Optimization for Data Science - Final project
Study of Boosting as Frank-Wolfe

Maria Aba
maria.aba@studenti.unipd.it

Yosr Ben Taher
yosr.bentaher@studenti.unipd.it

Matteo Posenato
matteo.posenato@studenti.unipd.it

1. Introduction

Boosting is an ensemble learning technique that aims at improving the performance of weak or base model typically decision trees by combining them into a single robust predictive model[**?**]. It is used for the purpose of reducing the bias and variance and so enhancing the generalization and the model's overall performance and robustness. It can handle complex datasets with noise and outliers, making it suitable for a wide range of real-world applications, including classification and regression problems.

Some boosting algorithms, such as LPBoost (Linear Programming Boosting), ERLPBoost (Exponential Regularized Linear Programming Boosting), and C-ERLPBoost (Cost-sensitive Exponential Regularized Linear Programming Boosting) aim at solving soft-margin optimization problems, they focus on improving the margin while allowing for some classification errors, making them suitable for problems where strict classification boundaries might not be desirable. The latter algorithms suffer either from a high number of iteration O(M) or a high number of computations.

The paper "Boosting as Frank-Wolfe"[**?**] aims at solving this problem by introducing another generic boosting algorithm that iterates the weight updates between LPBoost and a Frank-Wolfe algorithm, achieving the same convergence of ERLPBoost and C-ERLBoost.

2. Boosting

2.1. General Boosting

By integrating a number of weak learners into one strong learner to reduce training errors, boosting increases the predicted accuracy and performance of machine learning models. In boosting, a random sample of data is chosen, fitted with a model, and then trained sequentially, which means that each model attempts to compensate for the weaknesses of its predecessor. The weak rules from each classifier are combined during each iteration to create a single, strong prediction rule.

Weak Learner refers to simple models that have low prediction accuracy that is slightly better than random guessing. They are prone to overfitting, meaning that they are unable to classify data that varies too much from their original dataset.

The training method in boosting varies depending on the type of boosting process called the boosting algorithm. However, an algorithm takes the following general steps to train the boosting model. To begin, the boosting algorithm assigns equal weight to each data sample. It feeds the data to the first machine model, called the base algorithm and this later makes predictions for each data sample. Then, the algorithm assesses model predictions and increases the weight of samples with a more significant error but also assigns a weight based on model performance. Then, the weighted data are passed to the next decision tree. This process continues until a predetermined stopping condition is met, such as until a set number of weak learners have been created, or the model's performance has plateaued.[**?**]

Algorithm1. (General Boosting)

Given training set $(x_1, y_1), ...(x_m, y_m)$
$y_i \in \{-1, +1\}$ is the correct label of instance $x_i \in X$
For $t = 1...T$
    Construct a distribution D_t on $\{1, ...m\}$
    Find weak hypothesis ("rule of thumb")
        $h_t : X \to \{-1, +1\}$
    with small error ϵ_t on D_t
        $\epsilon_t = Pr_D[h_t(x_i) \neg = y_i]$
Output: final hypothesis H_{final}

Boosting has easy-to-understand and easy-to-interpret algorithms that learn from their mistakes. These algorithms don't require any data reprocessing, and they have built-in routines to handle missing data. Boosting algorithms combine multiple weak learners in a sequential method, which iteratively improves observations. This approach helps to

reduce high bias that is common in machine learning models. They also prioritize features that increase predictive accuracy during training as well as to reduce data attributes and handle large datasets efficiently.

However, boosting models are vulnerable to outliers or data values that are different from the rest of the dataset. Because each model attempts to correct the faults of its predecessor, outliers can skew results significantly. They can also increase computational efficiency, since they only select features that increase the predictive power during training.

2.2. LPBoost, ERLPBoost, C-ERLPBoost

LPBoost(Linear Programming Boosting) is a boosting algorithm designed to solve both binary classification problems and regression.. The latter differs from the regular boosting algorithm by its flexibility to consider different loss functions to determine the error. The most commonly used functions are the L1 (Absolute Loss) or L2 (Squared Error). One of its properties is its robustness against the noise in the data, which the regular boosting can easily be affected by, which leads to an overfitted model. Additionally, it adapts well to imbalance datasets.Its convergence depends consequently on the used function.However, it often converges faster because it optimizes a continuous loss function directly [**?**].

ERLPBoost(Exponential Regularized Linear Programming Boosting) is an extension of LPBoost where a regularized term is added to the loss function in order to prevent overfitting by penalizing overly complex models,consequently, leading to less an improved generalization. The regularization term can aid in achieving smoother convergence.

C-ERLPBoost(Cost-sensitive Exponential Regularized Linear Programming Boosting) is also an extension of both algorithms, it has both a regularization term and constraints which can enforce some conditions on the classifier.The added constraints can insure certain requirements, fairness and interpretability of the model. The last two variants are known to have a smoother convergence rate, more control on the classifier. However, they require more computational power compared to the LPBoost due to their higher complexity.

	LPBoost	C-ERLPBoost	ERLPBoost
Iter.bound	$\Omega(m)$	$\mathrm{O}\left(\frac{1}{\epsilon^2 \ln\left(\frac{m}{v}\right)}\right)$	$\mathrm{O}\left(\frac{1}{\epsilon^2 \ln\left(\frac{m}{v}\right)}\right)$

Table 1. Iteration bounds comparision

3. Algorithms

3.1. LPBoost

Linear Programming boosting as defined in the previous section, is a more flexible boosting method that incorporates a loss function to measure the error[**?**]. The LPBoost algorithm consists of the following steps:

Algorithm2. (LPBoost)

Given as input training set: S
$n \leftarrow 0$ *No weak hypotheses*
$a \leftarrow 0$ *All coefficients are* 0
$\beta \leftarrow 0$
$u \leftarrow (\frac{1}{m}, \ldots, \frac{1}{m})$ *Corresponding optimal dual*
REPEAT
$n \leftarrow n + 1$
Find weak hypothesis using equation (10) :
$h_n \leftarrow \mathcal{H}(S, u)$
Check for optimal solution:
If $\sum_{i=1}^{m} u_i y_i h_n(x_i) \leq \beta, n \leftarrow n - 1,$ *break*
$H_{in} \leftarrow h_n(x_i)$
Solve restricted master for new costs:

$$(u, \beta) \leftarrow \begin{array}{ll} \textit{argmin} & \beta \\ \textit{s.t.} & \sum_{i=1}^{m} u_i y_i h_j(x_i) \leq \beta \\ & j = 1, \ldots, n \\ & 0 \leq u_i \leq D, \ i = 1, \ldots, m \end{array}$$

END
$a \leftarrow$ *Lagrangian multipliers from last LP*
return $n, f = \sum_{j=1}^{n} a_j h_j$

In the paper[**?**], the LPBoost was implemented as the secondary weight w2.

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

3.2. Frank-Wolfe

The Frank-Wolfe algorithm (also known as conditional gradient) is an optimization technique introduced for constrained convex optimization, However its applications on nonconvex functions have begun to be studied only a few years ago. It is particularly well suited for large and It can handle problems with sparse gradients efficiently, which is useful in many real-world applications. Frank wolfe is a simple projection free algorithm relying on a linear minimization oracle over the convex set C.

One of the inconveniences of the standard FW is the ZigZag phenomenon which is observed in non-convex problems, when the optimal solution lies on the boundary, the algorithm may repeatedly switch between different directions, causing oscillations in the solution trajectory caus-

ing a sublinear convergence rate. To solve this problem, other variants of the algorithm were introduced. The article considers the Frank-Wolfe variants as an update rule for the weights along with the LPBoost as a sure guarantee for the convergence.[**?**]

3.2.1 Away Step Frank-Wolfe

The Away-step Frank-Wolfe algorithm is tailored to enhance convergence, especially in non-convex optimization scenarios. Unlike the original algorithm, Away-step Frank-Wolfe incorporates a crucial modification by identifying a step size that moves the current solution away from the gradient direction hence avoiding the zigzagging[**?**].

Algorithm3. (Away-Step FW)

Let $x^0 \in \mathscr{A}$, $S^0 := x^0$
for $k = 0, \ldots, T$ **do**
  Let $s^k = \arg\min_{s \in \mathscr{A}} < \nabla f(x^k), s >$ and $d_{\text{FW}} := s^k - x^k$
  Let $v^k \in \arg\max_{v \in S^k} < \nabla f(x^k), v >$ and $d_{\text{A}} := x^k - v^k$ ▷ (the away direction)
  if $g^k_{\text{FW}} := \left\langle -\nabla f(x^k), d_{\text{FW}} \right\rangle \leq \epsilon$ **then return** z^k
  end if
  if $\left\langle -\nabla f(x^k), d^k_{\text{FW}} \right\rangle \geq \left\langle -\nabla f(x^k), d^k_{\text{A}} \right\rangle$ **then** $d^k := d^k_{\text{FW}}$ ▷ (choose the FW direction)
  else: $d^k := d^k_{\text{A}}$ ▷ (choose the away-step direction)
  end if
  Choose γ^k
  Update $x^{k+1} := x^k + \gamma^k d^k$
  Update $S^{k+1} := \{v \in \mathscr{A} \text{ s.t. } \alpha^{k+1}_v > 0\}$
end for

The Away Frank-Wolfe algorithm checks two directions: the Frank-Wolfe direction also known as the "descent direction" first, and then the away direction.These steps are part of the process to determine the step size for the next iteration.

3.2.2 Short Step Frank-Wolfe

The short step Frank-Wolfe algorithm has a deliberate strategy of taking conservative, smaller steps at each iteration. By prioritizing proximity to the current solution, the variant maintains stability throughout the optimization process. It achieves this by carefully selecting step sizes that adhere to specific reduction criteria. Although it tends to converge more slowly compared to the other variants, its cautious approach can be valuable in scenarios where staying close to the current solution is crucial, such as in applications with tight constraints or when dealing with noisy data.

Algorithm : Short step rule $\mathcal{F}(A, \boldsymbol{w}_t, \boldsymbol{e}_{j_{t+1}}, \mathcal{E}_t, \boldsymbol{d}_t)$

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

3.2.3 Pairwise Frank-Wolfe

The Pairwise Frank-Wolfe is another variant that considers both the standard Frank-Wolfe and the Away step atoms to find the direction and then move along that direction[**?**].

Algorithm5. (Pairwise FW)

Let $x^0 \in \mathscr{A}$ and $S^0_x := x^0$
for $k = 0, \ldots, T$ **do**
  Let $s^k := \arg\min_{\mathscr{A}} < \nabla f(x^k), s >$ and $d_{\text{FW}} := s^k - x^k$
  Let $v^k \in \arg\max_{v \in S^k} < \nabla f(x^k), v >$
  if $g^k_{\text{FW}} := \left\langle -\nabla f(x^k), d_{\text{FW}} \right\rangle \leq \epsilon$ **then return** x^k
  end if
  Let $d^k := d^k_{\text{PFW}} := s^k - v^k$
  Choose γ^k
  Update $x^{k+1} := x^k + \gamma^k d^k$
  Update $S^{k+1} := \{v \in \mathscr{A} \text{ s.t. } \alpha^{k+1}_v > 0\}$
end for

The weight of the Frank-Wolfe is computed in the following way:

Algorithm : Pairwise rule $\mathcal{F}(A, \boldsymbol{w}_t, \boldsymbol{e}_{j_{t+1}}, \mathcal{E}_t, \boldsymbol{d}_t)$

Let $\boldsymbol{w}_t = \sum_{\boldsymbol{e} \in E_t} \alpha_{t,\boldsymbol{e}} \boldsymbol{e}$ be the current representation of $\boldsymbol{w}_t$ w.r.t. the basis vectors $E_t \subset \mathcal{E}_t$ with positive coefficients $\{\alpha_{t,\boldsymbol{e}}\}_{\boldsymbol{e} \in E_t}$.
Compute an *away* basis $\boldsymbol{e}^{\text{Away}} \in \arg\min_{\boldsymbol{e} \in E_t} \boldsymbol{d}_t^\top A\boldsymbol{e}$ and set $\lambda_{t,\max} = \alpha_{t,\boldsymbol{e}^{\text{Away}}}$.
Compute the step size $\lambda_t \leftarrow \arg\min_{\lambda \in [0, \lambda_{t,\max}]} \hat{f}^\star(-A(\boldsymbol{w}_t + \lambda(\boldsymbol{e}_{t+1} - \boldsymbol{e}^{\text{Away}})))$.

Ensure: $\boldsymbol{w}^{(1)}_{t+1} = \boldsymbol{w}_t + \lambda_t(\boldsymbol{e}_{j_{t+1}} - \boldsymbol{e}^{\text{Away}})$.

3.3. MLPBoost

The algorithm proposed by this paper [**?**] is called the MLPBoost which considers two updating rules, the LPBoost B and the one of FW variants. The update of the weight at iteration t+1 is the weight that minimizes the loss, resulting in improving the performance of the combined classifiers. It has also been proposed that changing the function in the rule B to its Fenchel conjugate makes the MLPBoost, an ERLPBoost. The authors have proven that the latter three boosting variants are instances of the Frank-Wolfe algorithm with different step sizes and objectives.The algorithm converges faster than ERLPBoost and results in low test error which makes it competitive with the LPBoost.

4. Experiments and Results

4.1. Datasets

The algorithms are tested on two datasets and the results are compared. The first dataset is Thyroid which consists of 215 data instances, five features, and one class label. The second dataset is Heart which has 270 data instances, 13 features, and one label. The label of both datasets are -1 and 1. We have also normalized the values of the feature to be in the range of [-1,1]. No further data preprocessing or transformation was conducted.

4.2. Experimental Settings

In our experiments, we used the same values of the parameters used in the paper[**?**].
We set ϵ= 0.01 (tolerance parameter) and we used a weak learner that is a decision tree of max depth 2. Due to the scarcity of our the resources we chose to work with a small number of iterations (50 iterations) as well as small number of instances datasets(215 and 270).
The code of the algorithm is written with Python 3 on Google Colab.

We also call our scheme with secondary algorithm (LPBoost) as MLPBoost and we refer to MLPB(AFW), MLPB(SS), and MLPB(PFW) are MLPBoosts with FW algorithms 3, 4, and 5 respectively.

For each dataset, we performed a 5-fold cross-validation and measure the test error with the test set as shown on Table 2.

4.3. Results

Dataset	Shape	MLPB(SS)	MLPB(AFW)	MLPB(PFW)
Thyroid	(215, 6)	0,05	0,05	0,05
Heart	(270, 14)	0,28	0,25	0,12

Table 2. Test errors for 5-fold cross validation for the best parameters.

Dataset	MLPB(SS)	MLPB(AFW)	MLPB(PFW)
Thyroid	539	1546	1379
Heart	878	2017	1837

Table 3. Comparison of the computation time in seconds.

Metrices	MLPB(SS)	MLPB(AFW)	MLPB(PFW)
number of iterations	50	50	50
Best accuracy train	0.94	0,93	0.93
Best accuracy test	0,95	0,90	0,83
Mean accuracy test	0.89	0,86	0,74
Mean AUC-ROC	0,91	0,86	0,64

Table 4. Other results for Thyroid dataset

Metrices	MLPB(SS)	MLPB(AFW)	MLPB(PFW)
number of iterations	50	50	50
Best accuracy train	0.79	0,79	0,77
Best accuracy test	0,72	0,75	0,78
Mean accuracy test	0.69	0,70	0,59
Mean AUC-ROC	0,76	0,79	0,56

Table 5. Other results for Heart dataset

Table 2 shows that the test errors using five-folder cross-validation, in the two dataset with the three MLPB algorithms. Table 3 shows the overall time for the three algorithms, it was calculate in seconds by subtracting the initial time and the end time. In the table 4 there are some other results obtained in the Thyroid dataset with 50 iterations and always using cross-validation, the best accuracy reached in the training and test set, the mean accuracy in the test set, and the mean of AUC-ROC value, that is another metrics to test if the model can discriminate well between the positive and the negative class, we obtain a value in a range between 0,7 and 0,9.

5. Conclusion

The aim of this project is to implement a boosting algorithm that iterates the weight updates between LPBoost and Frank-Wolfe variants in order to solve the issue of the soft-margin optimisation .

Our results show that the Thyroid dataset is able to reach a good values in all the metrics, the overall time increases a lot in the MLPB with away-step Frank-Wolfe and the Pairwise Frank-Wolfe. However, the Heart dataset has a lower performance but we still get a good value. We can say that this is due of the complexity of the dataset, the instances are similar but the features in the Heart dataset are more then twice this later so the methods need more iteration to capture all the information coming from all the features.

The results are constrained by our computation capacity, which made us unable to increase the number of iterations to obtain better results.

References

[1] John Shawe-Taylor Ayhan Demiriz, Kristin P. Bennett. Linear programming boosting via column generation. 11 2000.

[2] Weinan E Robert Schapire Chu Wang, Yingfei Wang. Functional frank-wolfe boosting for general loss functions. 10 2015.

[3] John Shawe-Taylo Jurij Leskovec. Linear programming boosting for uneven datasets.

[4] Gunnar Ra tsch Manfred K. Warmuth, Karen Glocer. Boosting algorithms for maximizing the soft margin.

[5] E. Takimoto R. Mitsuboshi, K. Hatano. Boosting as frank-wolfe. 10 2022.

[6] R. E. Schapire. The strength of weak learnability. machine learning. 1990.

[7] Martin Jaggi Simon Lacoste-Julien. On the global linear convergence of frank-wolfe optimization variants.