{\varepsilon^2} \ln m/\nu\right)$, which theoretically may be more effective and efficient than LPBoost, but has shown to be slower in practical applications.

In addition to linear programming, the authors of our reference paper introduce Frank-Wolfe (FW) as a primary weight update rule in the boosting scheme [3]. The authors propose two variations of the FW algorithm: short-step and pairwise [3]. In the short-step variant of the FW algorithm, $\lambda^{(t)}$ is defined as:

$$\lambda^{(t)} := \mathrm{clip}_{[0,1]} \left(\frac{(x^{(t)} - s^{(t+1)})^\top \nabla f(x^{(t)})}{\eta_k \| s^{(t+1)} - x^{(t)} \|^2} \right)$$

The pairwise method, on the other hand, considers an away step in addition to the forward step for defining the descent direction. The away step mitigates the zig-zagging problem seen in the classic FW algorithm by defining a new away direction that goes against the forward direction for gradient descent [2]. The Pairwise Frank Wolfe (PFW) algorithm then uses this away atom and the forward atom to define a new PFW descent direction. Therefore, it is a weighted value between the worst direction and the best one.

The boosting scheme proposed by Mitsuboshi, et al. is a modified version of LPBoost, called MLPBoost that integrates the aforementioned Frank-Wolfe variants with the booster [3]. In each iteration, both a FW and Booster weight will be calculated and the better weight will be used to make the final weight update. The scheme will be fully presented in the Algorithms section below. This report will outline the algorithms we implemented, the experiments on datasets we performed discussed in the reference paper, and our results and findings.

2 Definitions and Notations

1. **soft margin optimization**: In optimization problems where the data can be perfectly separated by a linear boundary, the approach of maximizing the margin between the classes has been shown to be a reliable strategy for achieving low generalization error. However, in scenarios where the data cannot be perfectly separated by a linear boundary, a more robust choice is to maximize the concept of a "soft margin". This concept allows for some degree of misclassification or overlap between the classes, making the model more adaptable to real-world, less ideal data distributions. By focusing on the soft margin, we strike a balance between separating the classes as much as possible while allowing for some degree of classification error, which can lead to better generalization performance when faced with complex and inseparable datasets.

2. **probability simplex**: In simpler terms, for an n-dimensional probability simplex denoted as $P_v^m = \{\mathbf{d} \in [0, 1/v]^m \mid \|\mathbf{d}\|_1 = 1\}$, it is the set of all probability vectors [3].

3. **linear program**: minimization or maximization of linear problems that are constrained, where the constraints can be a linear system of equations like a set of linear inequalities.

4. **binary classification boosting**: Let $S := ((\boldsymbol{x}_i, y_i))_{i=1}^m \in (X \times \{\pm 1\})^m$ be a sequence of m examples, where X is some set. Let $H \subset [-1, +1]^X$ be a set of hypotheses. For simplicity, we assume $|H| = |\{h_1, h_2, \ldots, h_n\}| = n$. Note that our scheme can use an infinite set H. It is convenient to regard each $h_j \in H$ as a canonical basis vector $\boldsymbol{e}_j \in \mathbb{R}^n$. Let $A = (y_i h_j(\boldsymbol{x}_i))_{i,j} \in [-1, +1]^{m \times n}$ be a matrix of size $m \times n$. For a set $C \subset \mathbb{R}^n$, we denote the convex hull of C as $\mathrm{CH}(C) := \{\sum_k w_k \boldsymbol{s}_k \mid \sum_k w_k = 1, w_k \geq 0, \{\boldsymbol{s}_k\}_k \subset C\}$, where w_k are the set of weights assigned to each hypothesis [3].

5. **fenchel duality**: From the following definitions of the Fenchel problems, we are able to define the edge minimization problem and its dual problem, which is the soft margin optimization problem [3].

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

$$f^*(\boldsymbol{\theta}) = \sup_{\boldsymbol{d}\in\mathbb{R}^m} \left(\boldsymbol{d}^T\boldsymbol{\theta} - f(\boldsymbol{d})\right).$$

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 as follows:

$$\gamma = \inf_{\boldsymbol{d}} f(\boldsymbol{d}) + g(A^T\boldsymbol{d}),$$

$$\rho = \sup_{\boldsymbol{w}} -f^*(-A\boldsymbol{w}) - g^*(\boldsymbol{w}).$$

where we have the weak duality $\rho \leq \gamma$. Moreover, we note that if $0 \in \operatorname{int}(\operatorname{dom} g - A^T \operatorname{dom} f)$, then strong duality holds: $\gamma = \rho$, which can be easily proven for the edge minimization and its dual problem.

6. **edge of hypothesis**: The edge of a single hypothesis relates to the average weighted goodness of the hypothesis across all the samples. The edge of the j_{th} hypothesis can be determined by:

$$(\boldsymbol{d}^T A)_j = \textstyle\sum_{i=1}^m y_i h_j(\boldsymbol{x}_i) \cdot \boldsymbol{d}_i, \text{ where } \boldsymbol{d} \in P^m$$

In the context of boosting, when the weight of misclassified examples is increased, the negative contribution to the edge calculation (edge= 1-2*error) becomes larger if the subsequent hypotheses also misclassify these examples, thus edge decreases [5]. And transforming the edge equation, you can see that the edge is simply a conversion of the weighted error, ε_j.

$$\varepsilon_j = \tfrac{1}{2} - \tfrac{1}{2}\dot{\text{edge}}$$

7. **margin of example**: The margin of an example refers to the average goodness of the example across all possible hypotheses. The margin of the i_{th} example can be determined by:

$$\textstyle\sum_{j=1}^n y_i h_j(\boldsymbol{x}_i) \cdot \boldsymbol{w}_j, \text{ where } \boldsymbol{w} \in P^n$$

8. **edge minimization problem and its dual problem**: After choosing the hypothesis with the highest edge, the distribution should be updated so that the weights of misclassified examples increase, leading to a decrease in the edge of previous hypothesis [5]. This problem can be formulated mathematically as:

$$\min_{\boldsymbol{d}} \max_{j\in[n]} \left((\boldsymbol{d}^T A)_j + f(\boldsymbol{d})\right), \text{ where } f(\boldsymbol{d}) = \begin{cases} 0 & \text{if } \boldsymbol{d} \in P^m_\nu \\ +\infty & \text{if } \boldsymbol{d} \notin P^m_\nu \end{cases}$$

By using the Fenchel duality equations, we can define the dual problem of the above edge minimization problem

$$\min_{\boldsymbol{d}} \left(g(\boldsymbol{d}^{\boldsymbol{T}}A) + f(\boldsymbol{d}) \right), \text{ where } g : \mathbb{R}^m \to \mathbb{R} \text{ and } g(\theta) = \max_{j\in[n]} \theta_j$$

as the Soft margin maximization problem:

$$\max_{w\in P^n} -f^*(-Aw) = \max_{w\in P^n} \min_{\boldsymbol{d}\in P_v^m} \boldsymbol{d}^T Aw$$

$$f^*(-Aw) = \max_{\boldsymbol{d}} \left(-\boldsymbol{d}^T Aw - f(\boldsymbol{d}) \right) = -\min_{\boldsymbol{d}} \boldsymbol{d}^T Aw$$

with duality gap equal to zero.

9. **duality gap**: The duality gap arises when solving optimization problems with both a primal problem and its corresponding dual problem. The primal problem is the main optimization problem that you are trying to solve, and it typically involves minimizing or maximizing an objective function subject to certain constraints. The dual problem associated with the primal problem is constructed using Lagrange multipliers and involves finding the lower bound on the optimal value of the primal problem. The duality gap is the difference between the optimal values of the primal and dual problems (G = P*-D*). If the duality gap is zero (i.e., P* = D*), it indicates strong duality. However, if duality gap is strictly positive (i.e., P* > D*), it indicates weak duality. In this case, the optimal solution to the primal problem exists, but the optimal solution to the dual problem may not exist or may not be achievable. Finally, if the duality gap is negative (i.e., P* < D*), it indicates lack of a feasible solution. In the context of our project, the primal problem is edge minimization and its dual is soft margin optimization.

10. **relative entropy**: The relative entropy used in the definition of the sample distribution, $\boldsymbol{d}$ can be defined as follow:

$$\Delta(\boldsymbol{d}) = \Delta(\boldsymbol{d}, \frac{1}{m}\mathbf{1}) = \sum_{i=1}^{m} d_i \ln \left(\frac{d_i}{\frac{1}{m}} \right) = \sum_{i=1}^{m} d_i \ln d_i + \ln m, \text{since} \sum_{i=1}^{m} d_i = 1$$

Here, d_i represents the i-th component of the distribution d, and m is the dimension of the distribution, representing the number of samples in the dataset.

11. **eta**: An eta term is predefined to scale the relative entropy in the definition of the distribution in the algorithm. $\eta = 2\ln(m/\nu)/\epsilon$, where m are the number of examples, ν is the capping parameter and ϵ is the tolerance level [3].

3 Models

3.1 Distribution

The distribution $\boldsymbol{d}_t$, which represents the weights on the samples, is defined according to "Shalev-Shwartz, S., & Singer, Y. (2010)" [4] by:

$$\boldsymbol{d_t} = \arg\min_{d \in P_\nu^m} \left(\boldsymbol{d}^T A \boldsymbol{w}_t + f(d) + \beta h(\boldsymbol{d}) \right)$$

where $\beta = \frac{1}{\eta}$ and $h(\boldsymbol{d})$ is the relative entropy function, $\Delta(\boldsymbol{d})$.

We note that since we have $f(\boldsymbol{d}) = I_{d \in P_\nu^m}$ (paper uses S^m instead which is the probability simplex), we can express $\boldsymbol{d_t}$ as the gradient of the Fenchel conjugate of the function βh, where the Fenchel conjugate of the relative entropy is given by $h^*(\theta) = \log(\frac{1}{m}\sum_{k=1}^m e^{\theta_k})$:

Proof.

$$\begin{aligned} \boldsymbol{d}_t &= \nabla(\beta h)^*(\theta) \\ &= \arg\max_{d \in P_\nu^m} (\langle \theta, \boldsymbol{d} \rangle - \beta h(\boldsymbol{d})) \\ &= \arg\max_{d \in P_\nu^m} (\langle -Aw_t, \boldsymbol{d} \rangle - \beta h(\boldsymbol{d})) \\ &= \arg\min_{d \in P_\nu^m} (\langle Aw_t, \boldsymbol{d} \rangle + \beta h(\boldsymbol{d})) \end{aligned} \qquad \square$$

Furthermore, using the property $(\beta h)^*(\theta) = \beta h^*(\frac{\theta}{\beta})$ and the formula for $h^*(\theta)$ we can derive the i-th component of $\boldsymbol{d_t}$ where $\theta = -Aw_t$ as:

$$\boldsymbol{d}_{t,i} = \nabla_{\theta_i} \beta h^*(\frac{\theta}{\beta}) = \frac{\partial \beta h^*(\frac{\theta}{\beta})}{\partial \theta_i} = \frac{\beta \frac{1}{m}\frac{1}{\beta} e^{\frac{\theta_i}{\beta}}}{\frac{1}{m}\sum_{k=1}^m e^{\frac{\theta_k}{\beta}}} = \frac{e^{\frac{(-Aw_t)_i}{\beta}}}{\sum_{k=1}^m e^{\frac{(-Aw_t)_k}{\beta}}} \propto e^{-\frac{1}{\beta}(Aw_t)_i}$$

The aforementioned paper proposes a procedure for calculating the distribution $\boldsymbol{d}_t$ such that $\boldsymbol{d}_{0,i} \propto e^{\frac{(-Aw_t)_i}{\beta}}$ based on the Bregman divergence:

$$\boldsymbol{d}_t = \arg\min_{\boldsymbol{d} \in P_\nu^m : \|\boldsymbol{d}\|_\infty \le \mu} B_h(\boldsymbol{d} \| \boldsymbol{d}_0) \qquad (*)$$

Where the Bregman divergence with respect to a convex function h between two vectors $\boldsymbol{d}$ and $\boldsymbol{d}_0$ is defined as $B_h(\boldsymbol{d}\|\boldsymbol{d}_0) = h(\boldsymbol{d}) - h(\boldsymbol{d}_0) - \langle \nabla h(\boldsymbol{d}_0), \boldsymbol{d} - \boldsymbol{d}_0 \rangle$.

Briefly, the paper shows that given a non-increasing vector $\boldsymbol{d}_0$ with positive elements, the optimal solution of (*) is of the form $(\mu, \ldots, \mu, \boldsymbol{d}_{t,i}, \ldots, \boldsymbol{d}_{t,m})$, where $\boldsymbol{d}_{t,m} > 0$, and for $r \in \{i, \ldots, m\}$, we have $\boldsymbol{d}_{t,r} = \phi \boldsymbol{d}_{0,r} \le \mu$ where $\phi = \frac{1-\mu(i-1)}{\sum_{r=i}^m \boldsymbol{d}_{0,r}}$.

Algorithm 1 allows us to identify the index i by introducing a breaking condition. We start by sorting the vector $\boldsymbol{d}_0$ in non-increasing order, saving it as a new vector u and defining the sum of all its components as Z. We then iterate over the index $i \in 1, .., m$ and we define ϕ

using the above definition. Next, we check if the inequality $\phi u_i \leq \mu$ is true in order to find the position at which we stop capping the weights. If the breaking condition is not satisfied, we subtract the i_{th} component of the sorted vector from Z and move to next iteration. However, if the condition is satisfied, we know that we have reached the targeted index where we start defining $\boldsymbol{d}_{t,i}$ as ϕu_i. Finally, after breaking from the loop, we have obtained the correct value of ϕ and we define $d_{t,r}$ as $\min(\mu, \phi \boldsymbol{d}_{0,r})$.

We note that in our code, in order to avoid underflow or overflow with Z since it is taking the sum over exponential values, we used log_sum_exponential trick.

$$\log \sum_{i=1}^{n} e^{x_i} = \log(e^{x_{max}}) + \log \sum_{i=1}^{n} e^{x_i - x_{max}} = x_{max} + \log(\sum_{i=1}^{n} e^{x_i - x_{max}})$$

Expanding on that notion, we compute the logarithmic values of the initial distribution and Z so that $\log \phi = \log(1 - \mu(i-1)) - \log(Z)$.

Algorithm 1 Procedure for solving the Entropic Projection problem

```
Require: A vector $\boldsymbol{d}_0 \in S^m$ and a scalar $\mu \in (0,1)$
 1: Sort $\boldsymbol{d}_0$ in non-increasing order $\Rightarrow \boldsymbol{u}$
 2: Initialize $Z = \sum_{r=1}^{m} \boldsymbol{u}_r$
 3: for $i = 1, \ldots, m$ do
 4:     $\phi = \frac{1-\mu(i-1)}{Z}$
 5:     if $\phi u_i \leq \mu$ then
 6:         Break
 7:     end if
 8:     $Z \leftarrow Z - u_i$
 9: end for
10: Output: $\boldsymbol{d}_t$ such that $\boldsymbol{d}_{t,r} = \min\{\mu, \phi d_{0,r}\}$
```

3.2 Frank Wolfe Variants

The FW algorithm, also known as the Conditional Gradient algorithm, is a first-order optimization technique used to solve convex optimization problems [2]. It aims to solve problems of the form: $\min_{x \in C} f(x)$ where C is a closed convex set and f is a smooth and convex function. The original algorithm seeks to find the optimal solution by iteratively moving in the direction of the negative gradient of the objective function, subject to certain constraints. It starts with an initial feasible solution, denoted as $x^{(0)}$. Then, for each iteration, it computes the gradient of the objective function at the current point: $\nabla f(x^{(t)})$ and finds a new extreme point, $s^{(t+1)} \in \arg\min_{s \in C} \langle s, \nabla f(x^{(t)}) \rangle$. Finally, it performs a line search or step size selection to find the optimal step size. This step size is denoted as $\lambda^{(t)}$. Then, it updates the iterate using the chosen step size: $x^{(t+1)} = x^{(t)} + \lambda^{(t)}(s^{(t+1)} - x^{(t)})$.

3.2.1 Short-step Rule

The classical approach in the Frank-Wolfe optimization algorithm involves using a specific formula to determine the step size at each iteration. This step size is used to move towards the solution incrementally. Traditionally, the step size is set as $\lambda_t = \frac{2}{t+2}$, where t is the iteration number. This choice ensures that the step size decreases over time, allowing for finer adjustments as the algorithm progresses.
One alternative approach is the short-step strategy that calculates the step-size λ_t as:

$$\lambda_t = \text{clip}_{[0,1]} \frac{\boldsymbol{d}_t^T A(\boldsymbol{e}_{j+1} - \boldsymbol{w}_t)}{\eta \left\| A(\boldsymbol{e}_{j+1} - \boldsymbol{w}_t) \right\|_\infty^2}$$

The function $\text{clip}_{[0,1]}(x) = \max\{0, \min\{1, x\}\}$ ensures that x is bounded between 0 and 1.

3.2.2 Pairwise Rule

Another variant of the Frank-Wolfe algorithm that we implemented is the Pairwise Update rule, characterized by its use of a combination of forward and away directions in the weight update process. This approach navigates the weight space by leveraging the difference between the forward direction vector $\boldsymbol{e}_{j_{t+1}}$ and the away direction vector $\boldsymbol{e}_{\text{Away}}$ (away basis corresponding to the hypothesis having the smallest edge amongst the first t hypotheses), added to the current weight vector $\boldsymbol{w}_t$, to determine the update direction. After computing the away basis, a line search method is implemented to find the best lambda or step size that minimizes the objective function:

$$\lambda_t \leftarrow \arg \min_{\lambda \in [0, \lambda_{t,\max}]} \tilde{f}^*(-A(\boldsymbol{w}_t + \lambda(\boldsymbol{e}_{j_{t+1}} - \boldsymbol{e}_{\text{Away}})))$$

The fenchel conjugate of the f tilde function, $\tilde{f}^*$ can be expressed as the following:

$$\tilde{f}^*(-A(\boldsymbol{w}_t + \lambda(\boldsymbol{e}_{j_{t+1}} - \boldsymbol{e}_{\text{Away}}))) = \max_{d \in P_v^m} [\boldsymbol{d}^T(-A(\boldsymbol{w}_t + \lambda(\boldsymbol{e}_{j_{t+1}} - \boldsymbol{e}_{\text{Away}})) - \tilde{f}(\boldsymbol{d})]$$

$$= \max_d [\boldsymbol{d}^T(-A\boldsymbol{w}_t) - \lambda \boldsymbol{d}^T A(\boldsymbol{e}_{j_{t+1}} - \boldsymbol{e}_{\text{Away}}) - \tilde{f}(\boldsymbol{d})]$$

$$= -\min_d [\boldsymbol{d}^T(A\boldsymbol{w}_t) + \tilde{f}(\boldsymbol{d}) + \lambda \boldsymbol{d}^T A(\boldsymbol{e}_{\boldsymbol{j_{t+1}}} - \boldsymbol{e}_{\text{Away}})]$$

In our code, a bisection method is utilized for line search to efficiently find λ_t that minimizes the objective function within the constraints of the problem. The bisection method applies a systematic approach to narrowing the interval $[0, \lambda_{t,\max}]$ by iteratively bisecting it and evaluating the objective function $\tilde{f}^*$ at the interval's midpoint. Based on these evaluations, the method determines the sub-interval more likely to contain the optimal λ_t and continues this process until the range of λ_t is sufficiently small. This ensures that the chosen λ_t is an accurate approximation of the true minimum of $\tilde{f}^*$, leading to an effective and methodical weight update process in the Pairwise Frank-Wolfe variant.

3.3 LPBoost

LPBoost, or Linear Programming Boost, is a boosting algorithm designed to construct a strong classifier by linearly combining a set of weak classifiers. The algorithm is formulated as a linear program that aims to minimize a linear objective function subject to a set of linear constraints. In our study, we are utilizing LPBoost to solve the following edge minimization problem:

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

or its dual problem, the soft margin optimization problem that we have previously defined as:

$$\max_{\boldsymbol{w} \in P^n} -f^*(-A\boldsymbol{w}) = \max_{\boldsymbol{w} \in P^n} \min_{\boldsymbol{d} \in P_v^m} \boldsymbol{d}^T A \boldsymbol{w}$$

This algorithm essentially tries to maximize the objective function $-f^*$ which is computed by assigning more weights d to smaller margins in order to minimize the inner product between $\boldsymbol{d}^T$ and Aw as presented in Algorithm 5.

Moreover, a more stabilized version of the LPBoost, is the **ERLPboost** which introduces a regularization term in the distribution domain and updates $\boldsymbol{d}$ by solving the problem:

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

where $\Delta(\boldsymbol{d})$ is the relative entropy from the uniform distribution $\frac{1}{m}\mathbf{1} \in P_\nu^m$.

On the other hand, **C-ERLPBoost** (Corrective Edge Regularized Linear Programming Boost), is an enhanced version of the ERLPBoost that significantly improves computational efficiency by maintaining a weight vector $\boldsymbol{w}_t \in \mathcal{P}^n$ that only contains non-zero values for a subset of past hypotheses, specifically $\{h_{j1}, h_{j2}, \ldots, h_{jt}\} \subset H$. This selective focus on a subset of hypotheses contributes to much faster computations per iteration compared to LPBoost.

3.3.1 MLPBoost

In the referenced paper by Mitsuboshi et al. [3], a new boosting algorithm inspired by the LPBoost and Frank-Wolfe (FW) framework is introduced. The **MLPBoost** algorithm combines two update mechanisms for the weak-learner weights' w: a primary FW update (denoted as F) which uses either a pairwise or shortstep update rule, and a secondary LP update (B) as shown in lines 11 and 12 in Algorithm 2. Both updates are designed to generate a weight vector for base learners. Then, in line 13 of the algorithm, the algorithm determines the most suitable weight vector for the update by assessing which weight vector optimally minimizes the objective function $\tilde{f}^*(-A\boldsymbol{w}_t)$.

Moreover, the MLPBoost algorithm's workflow involves the computation of the distribution $\boldsymbol{d}_t$ at each iteration in line 5 of Algorithm 2. This computation employs a sorting-based algorithm, as shown in section 3.1.

In line 7, the algorithm establishes a stopping condition based on the optimality gap, denoted by $\epsilon_t = \min_{0\leq\tau\leq t}(\boldsymbol{d}^T A)_{j_{\tau+1}} + \tilde{f}^*(-A\boldsymbol{w}_t)$. Utilizing the assumption that our weak learner returns a hypothesis $h_{j_{t+1}}$ having an edge of at least g and that $\epsilon_t \leq \frac{\epsilon}{2}$, we can verify the stopping condition used in our algorithm:

$$g + \tilde{f}^*(-A\boldsymbol{w}_t) \leq \epsilon_t \leq \frac{\epsilon}{2} \implies -\tilde{f}^*(-A\boldsymbol{w}_t) \geq g - \frac{\epsilon}{2} \geq g - \epsilon$$

and by Lemma 1 in Mitsuboshi et al. [3], this implies that $-f^*(-A\boldsymbol{w}_t) \geq g - \epsilon$.

3.3.2 Instances of Frank Wolfe

We have established that ERLPBoost aims to solve the $\min_d \max_{j\in[n]}(\boldsymbol{d}^T A)_j + \tilde{f}^*(\boldsymbol{d})$ or alternatively

$$\max_{\boldsymbol{w}\in P^n} -\tilde{f}^*(-A\mathbf{w}) = -\min_{\boldsymbol{w}\in P^n} \tilde{f}^*(-A\mathbf{w}) = -\min_{\theta\in -AP^n} \tilde{f}^*(\theta)$$

where $\tilde{f} = f + \frac{1}{\eta}\Delta$. Given that $\frac{1}{\eta}\Delta$ is a $\frac{1}{\eta}$-strongly convex function with respect to the l_1-norm, $\tilde{f}$ also inherits this property. Since $\tilde{f}^*$ is in turn an η-smooth function with respect to the l_∞-norm, we can perceive our soft margin optimization problem as a minimization problem of a smooth function. And so, our algorithm updates the distribution by minimizing $d^T A\mathbf{w} + \tilde{f}(\boldsymbol{d})$ which is equivalent to the computation of the gradient of the objective function $\tilde{f}^*(\theta)$, where $\theta = -A\boldsymbol{w}$.

$$\nabla\tilde{f}^*(-A\mathbf{w}) = \arg\max_{\boldsymbol{d}\in\mathbb{P}^m_\nu}(-\boldsymbol{d}^T A\mathbf{w} - \tilde{f}(\boldsymbol{d})) = \arg\min_{\boldsymbol{d}\in\mathbb{P}^m_\nu}(\boldsymbol{d}^T A\mathbf{w} + \tilde{f}(\boldsymbol{d}))$$

It then obtains a basis vector $\boldsymbol{e}_{j_{t+1}}$ for the corresponding new hypothesis which aims to maximize the edge:

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

Therefore, finding said hypothesis corresponds to solving a linear program in FW algorithm and the LPBoost algorithms can all be perceived as instances of the Frank Wolfe algorithm.

3.4 Algorithms

Algorithm 2 Modified LPBoost (MLPBoost)

```
Require: Training examples $S = ((x_i, y_i))_{i=1}^m \in (X \times \{\pm 1\})^m$, a hypothesis set $H \subset [-1,+1]^X$, a FW algorithm $F$, a secondary algorithm $B$, and parameters $\nu > 0$ and $\epsilon > 0$. Predefine FW algorithm as either short-step or pairwise. Secondary algorithm will always be LPBoost.
 1: Set $A = (y_i h_j(\boldsymbol{x}_i))_{i,j} \in [-1,+1]^{m \times n}$.
 2: Send $d_0 = \frac{1}{m}\mathbf{1}$ to the weak learner and obtain a hypothesis $h_{j_1} \in H$.
 3: Set $\boldsymbol{w}_{\mathbf{1}} = \boldsymbol{e}_{j_1}$.
 4: for $t = 1, 2, \ldots, T$ do
 5:     Compute the distribution $\boldsymbol{d_t} = \nabla \tilde{f}^*(-A\boldsymbol{w}_t) = \arg\min_{d \in P_m^\nu} \left(\boldsymbol{d}^T A \boldsymbol{w}_t + \frac{1}{\eta}\Delta(\boldsymbol{d})\right)$.
 6:     Obtain a hypothesis $h_{j(t+1)} \in H$ and the corresponding basis vector $e_{j_{(t+1)}} \in P^n$.
 7:     Set $\epsilon_t = \min_{0 \le \tau \le t}(\boldsymbol{d}_\tau^T A)_{j_{(\tau+1)}} + \tilde{f}^*(-Aw_t)$ and let $\varepsilon_{t+1} = \{\boldsymbol{e}_{j_\tau}\}_{\tau=1}^{t+1}$.
 8:     if $\epsilon_t \le \frac{\epsilon}{2}$ then
 9:         Set $T = t$ and break.
10:     end if
11:     Compute the FW weight $\boldsymbol{w}_{t+1}^{(1)} = F(A, \boldsymbol{w}_t, \boldsymbol{e}_{j_{(t+1)}}, \varepsilon_t, \boldsymbol{d}_t)$.
12:     Compute the secondary weight $\boldsymbol{w}_{t+1}^{(2)} = B(A, \varepsilon_{t+1})$.
13:     Update the weight $\boldsymbol{w}_{t+1} \leftarrow \arg\min_{\boldsymbol{w}_{(t+1)}^{(k)}, k \in \{1,2\}} \tilde{f}^*(-A(\boldsymbol{w}_{t+1}^{(k)}))$.
14: end for
Ensure: Combined classifier $H_T = \sum_{t=1}^T \boldsymbol{w}_{T,t} h_t$.
```

Algorithm 3 Boosting Pairwise Frank-Wolfe algorithm

```
Require: Apply same requirements as Algorithm 2.
  Follow steps 1-3 from Algorithm 2 to compute A, $d_0$ and $w_1$
  for $t = 1$ to $T$ do
      Follow steps 5-10 from Algorithm 1 to compute $\boldsymbol{d}_t$, $\boldsymbol{e}_{j+1}$, $\epsilon_t$, $\varepsilon_{t+1}$, and check for breaking conditions
      Let $\boldsymbol{w}_t = \sum_{\boldsymbol{e} \in E_t} \alpha_{t,\boldsymbol{e}} \boldsymbol{e}$ be the current representation of $\boldsymbol{w}_t$ with respect to the basis vectors $E_t \subset \varepsilon_t$ with positive coefficients $\{\alpha_{t,e}\}_{e \in E_t}$.
      Compute an away basis $\boldsymbol{e}^{\text{Away}} \in \arg\min_{\boldsymbol{e} \in E_t} \boldsymbol{d}_t^T A \boldsymbol{e}$ and set $\lambda_{t,\max} = \alpha_{t,\boldsymbol{e}^{\text{Away}}}$.
      Line-search: $\lambda_t \leftarrow \arg\min_{\lambda \in [0,\lambda_{t,\max}]} \tilde{f}^*(-A(\boldsymbol{w}_t + \lambda(\boldsymbol{e}_{j_{t+1}} - \boldsymbol{e}_{\text{Away}})))$.
      Update weights: $\boldsymbol{w}_{t+1}^{(1)} = \boldsymbol{w}_t + \lambda_t(\boldsymbol{e}_{j_{t+1}} - \boldsymbol{e}_{\text{Away}})$
  end for
```

Algorithm 4 Short-Step Rule for Boosting FW

```
  Update weights: $\boldsymbol{w}_{t+1}^{(1)} = \boldsymbol{w}_t + \lambda_t(\boldsymbol{e}_{\boldsymbol{j_{t+1}}} - \boldsymbol{w}_t)$ , where $\lambda_t = \text{clip}_{[0,1]} \frac{\boldsymbol{d}_t^T A(\boldsymbol{e}_{j+1} - \boldsymbol{w}_t)}{\eta \|A(\boldsymbol{e}_{j+1} - \boldsymbol{w}_t)\|_\infty^2}$
```

Algorithm 5 LPBoost algorithm

```
1: Given as input training set: $S$
2: $\boldsymbol{n} \leftarrow 0$                                        ▷ No weak hypotheses
3: $\boldsymbol{w} \leftarrow 0$                                        ▷ All coefficients are 0
4: $\beta \leftarrow 0$
5: $\boldsymbol{d} \leftarrow (\frac{1}{l}, \dots, \frac{1}{l})$        ▷ Corresponding optimal dual
6: repeat
7:     $n \leftarrow n + 1$
8:     Find weak hypothesis: $h_n \leftarrow H(S, d)$
9:     Check for optimal solution:
10:    if $\sum_{i=1}^{m} d_i y_i h_n(x_i) \leq \beta$ then
11:        $n \leftarrow n - 1$
12:        break
13:    end if
14:    $H_{in} \leftarrow h_n(x_i)$
15:    Solve restricted master for new costs:
16:    arg min    $\beta$
17:         s.t.  $\sum_{i=1}^{m} d_i y_i h_{ij}(x_i) \leq \beta$
18:    $(d, \beta) \leftarrow$  $j = 1, \dots, n$
19:               $\sum_{i=1}^{n} d_i = 1$
20:               $0 \leq d_i \leq D, i = 1, \dots, n$
21: until
22: $\boldsymbol{w} \leftarrow \arg\max_{\boldsymbol{w} \in \mathrm{CH}(\varepsilon_{t+1})} \min_{\boldsymbol{d} \in P_\nu^m} \boldsymbol{d}^T A \boldsymbol{w}$ ▷ w are the Lagrangian multipliers from last LP
23: return $n, f = \sum_{j=1}^{n} w_j h_j$
```

Algorithm 6 LPBoost rule for Boosting FW

```
1: Update weights: $\boldsymbol{w}_{t+1}^{(2)} \leftarrow \arg\max_{\boldsymbol{w} \in \mathrm{CH}(\varepsilon_{t+1})} \min_{\boldsymbol{d} \in P_\nu^m} \boldsymbol{d}^T A \boldsymbol{w}$
```

4 Experiments

4.1 Datasets

The algorithms we developed (LPBoost and MLPBoost) were tested and compared on two datasets: Breast Cancer and Thryoid. The Breast Cancer dataset comes from the sklearn package that can be directly loaded into python (https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html). The Thryoid dataset, on the other hand, comes from Gunnar Rätsch's benchmark dataset and was retrieved from this location: http://theoval.cmp.uea.ac.uk/~gcc/matlab/default.html#benchmarks. Each dataset contains X explanatory features and one target variable equal to ± 1 to use in our binary classification optimization problem.

4.2 Test/Train Split

Prior to feeding our data to our model, we applied the train/test split which is a common ML technique that involves splitting the original data set into two subsets: the training set and the test set. While the Thyroid dataset was already pre-split 75:25 from the source, we had to manually split the Breast Cancer dataset. The training set was made up of the majority of the data set (80%) and is used for training, where the model learns the best parameters, without playing any part in the evaluation phase. On the other hand, the test set (25%) is exclusively used for testing our model's performance in predicting the target variable.

Table 1: Dataset structures for train-test split

Dataset	**Train Shape**	**Test Shape**
B. Cancer	(455, 31)	(114, 31)
Thyroid	(215, 6)	(75, 6)

4.3 Parameters and Cross Validation

The reference paper specified the parameter settings to use [3]. We set the capping parameters $\nu \in \mathbb{N} := \{p \cdot m \,|\, p = 0.1, 0.2, \dots, 0.5\}$ and the tolerance parameter $\epsilon = 0.01$, where m is the number of training instances. We use a custom Decision Tree Classifier as the weak learner that returns the best decision tree based on the max edge and given sample weights. We set $\eta = 2 \ln(m/\nu)/\epsilon$. After the train-test split, we performed a 5-fold cross validation over the training set and computed the test errors to choose the best capping parameter ν. We used that value of ν to train our algorithm on the full training set and measured test error on test set. We set a maximum number of iterations to 100, even though the algorithm often breaks out of the loop before the max_iter condition is met.

4.4 Testing Criteria

In addition to the final accuracy calculation on the test set using the best ν, we measured the System time. We then calculated the average computation time of convergence, i.e. the algorithm terminates, over the capping parameters over N. Once the best capping parameter was determined for each dataset, we measured test error and soft margin objective convergence over time, as well as the number of LPBoost weight updates over the iterations. We counted LPBoost weight updates when the LP weight update was chosen over the Frank-Wolfe strategy update in each iteration.

4.5 Results

In the tables and plots below, we compare the performance among our algorithms, LPBoost, MLPBoost with Short-Step, and MLPBoost with Pairwise FW.

4.5.1 Computation Time and Test Error over ν

These tables compare the performance of our implemented algorithms across the two datasets over the tuning parameter ν. Table 2 demonstrates that LPBoost terminates or reaches the maximum iterations faster than MLPBoost for both Short Step (SS) and the Pairwise update rule (PFW) on average for both datasets. Between the two variations of MLPBoost, MLPB (PFW) exhibits a faster computation time than MLPB (SS). This aligns with what the authors of the original paper found.

The final test error was recorded from the final iteration of each algorithm per dataset in Table 3. The algorithms reached an error rate comparable and sometimes better than the benchmarks from the reference paper. While the reference paper lacks results for the MLPB (PFW only) and MLPB (SS only), the results on our datasets align with expectations, particularly with regard to MLPB (SS only) exhibiting the highest error. We expect that the short-step algorithm on its own may take longer to converge with its conservative approach, so would likely require more iterations to converge.

Table 2: Comparison of the computation time (seconds). Each cell is the average computation time over the capping parameters over N.

Dataset	**LPBoost**	**MLPB (SS)**	**MLPB (PFW)**
B.Cancer	4.82	108.27	78.67
Thyroid	0.57	65.0	39.38

Table 3: Final test errors for 0th fold of each dataset for the best parameters.

Dataset	**LPBoost**	**MLPB (PFW only)**	**MLPB (SS only)**	**MLPB (SS)**	**MLPB (PFW)**
B.Cancer	0.03	0.03	0.08	0.03	0.03
Thyroid	0.08	0.05	0.22	0.01	0.01

4.5.2 Plots with best ν

The plots below measure various performance metrics over time and iterations after determining the best parameters for each dataset. Figure 1 illustrates the fast convergence to the soft-margin objective particularly for LPBoost. While initialized at a lower objective value, MLPBoost for both Short-Step and Pairwise catch up fairly quickly to LPBoost. MLPBoost (SS) converges faster than MLPBoost (PW), which we can likely attribute to the algorithm choosing a LP update more often than the short-step FW weight update. This is particularly evident in the plot for thyroid, a smaller and sparser dataset. Figure 2 shows the erratic behavior of test error over time for the two respective datasets, especially at the beginning of the algorithm. We observe a high initial test error, but one which converges to the MLPBoost algorithm with time, for the LPBoost algorithm compared to the other two algorithms for the Breast Cancer dataset. This behavior aligns with the reference paper. Figure 3 narrows in on the MLPBoost algorithm, particularly on the role of LP Updates over the course of iterations. In the Breast Cancer dataset, it appears that the algorithm initially exercises a steady trend of LP weight updates, but employs the FW updates after awhile. The thyroid dataset, on the other hand, being less well-conditioned, immediately switches to FW updates. From the reference paper, we expected this– that the weight updates to taper off over time as the FW weights become more effective. Finally, we can observe the effectiveness of the modified algorithm in Figure 4 by comparing the convergence of the soft margin objective between the MLPBoost with Short-Step or Pairwise update with their Frank-Wolfe only counterparts. The improvement in performance can be clearly seen between MLPB (SS) and (SS only), where we see MLPB (SS) showing almost immediate improvement and MLPB (SS only) still slowly moving toward the function's objective. In the case of pairwise, we detect a significant improvement after some time with the MLPB (PW) compared to the PW only version, indicating that the pairwise method, even on its own) is an effective method for efficient convergence.

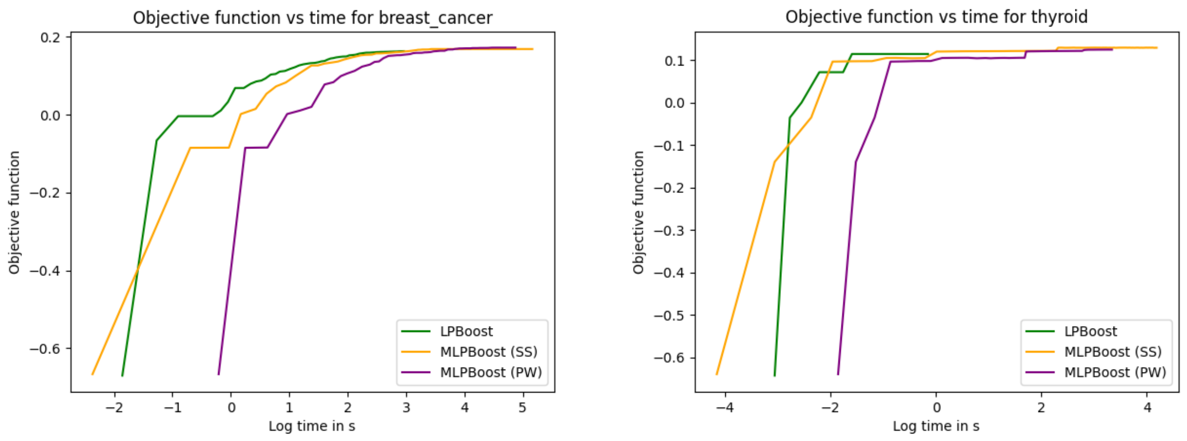

Figure 1: Objective function with respect to time in log scale for Breast cancer and Thyroid datasets

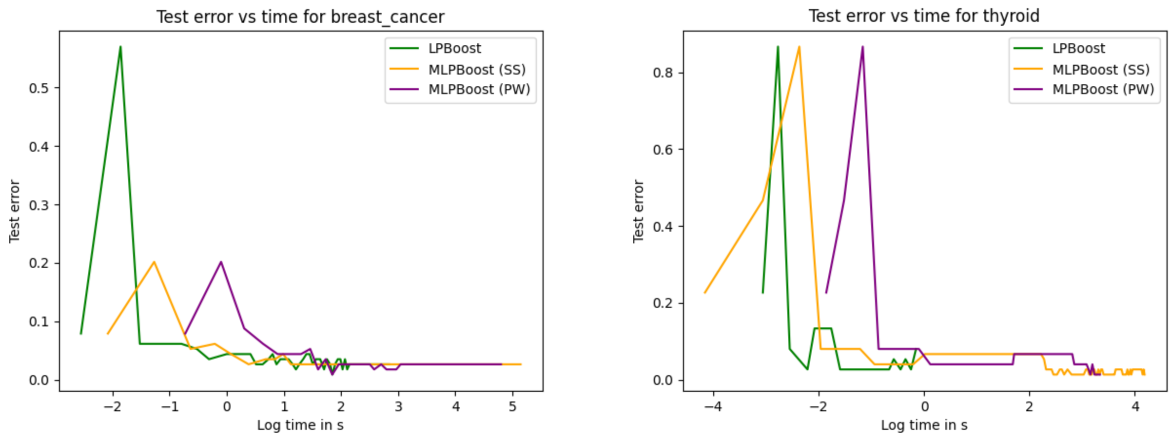

Figure 2: Test Error with respect to time in log scale for Breast Cancer and Thyroid datasets

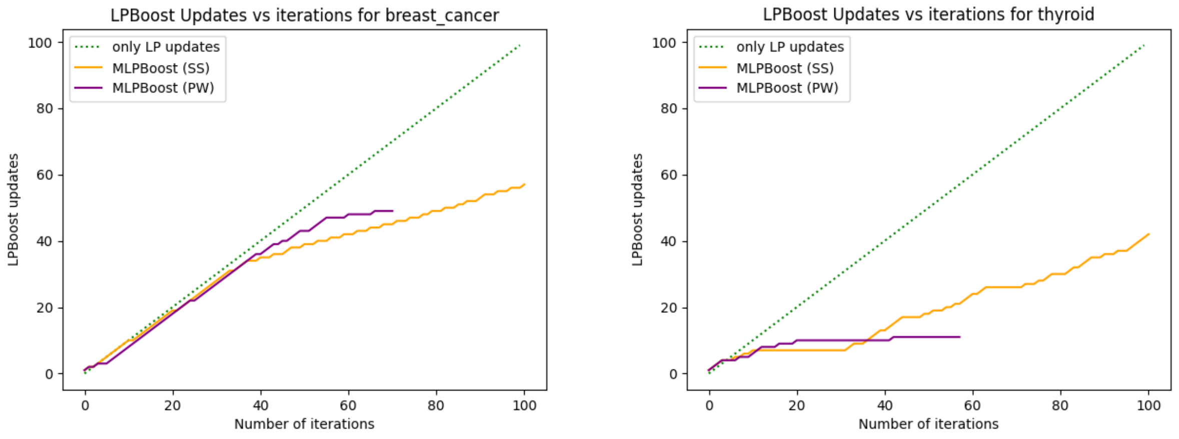

Figure 3: Number of LPboost Updates done in MLPBoot Algorithm with respect to iterations

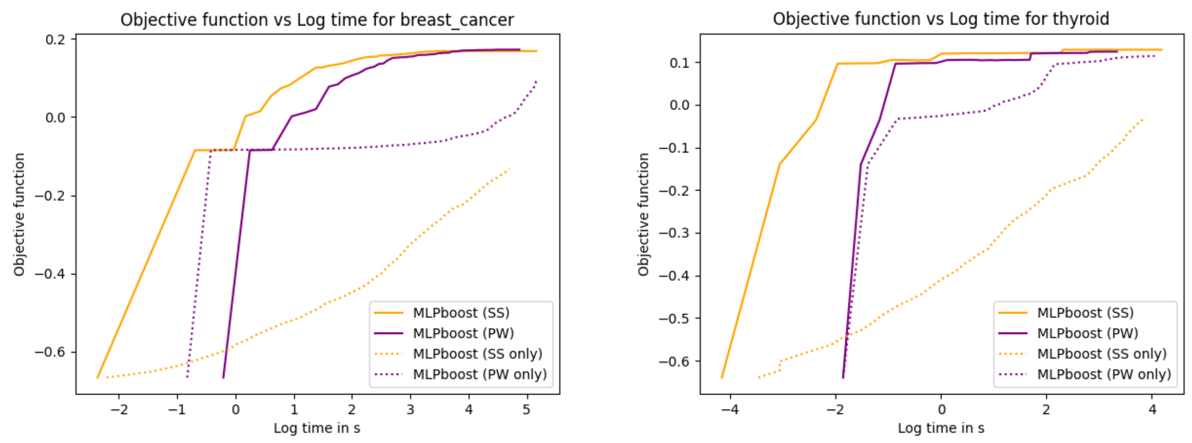

Figure 4: Comparing the objective functions over time in log scale between MLPBoost with and without secondary update rule.

5 Conclusion

Following the research done in "Boosting as Frank-Wolfe" [3], we aimed to study the performance of boosting schemes in the context of Frank-Wolfe for solving a Soft-Margin optimization problem. Furthermore, we compared the results obtained on Modified LPBoost which incorporated two updates rules by implementing our algorithms on two real data sets. As predicted, LPBoost continues to be the fastest algorithm. However, the proposed MLPBoost may provide a competitive alternative that converges to the optimization objective for data with varying sparsity and complexity.

6 References

[1] Demiriz, A., Bennett, K. P., & Shawe-Taylor, J. (2002). Linear Programming Boosting via Column Generation. *Machine Learning, 46*(2-3), 225–254.

[2] Lacoste-Julien, S., & Jaggi, M. (2015). On the Global Linear Convergence of Frank-Wolfe Optimization Variants. In *Advances in Neural Information Processing Systems, 28*, 496–504.

[3] Mitsuboshi, R., Hatano, K., & Takimoto, E. (2022). Boosting as Frank-Wolfe, 1–16. Retrieved from http://arxiv.org/abs/2209.10831

[4] Shalev-Shwartz, S., & Singer, Y. (2010). On the Equivalence of Weak Learnability and Linear Separability: New Relaxations and Efficient Boosting Algorithms. *Machine Learning, 80*(2-3), 197-223.

[5] Warmuth, M. K., Glocer, K., & Rätsch, G. (2007). Boosting Algorithms for Maximizing the Soft Margin. In *Advances in Neural Information Processing Systems 20 - Proceedings of the 2007 Conference*, 1585-1592.

[6] Warmuth, M. K., Glocer, K. A., & Vishwanathan, S. V. N. (2008). Entropy Regularized LPBoost. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5254 LNAI*, 256–271. https://doi.org/10.1007/978-3-540-87987-9_23

[7] Warmuth, M. K., & Vishwanathan, S. (2009). Survey of Boosting from an Optimization Perspective [Video]. VideoLectures.NET. https://videolectures.net/icml09_warmuth_vishwanathan_sbop/