Training Soft-Margin SVMs using Frank-Wolfe, Away-Step Frank-Wolfe and Pairwise Frank-Wolfe Project Report

Mehran Farajinegarestan[†], Nazanin Karimi[‡]

***Abstract*—The project report delves into the optimization of Support Vector Machines (SVMs) utilizing the Frank-Wolfe algorithm and its variants, namely Away-Step Frank-Wolfe and Pairwise Frank-Wolfe. The focus is on the application of these algorithms in machine learning, specifically for linear classifiers. We highlight the concept of soft margin SVMs offering a flexible model compared to hard margin SVMs. The dual problem formulation of SVMs is also addressed, setting the stage for the application of Frank-Wolfe optimization methods. We demonstrate the linear convergence of the implemented algorithms, and show that the variants of Frank-Wolfe outperform the classic algorithm.**

***Index Terms*—Optimization, Frank-Wolfe, Away Step Frank-Wolfe, Pairwise Frank-Wolfe.**

I. Introduction

In this report, we delve into the optimizing Support Vector Machines (SVMs) using the Frank-Wolfe algorithm and its two notable variants: Away-Step Frank-Wolfe and Pairwise Frank-Wolfe. Our primary focus lies on the comparison of these algorithms with respect to their convergence rate and CPU time for training, specifically targeting linear classifiers. We explore the concept of soft margin SVMs, presenting a more flexible model alternative to the traditional hard margin SVMs. A detailed analysis of the dual problem formulation of SVMs sets the stage for applying the Frank-Wolfe optimization methods. Through our investigations, we demonstrate the linear convergence of these algorithms and the superior performance of Frank-Wolfe variants over the classical Frank-Wolfe algorithm.

The report is structured as follows: section II describes Support Vector Machines (SVMs) and their formulation. Section III presents Frank-Wolfe algorithm and its variants, also the concept of duality gap. Section IV presents some details regarding the implementation of these algorithms. Section V illustrates the main results of experiments and finally section VI presents the concluding remarks.

II. Support Vector Machines

SVMs: Support Vector Machines (SVMs) represent a prominent category of linear classifiers within the domain of machine learning. Their primary objective is not merely to identify any separating hyperplane between two classes of data; instead, it is to determine the hyperplane that ensures the largest possible margin between these classes. The margin is conceptualized as the minimal distance from the hyperplane to the nearest data points [1].

The essence of SVMs is encapsulated in their optimization problem, which aims to maximize this margin. The optimization is mathematically framed as:

$$\max_{w\in\mathbb{R}^d} \min_i \frac{y_i X_i^T w}{\|w\|} \tag{1}$$

where X_i denotes the data points, and Y_i represents the labels associated with these points, taking values of either $+1$ or -1 for the two different classes.

The importance of SVMs in machine learning stems from their ability to locate the most robust dividing hyperplane, which is vital for ensuring effective generalization to unseen data. The SVM optimization framework is adaptable and supports enhancements such as kernel methods, which enable SVMs to efficiently handle non-linear classification tasks.

Soft Margin SVMs: The concept of soft margin is crucial in the practical application of SVMs, particularly for handling outliers. Unlike the hard margin approach, which strictly separates classes without allowing any violations, the soft margin approach introduces flexibility. This flexibility allows some data points (outliers) to be on the wrong side of the margin, providing a more adaptable model in real-world scenarios where a perfect separation is often not achievable.

The primal optimization problem of the two-class soft margin SVM with squared loss (ℓ_2-loss), can be formalized as follows:

$$\begin{aligned} \min_{\bar{w}\in\mathbb{R}^d,\rho\in\mathbb{R},\xi\in\mathbb{R}^n} & \frac{1}{2}|\bar{w}|^2 - \rho + \frac{C}{2}\sum_{i=1}^{n}\xi_i^2 \\ \text{s.t.} \quad & y_i \cdot \bar{w}^T X_i \geq \rho - \xi_i \quad \forall i \in \{1,\dots,n\} \end{aligned} \tag{2}$$

This problem can be described in both cases with a regularized offset or without any offset term. In this formula, $\bar{w}$ denotes the weight vector, ρ is a scalar related to the margin, and ξ_i are slack variables corresponding to each data point. The parameter C plays a key role as it determines the trade-off between maximizing the margin and penalizing the margin violations.

Dual Problem: A significant feature of SVMs is their dual formulation. By applying the Lagrange dual to the primary

[†]Department of Mathematics, University of Padova, email: {mehran.farajinegarestan}@studenti.unipd.it

[‡]Department of Mathematics, University of Padova, email: {Nazanin.Karimi}@studenti.unipd.it

SVM optimization problem, we obtain:

$$\max_{x} \min_{i} A_i^T \frac{Ax}{|Ax|} \tag{3}$$

"In this formulation, A represents the matrix that contains the data points with their signs, which can be written as $A_i = y_i X_i$, and $A \in R^{d \times n}$. x is a non-negative vector whose entries sum up to one. We have $x \in \Delta \subset R^n$ in which Δ is the set of probability vectors called unit simplex in R^n. Given any feasible x we can obtain a corresponding classifier $w = Ax$. Since x is a lies in the simplex, vector w is a convex combination of the data points in the training dataset. Based on the definition in [1], The data points corresponding to non-zero entries in x are called the support vectors.

Conversely, the dual form of the soft-margin SVM can be formulated as:

$$\min_{x \in \Delta} ||Ax||^2 \tag{4}$$

Where, $A := \begin{pmatrix} Z \\ \frac{1}{\sqrt{C}} I_n \end{pmatrix} \in R^{(d+n) \times n}$ and $Z \in R^{d \times n}$ is the data matrix, with $Z_i := y_i X_i \quad \forall i = 1, ..., n$.

The dual form of the soft-margin SVM represented above will be used to optimize it using Frank-Wolfe algorithm and its variants.

III. Frank-Wolfe Algorithms

Frank-Wolfe: The Frank-Wolfe algorithm, introduced in 1956 and cited in [2], has gained popularity in the field of Data Science, especially for addressing constrained convex optimization problems. This method stands out for its simplicity as one of the earliest approaches to constrained optimization. The core idea of the Frank-Wolfe algorithm is to solve optimization problems of the form:

$$\min_{x \in D} f(x)$$

where f is a convex and continuously differentiable objective function, and D is a compact convex subset of a vector space. The algorithm is distinguished by its iterative approach, efficiently navigating the feasible region defined by D to find the minimum of f.

In the context of applying the Frank-Wolfe algorithm to SVMs, the domain D represents the unit simplex denoted as Δ. Algorithm 1 describes the Frank-Wolfe algorithm implemented in this project.

While the step size in the original Frank-Wolfe algorithm is $\gamma := \frac{2}{k+2}$, dependent on the number of iterations, we utilized line search to determine the optimal step size. At each step of Frank-Wolfe algorithm, using $\nabla f(x^{(t)})$, we consider the linearization of the objective function and move towards this linear function which means solving less expensive linear subproblems. In terms of convergence rate we have $f(x^{(t)}) - f(x^*) \leq O(\frac{1}{t})$, x^* being an optimal solution of our problem.

Duality Gap: It is important to define the Duality Gap as it provides a certificate for the approximation quality. The duality gap for any constrained convex optimization problem and a feasible point $x \in D$ is defined as:

$$g(x) = \max_{s \in D} \langle x - s, \nabla f(x) \rangle$$

The duality gap $g(x)$ is a surrogate for the difference between the linearization of the objective function f at the point x and its optimal value over the domain D. In the context of algorithms like Frank-Wolfe, when s is a minimizer of the linearized problem at an arbitrary point x, then s serves as a certificate for the current duality gap. It is easy to show that for any feasible point x we have: $g(x) \geq f(x) - f(x^*)$.

Hence, the convergence of the Frank-Wolfe algorithm, including primal-dual convergence, is often analyzed in terms of the duality gap.

Algorithm 1 Frank-Wolfe algorithm: FW$(x^{(0)}, \Delta, \epsilon)$

1: Let $x^{(0)} \in \Delta$
2: **for** $t = 0...T$ **do**
3: Compute $s := arg \min_{s \in \Delta} \langle s, \nabla f(x^{(t)}) \rangle$
4: **if** $g_t^{FW} := \langle -\nabla f(x^{(t)}), s - x^{(t)} \rangle \leq \epsilon$ **then** return $x^{(t)}$ (FW gap is small enough, End of algorithm)
5: **end if**
6: Step-Size $\gamma := arg \min_{\gamma \in [0,1]} f(x^{(t)} + \gamma(s - x^{(t)}))$ (Line-Search for the Step-Size γ)
7: Update $x^{(t+1)} := (1 - \gamma)x^{(t)} + \gamma s$
8: **end for**

Away Steps: Away Steps is a technique used to enhance the convergence rate of the algorithm, particularly in scenarios where the solution lies at the boundary of the feasible region.

In the classic Frank-Wolfe algorithm, each iteration involves moving towards a new vertex in the feasible region that minimizes the linear approximation of the objective function. However, this approach can lead to inefficient "zig-zagging" when the algorithm approaches the boundary of the feasible region, resulting in a slow convergence rate [3].

The away step addresses this issue by enabling the algorithm to move not only towards new vertices but also away from vertices in the current convex combination that are not contributing effectively to the descent direction. Specifically, the algorithm identifies an "away vertex" from the current convex combination that is most aligned with the ascent direction of the objective function. By moving away from this vertex, the algorithm can make more direct progress towards the optimal solution.

This approach reduces the zig-zagging behaviour and accelerates convergence. By incorporating away steps, the Frank-Wolfe algorithm becomes more efficient in navigating the feasible region and converging to the optimal solution. We describe the away steps variant in algorithm 2.

The main parts of the algorithm are as follows:

- Frank-Wolfe Direction :

$$d_t^{FW} = s_t - x^{(t)}$$

Where s_t is the vertex from the Linear Minimization

Algorithm 2 Away-steps Frank-Wolfe algorithm: AFW$(x^{(0)}, \mathcal{A}, \epsilon)$

Let $x^{(0)} \in \mathcal{A}$, and $S^{(0)} := \left\{x^{(0)}\right\}$ (so that $\alpha_v^{(0)} = 1$ for $v = x^{(0)}$ and 0 otherwise)
for $t = 0...T$ **do**
  Let $s_t := LMO_{\mathcal{A}}(\nabla f(x^{(t)}))$ and $d_t^{\text{FW}} := s_t - x^{(t)}$ (the FW direction)
  Let $v_t \in \arg\max_{v \in S^(t)} \langle \nabla f(x^{(t)}), v \rangle$ *and* $d_t^{\text{A}} := x^{(t)} - v_t$ (the away direction)
  if $g_t^{FW} := \langle -\nabla f(x^{(t)}), d_t^{FW} \rangle \leq \epsilon$ **then** return $x^{(t)}$ (FW gap is small enough, End of algorithm)
  end if
  if $\langle -\nabla f(x^{(t)}), d_t^{FW} \rangle \geq \langle -\nabla f(x^{(t)}), d_t^{A} \rangle$ **then**
    $d_t := d_t^{FW}$, and $\gamma_{max} := 1$ (choose the FW direction)
  else
    $d_t := d_t^{A}$, and $\gamma_{max} := \alpha_{v_t}/(1 - \alpha_{v_t})$ (choose away direction; maximum feasible step-size)
  end if
  Line-search: $\arg\min_{\gamma \in [0, \gamma_m ax]} f(x^{(t)} + \gamma d_t)$
  Update $x^{(t+1)} := x^{(t)} + \gamma_t d_t$
  Update $S^{(t+1)} := \{v \in \mathcal{A} \;\; s.t. \; \alpha_v^{(t+1)} > 0\}$
end for

Oracle (LMO) over the gradient of the objective function at iteration t, and $x^{(t)}$ is the current position.

- Away Direction (Away Step):

$$d_t^A = x^{(t)} - v_t$$

Here, v_t is the "away vertex" from the current active set $S^{(t)}$ that maximizes the inner product with the gradient of the objective function.

- Step Size Determination:

$$\arg\min_{\gamma \in [0, \gamma_m ax]} f(x^{(t)} + \gamma d_t)$$

The step size γ_t is calculated through a line search.

- Update Rule:

$$x^{(t+1)} = x^{(t)} + \gamma_t \cdot d_t$$

The direction d_t can be either d_t^{FW} (FW direction) or d_t^A (away step), chosen based on which direction provides a better decrease in the objective function.

Pairwise: The Pairwise Frank-Wolfe (PFW) algorithm is another variant of the classic Frank-Wolfe optimization algorithm. This variant is especially useful in scenarios where the objective is to maintain sparse solutions and control the size of the active set, which can be crucial in high-dimensional optimization problems.

In each iteration, the Pairwise Frank-Wolfe algorithm adjusts weights from a selected 'away' atom (v_t) to a 'FW' atom (s_t). Specifically, the algorithm identifies an 'away' vertex (atom) from the active set and a 'FW' vertex from the LMO oracle. The update direction is then determined as the difference between these two vertices. The step size is carefully chosen, ensuring that the solution remains within the feasible region. This approach differs from the classic Frank-Wolfe algorithm, which may adjust weights across the entire active set.

Algorithm 3 Pairwise Frank-Wolfe algorithm: PFW$(x^{(0)}, \mathcal{A}, \epsilon)$

... as in Algorithm 2, except replacing lines 7 to 11 by: $d_t = d_t^{PFW} := s_t - v_t$, and $\gamma_{max} := \alpha_{v_t}$

The main parts of the algorithm are as follows:

- Pairwise Direction (Pairwise Step):

$$d_{\text{PFW}} = s_t - v_t$$

In this formula, d_{PFW} represents the pairwise direction in the Pairwise Frank-Wolfe algorithm, where s_t is the atom obtained from the Linear Minimization Oracle (LMO), and v_t is the 'away atom' from the current active set.

- Maximum Step Size (γ_{max}):

$$\gamma_{\text{max}} = \alpha_{v_t}$$

Here, γ_{max} is the maximum step size in the PFW algorithm, determined by the weight α_{v_t} of the 'away atom' v_t.

IV. IMPLEMENTATION

We implemented the following algorithms:

- Classic Frank-Wolfe (FW): The algorithm described in [2], implemented as algorithm 1.
- Away Step Frank-Wolfe (AFW): The algorithm described in [3], implemented as algorithm 2.
- Pairwise Frank-Wolfe (PFW): The algorithm described in [3], implemented as algorithm 3.

In this section we discuss the implementation details of these algorithms.

A. Starting Point ($x^{(0)}$)
The starting point is a vector with all elements set to zero, except the first one, which is set to one. The size of the vector is equal to the number of examples in the dataset.

B. Linear Minimization Oracle (LMO)
The solution of the linear minimization oracle is the vertex with the smallest gradient value.

C. Step Size (γ)
All algorithms employed Armijo line search to determine the optimal step size for each iteration. This algorithm defined as follows: for fixed parameters δ and α with $\delta \in (0, 1)$ and $\alpha \in \left(0, \frac{1}{2}\right)$ and a maximum step size γ_{max} we try steps with $m = 0, 1, 2, ...$ until the inequality is satisfied.

$$\gamma = \delta^m \gamma_{max}$$

$$f(x^{(t)} + \alpha d_t) \leq f(x^{(t)}) + \gamma \alpha \nabla f(x^{(t)})^T d_t \qquad (5)$$

We used $\alpha = 0.1$ and $\delta = 0.8$ in our implementation. Additionally, after 1000 iterations, if the condition was not satisfied, the new point was chosen based on the last calculated step size. However, in our experiments this never happened.

D. Prediction

While [3] and [2] exclusively discuss the optimization of Frank-Wolfe and its variants, and [1] focuses solely on the theory behind SVM models, Simply finding the optimal value of $x^{(t)}$ is not sufficient; we also need to obtain a hyperplane that partitions the space into two separate parts to make predictions for new data points. To do so after obtaining $x^{(t)}$ close enough to x^*, we can obtain $w' = Ax^{(t)} \in R^{d+1+n}$. Here d being the number of dimensions of our data points. The first $d+1$ elements of $Ax^{(t)}$ is the normal vector w that the $d+1th$ dimension represents the offset parameter b. This way the hyperplane does not necessarily have to pass through the origin. Given a new data point X_i, to make a prediction we simply need to first increase the dimensionality of X_i by one with a value equal to one. If $X^T w < 0$ we have $y = -1$ and if $X^T w > 0$ we have $y = +1$

V. Illustrative Experiments

We carried out numerical experiments on three different data sets to compare the performance of the Frank-Wolfe algorithm and its variants. The selected datasets vary in terms of the number of training data instances, dimensions per dataset, and the sparsity of each data sample. Below is a brief description of each dataset:

1) The "a4a" dataset is derived from the Adult dataset and is associated with John C. Platt's work on fast training of support vector machines using sequential minimal optimization. The original Adult dataset has 14 features, with six continuous and eight categorical features. In the "a4a" dataset, continuous features are discretized into quantiles, and each quantile is represented by a binary feature. Additionally, categorical features with multiple categories are converted into binary features. The dataset is binary-class with two classes, and the number of instances for training and testing are 1,605 and 30,956, respectively. The dataset contains 123 features for both training and testing sets.
2) The "liver-disorders" dataset is introduced in the work by James McDermott and Richard S. Forsyth focused on diagnosing disorders in a classification benchmark. It is a binary-class dataset for classification. The dataset comprises 145 instances designated for training and an additional 200 instances for testing. With a minimal set of features, the dataset contains only five features, providing a compact representation for diagnostic classification tasks related to liver disorders.
3) The "svmguide1" dataset is featured in the technical report by Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, offering practical guidance on support vector classification. The dataset originates from an astroparticle application provided by Jan Conrad at Uppsala University, Sweden. It is a binary-class dataset with two classes, consisting of 3,089 instances allocated for training and 4,000 instances for testing. With a minimal set of features, the dataset comprises four features, making it suitable for support vector classification tasks in the specified application domain.

For the a4a and liver-disorders datasets the algorithms were run for maximum of 5000 iterations and for the svmguide1 dataset, they were run for 1000 iterations. For each algorithm training on each dataset, the duality gap, CPU time and number of iterations were recorded. Figures 1 to 6 illustrate the duality gap versus number of iterations and the duality gap versus CPU time.

In all of our experiments Pairwise Frank-Wolfe outperforms the classic Frank-Wolfe and Away Step Frank-Wolfe algorithm. Away Step Frank-Wolfe performs slightly better than classic one. Additionally, our plots demonstrate the linear convergence of the algorithms. All of the results obtained in our experiments are consistent with the results obtained in [3]. Additionally, in terms of CPU time per iteration the Pairwise Frank-Wolfe is much more efficient than the others and runs faster.

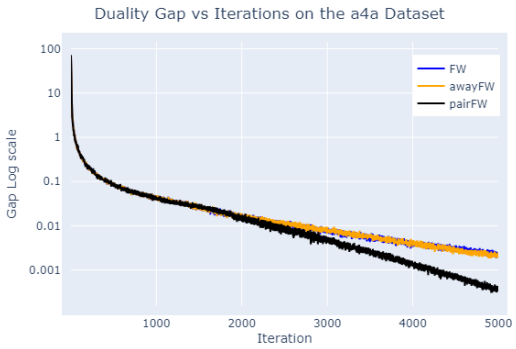

Fig. 1: Duality Gap vs Iterations on the a4a Dataset

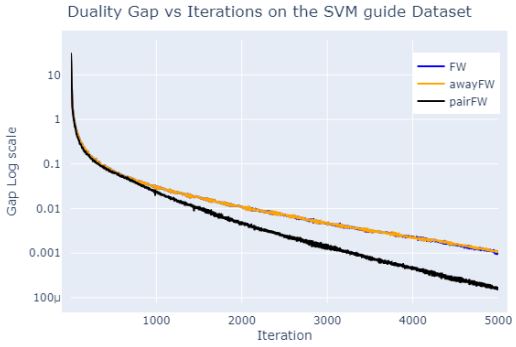

Fig. 2: Duality Gap vs Iterations on the SVM guide Dataset

Although the convergence of the algorithms and their performance regarding the duality gap per iteration were the main focus of our project, we also tested the trained models

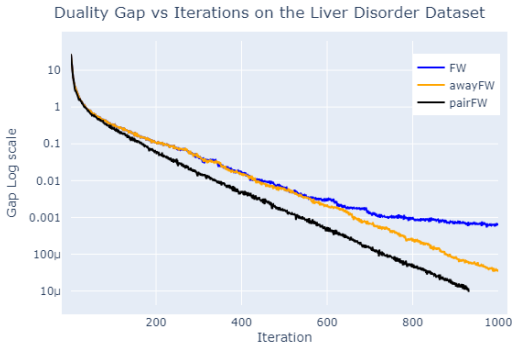

Fig. 3: Duality Gap vs Iterations on the Liver Disorder Dataset

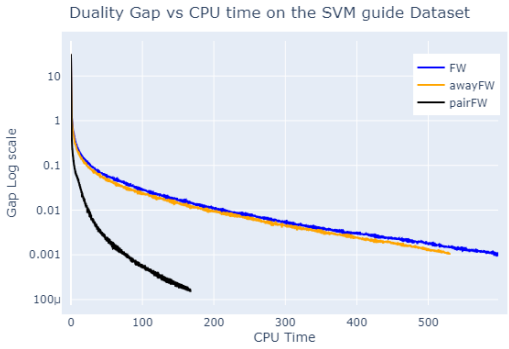

Fig. 5: Duality Gap vs CPU time on the SVM guide Dataset

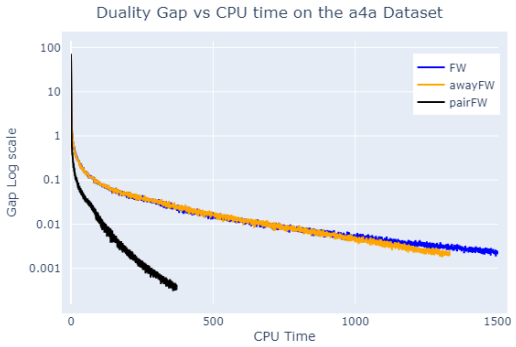

Fig. 4: Duality Gap vs CPU time on the a4a Dataset

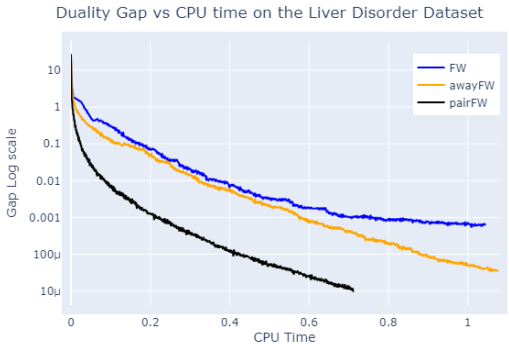

Fig. 6: Duality Gap vs CPU time on the Liver Disorder Dataset

on the test datasets. As it can be seen in the Table 1 the performance of each algorithm on binary classification task is almost the same with Pairwise algorithm slightly performing better that the others. Note that the ratio of the negative class in the a4a dataset is 0.78, so accuracy is not the best metric to demonstrate the performance of the algorithms. For other datasets this ratio is 0.50. Here the purpose is just to show the ability of the trained models to make predictions on unseen data.

TABLE 1: Accuracy of Algorithms on the Test Sets

Test Set	**FW**	**Away-Step FW**	**Pairwise FW**
a4a	84.64%	84.58%	84.63%
SVM Guide	93.45%	93.53%	93.58%
Liver Disorder	58.50%	58.50%	58.50%

VI. Concluding Remarks

In conclusion, we investigated the Frank-Wolfe algorithm and its variants in optimizing the dual form of soft-margin SVMs. The Away-Step and Pairwise Frank-Wolfe algorithms are highlighted for their efficiency and effectiveness in handling the optimization challenges posed by SVMs. We conclude by affirming the superiority of these algorithms, particularly the Pairwise Frank-Wolfe, in terms of performance and efficiency. The results obtained from the implementation of these algorithms are consistent with existing literature, confirming their potential in enhancing machine learning models, especially in scenarios requiring robust classification.

References

[1] M. Jaggi, "An equivalence between the lasso and support vector machines," *CoRR*, vol. abs/1303.1152, 2013.

[2] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *Naval Research Logistics Quarterly*, vol. 3, pp. 95–110, 1956.

[3] S. Lacoste-Julien and M. Jaggi, "On the global linear convergence of frank-wolfe optimization variants," 2015.