Frank Wolfe Variants for SVMs training

CHEUNG, Hau Yee
hauyee.cheung@studenti.unipd.it

HUANG, Ze Sen
zesen.huang@studenti.unipd.it

Abstract

In this paper, firstly we focus on applying variants of Frank Wolfe (FW) algorithms, including original FW, Away FW (AFW) and Pairwise FW (PFW), for training soft margin Support Vector Machines (SVM). Then, we analyze the equivalence of Lasso and SVM. Then,Two datasets are going to be implemented to the SVM models by using different Frank Wolfe algorithm and finally compare the performance.

1. Introduction

The Frank Wolfe algorithm is one of the non-linear constrained optimization algorithms, which solves nonlinear problems by solving a sequence of linear problems. As the output of each update is always a vertex of the domain, it can benefit in situations with a large number of features, such as optimization in neural networks. To improve the rate of convergence, Away FW and Pairwise FW are designed, which enjoy a linear convergence in batch size. The details of soft margin SVM are going to be trained by applying a variant FW method (original FW, Away FW, Pairwise FW) will be discussed in section 3.

SVM is commonly used due to its kernel trick technique, it works well in high dimensional space and provides good results in general. SVM is one of the classification techniques which aim to minimize misclassification errors. It draws hyperplanes to separate the groups according to the pattern, where the hyperplanes try to maximize the margin between classes. Soft margin allows SVM to make a certain number of mistakes and keep the margin as wide as possible to maintain the overall correct classification rate. The Least Absolute Shrinkage and Selection Operator (LASSO) is one of the regularizations of the data models and feature selection method. LASSO uses shrinkage to shrink data values towards a central point as the mean. As both soft margin SVM and LASSO also minimize the sum of margin of each variable with a cost penalty, they provide similar results. In the following section 4, we are going to go in-depth about the equivalence of soft margin SVM and LASSO.

2. Dataset

The two datasets used for this project to compare the performances of the algorithms are small datasets that can be downloaded freely from https://www.csie.ntu.edu.tw/˜cjlin/libsvmtools/datasets/. The first one is called svmguide1 consisting of 3089 training samples and 4000 testing samples with 4 features. The second dataset is called splice including 1000 training samples and 2175 testing samples with 60 features. Both datasets come with binary results which were labeled as 1 and -1. Data preprocessing is a technique that transforms raw data into an understandable format. We applied StandardScaler in the svmguide1 dataset to remove the mean and scale each feature to unit variance. This operation is performed feature-wise in an independent way. StandardScaler can be influenced by outliers since it involves the estimation of the empirical mean and standard deviation of each feature.

3. Optimisation methods

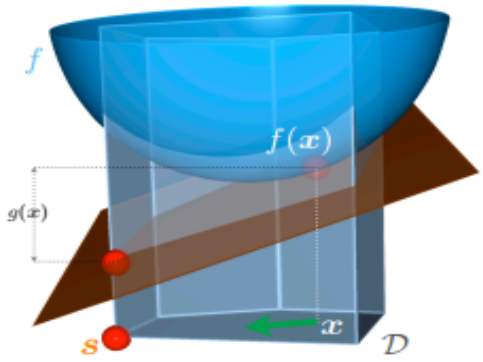
3.1. Original FW

The Frank-Wolfe algorithm is also called the conditional gradient method[2][3]. This algorithm and its variants have recently gained a lot of attention in the Machine Learning community because they avoid the use of projection operators to ensure the feasibility of the iterates. It is a first-order algorithm for constrained optimization that avoids the use of projections

at each iteration. The dual gap (and its stronger variants) is extremely useful as a progress and stopping criterion when running Frank-Wolfe algorithms. The details of the algorithm are attached in Figure 1.

There are three major steps involved in the algorithm. Firstly, direction finding, solves the following linear program:

$$s := \underset{s \in D}{argmin}(s, \bigtriangledown f(x^{(k)})) \tag{1}$$



Secondly, step size determination, there are few possibilities to choose the step size to archive a convergence, including Inversely depending on the number of iterations $(2/t + 2)$, picking the minimizing point on the segment, considering the complete history and also the latter step choices making each optimization more aggressive. In the algorithm shown in Figure 1, the step size inversely depends on the number of iterations $(2/t + 2)$. Finally, update the weight and go back to the first step.

This algorithm has a very slow convergence rate (at the order of $O(1/t)$). On the other hand, this algorithm is very simple, using only linear optimization and iterates through at most t vertices after t steps.

Algorithm 1 Frank-Wolfe (1956)

Let $\boldsymbol{x}^{(0)} \in \mathcal{D}$
for $k = 0 \ldots K$ **do**
  Compute $\boldsymbol{s} := \arg\min_{\boldsymbol{s} \in \mathcal{D}} \left\langle \boldsymbol{s}, \nabla f(\boldsymbol{x}^{(k)}) \right\rangle$
  Update $\boldsymbol{x}^{(k+1)} := (1-\gamma)\boldsymbol{x}^{(k)} + \gamma \boldsymbol{s}$, for $\gamma := \frac{2}{k+2}$
end for

Figure 1. Frank-Wolfe algorithm

3.2. Away FW

Away FW is designed to improve the problem Zig-Zag faced in the original FW by providing away steps. In some special problems, Away FW also provides faster linear convergence results. Firstly, we set the original FW point with equation: $\hat{x}_k^{FW} = \underset{x \in C}{Argmin} \bigtriangledown f(x_k)^T(x - x_k)$ and also away FW point with equation: $\hat{x}_k^{AS} = \underset{x \in S_k}{Argmax} \bigtriangledown f(x_k)^T(x - x_k)$. The Away-step direction points move away from the worst vertex, which is the one with the highest value of the linearized function described in the current iteration. Then, define the direction of FW with an equation: $d_k^{FW} = \hat{x}_k^{FW} - x_k$ and the direction of Away FW with equation $d_k^{AS} = x_k - \hat{x}_k^{AS}$. If $\bigtriangledown f(x_k)^T d_k^{FW} \leqslant \bigtriangledown f(x_k)^T d_k^{AS}$, then set d_k as the FW direction, else set the Away FW direction. Afterwards, update the weight, $x_{k+1} = x_k + \alpha d_k$, with a suitable stepsize α. Finally, calculate S_{k+1} with a set of currently used vertices.

3.3. Pairwise FW

The pairwise Frank-Wolfe algorithm is mostly similar to the Away Frank-Wolfe algorithm, except the away direction is replaced by the pairwise direction. The pairwise direction is the direction from the away atom to the FW atom. The main

idea is to only move weight mass between two atoms in each step. The pairwise FW convergence rate is looser than other algorithms. The details of the away FW and pairwise FW algorithms are attached in Figure 2.

Algorithm 1 Away-steps Frank-Wolfe algorithm: $\mathbf{AFW}(x^{(0)}, \mathcal{A}, \epsilon)$

1: Let $x^{(0)} \in \mathcal{A}$, and $S^{(0)} := \{x^{(0)}\}$ *(so that $\alpha_v^{(0)} = 1$ for $v = x^{(0)}$ and 0 otherwise)*
2: **for** $t = 0 \dots T$ **do**
3: Let $s_t := \mathrm{LMO}_{\mathcal{A}}\left(\nabla f(x^{(t)})\right)$ and $d_t^{\mathrm{FW}} := s_t - x^{(t)}$ *(the FW direction)*
4: Let $v_t \in \arg\max_{v \in S^{(t)}} \left\langle \nabla f(x^{(t)}), v \right\rangle$ and $d_t^{\mathrm{A}} := x^{(t)} - v_t$ *(the away direction)*
5: **if** $g_t^{\mathrm{FW}} := \left\langle -\nabla f(x^{(t)}), d_t^{\mathrm{FW}} \right\rangle \le \epsilon$ **then return** $x^{(t)}$ *(FW gap is small enough, so return)*
6: **if** $\left\langle -\nabla f(x^{(t)}), d_t^{\mathrm{FW}} \right\rangle \ge \left\langle -\nabla f(x^{(t)}), d_t^{\mathrm{A}} \right\rangle$ **then**
7: $d_t := d_t^{\mathrm{FW}}$, and $\gamma_{\max} := 1$ *(choose the FW direction)*
8: **else**
9: $d_t := d_t^{\mathrm{A}}$, and $\gamma_{\max} := \alpha_{v_t}/(1 - \alpha_{v_t})$ *(choose away direction; maximum feasible step-size)*
10: **end if**
11: Line-search: $\gamma_t \in \arg\min_{\gamma \in [0, \gamma_{\max}]} f\left(x^{(t)} + \gamma d_t\right)$
12: Update $x^{(t+1)} := x^{(t)} + \gamma_t d_t$ *(and accordingly for the weights $\alpha^{(t+1)}$, see text)*
13: Update $S^{(t+1)} := \{v \in \mathcal{A} \text{ s.t. } \alpha_v^{(t+1)} > 0\}$
14: **end for**

Algorithm 2 Pairwise Frank-Wolfe algorithm: $\mathbf{PFW}(x^{(0)}, \mathcal{A}, \epsilon)$

1: ... as in Algorithm 1, except replacing lines 6 to 10 by: $d_t = d_t^{\mathrm{PFW}} := s_t - v_t$, and $\gamma_{\max} := \alpha_{v_t}$.

Figure 2. Away & Pairwise Frank-Wolfe algorithm

4. The Equivalence between SVM and Lasso

We are going to introduce the SVM and Lasso in the following subsections.

4.1. Lasso regression

Lasso regression is also called L1 regression. It is a regularization method by decreasing the generalization error. Moreover, since L1 regression can shrink the slope to zero, which can exclude useless variables from equations. It reduces the variance in the model that contains a lot of useless variables to archive the feature selection function.

The loss function for the Lasso problem is given by the quadratic program in equation 2:

$$min_{x \in \blacklozenge} \parallel Ax - b \parallel^2 \tag{2}$$

where $\blacklozenge$ is the l_1 *unit ball* in R^n, the matrix $A \in R^{d \times n}$ contains all n datapoints as its columns, and the right hand side b is a fixed vector $b \in R^d$.

4.2. Support Vector Machine

SVM algorithm is a supervised learning method for classification, regression, and detection. The objective of the SVM algorithm is to find a hyperplane plane in an N-dimensional space that classifies the data point. The hyperplane should maximize margin, which means maximizing the distance between data points between classes. The loss function helps us to

minimize the loss when we find the optimization margin. Hinge loss function (L1-loss) is the most commonly used in SVM. In Jaggi's paper, he only included the dual of the soft margin SVM function (L2-loss) instead of the Hinge loss function. Hence, we only consider the L2-loss in our report. SVM with a soft margin allows misclassification to happen, and the loss of misclassified points is called a slack variable. A new regularization parameter C controls the trade-off between maximizing the margin and minimizing the loss. We focus on large margin linear classifiers, and more precisely on those SVM variants whose dual optimization problem is of the form in equation 3:

$$min_{x\in\Delta} \parallel Ax \parallel^2 \tag{3}$$

Here the Δ is the unit simplex in R^n.

The dual of the soft-margin SVM that we solved in the work is an instance of the classifier formulation, that is$min_{x\in\Delta}\parallel Ax \parallel^2$, with

$$A := \begin{pmatrix} Z \\ \frac{1}{\sqrt{C}I_n} \end{pmatrix} \in R^{(d+n)n} \tag{4}$$

where the data matrix $Z \in R^{dn}$ consists of the n columns $Z_i := y_i X_i$

4.3. The Equivalence

Kernelization is the major characteristic of SVM, which is a technique using a linear classifier to solve a nonlinear problem. The Kernel function is applied on each data instance to map the original nonlinear observation into a higher-dimensional space in which it becomes more separable. In the paper, Jaggi tried to transfer this characteristic to the LASSO problem.

The screening rule is a way to preprocessing the input data for a LASSO problem, which explicitly identifies active structures. The author tried to apply the screening rule to SVM, in order to eliminate potential support vectors before training. The screening rule then guarantees that any discarded point will not be a support vector, so the resulting optimal classifier remains unchanged.

Equivalence between SVM and Non-Negative Lasso [1]. In a non-negative Lasso instance, its each column vector of the matrix A can be translated by the vector $-b$. With this translation, we change an SVM instance, with the data matrix being

$$\widetilde{A} := A \ - \ b1^T \ \in R^{d\times n}$$

We should make sure that $b1^T x = b$ for any $x \in \Delta$.

Given a Lasso Instance, Constructing an Equivalent SVM Instance. Note that the Lasso is with l_1 $unit\, ball$ and SVM is with unit simplex, thus we should represent the l_1 $ball$ ♦ by a simplex Δ. The l_1 $ball$ ♦ is the convex hull of its $2n$ vertices, which are $\{e_i \mid i \in [1..n]\}$, illustrating why ♦ is also called the cross-polytope. Formally, any n-vector $x_\diamondsuit \in$ ♦ can be written as

$$x_\diamondsuit = (I_n \mid I_n)x_\Delta \ \ for\, x_\diamondsuit \in \Delta \subset R^{2n}$$

Here $x_\diamondsuit$ is a 2n-vector, and we used the notation $(A \mid B)$ for the horizontal concatenation of two matrices$A$, B. Therefore, given a Lasso instance of the form we can directly parameterize the l_1 $ball$ ♦ by the $2n$-dimensional simplex as described above. By writing $(In \mid In)x_\diamondsuit$ for any $x \in$ ♦, the objective function becomes $\parallel (A \mid A)x_\diamondsuit \ - \ b \parallel^2$. Through the translation, we have the equivalent to the SVM formulation, which is:

$$min_{x\in\Delta} \parallel \widetilde{A}x_\diamondsuit \parallel^2$$

where the data matrix is given by

$$\widetilde{A} \ := \ (A \mid -A) \ - \ b1^T \ \in R^{d\times 2n}$$

From the translation, We can learn that for any points solution $x \in R^n$ to the Lasso, we get a feasible SVM point $x_\diamondsuit \in \Delta \subset R^{2n}$ of the same objective value, and vice versa.

Given an SVM Instance, Constructing an Equivalent Lasso Instance. We first define the Lasso instance $(\widetilde{A}, \widetilde{b})$ as the translated SVM datapoints

$$\widetilde{A} := \{A_i \ + \ \widetilde{b} \mid i \in [1..n]\}$$

together with the right hand side

$$\widetilde{b} := \ -\frac{w}{\parallel w \parallel} \cdot \frac{D^2}{\sigma}$$

Here D > 0 is a strict upper bound on the length of the original SVM datapoints, i.e.$\| A_i \| < D \, \forall i$ Therefore, we can obtain the new Lasso objective function by the definition of $\widetilde{A}$:

$$\| \widetilde{A}x - \widetilde{b} = \| (A + \widetilde{b}1^T)x - \widetilde{b} \| = \| Ax + (1^T x - 1)\widetilde{b} \|$$

5. Discussion

The line search approach first finds a descent direction along which the objective function will be reduced and then computes a step size that determines how far it should move along that direction. The function value is the value at that iteration closer to the absolute minimum of the function. In the datasets we tested in this project, we only picked step size equal to inversely depending on the number of iterations $(2/t + 2)$ for all the algorithms tried. Figure 3 and Figure 4 below show that the function value decreased exponentially in both the original Frank-Wolfe algorithm and Away Frank-Wolfe Algorithm and at the end it achieved a state in which the function value settles to within an error range around the final stage. Figure 5 and Figure 6 below show the duality gap with the same decreasing trend as the function value. In other words, the model converges when additional training will not improve the model. Although the function values in the pairwise Frank-Wolfe algorithm in both datasets fluctuated at the beginning of model training, at the end it also converged.

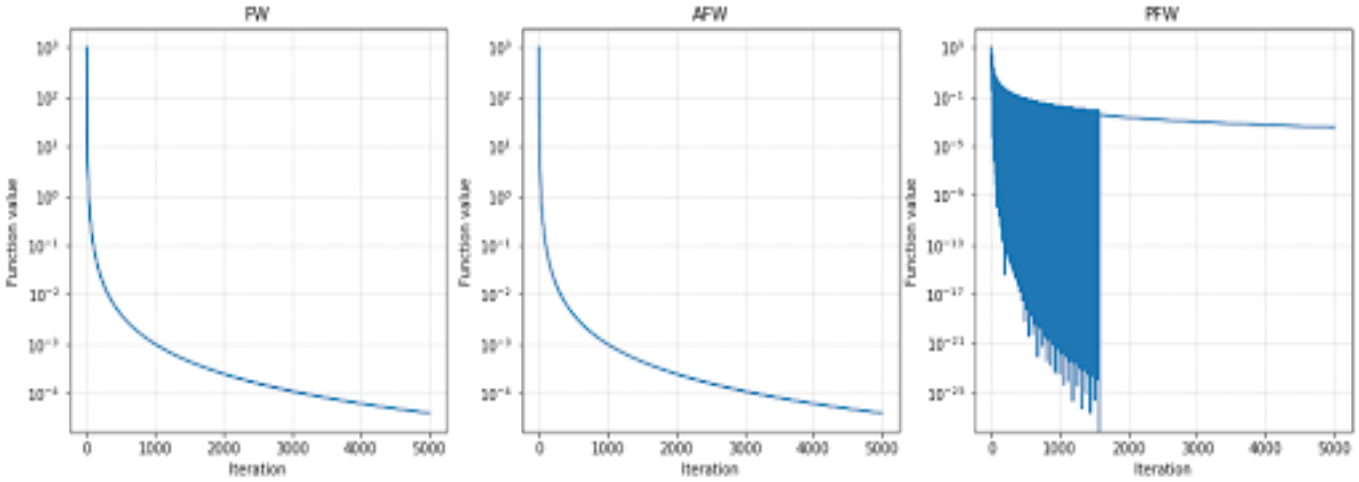


Figure 3. Function value with svmguide1

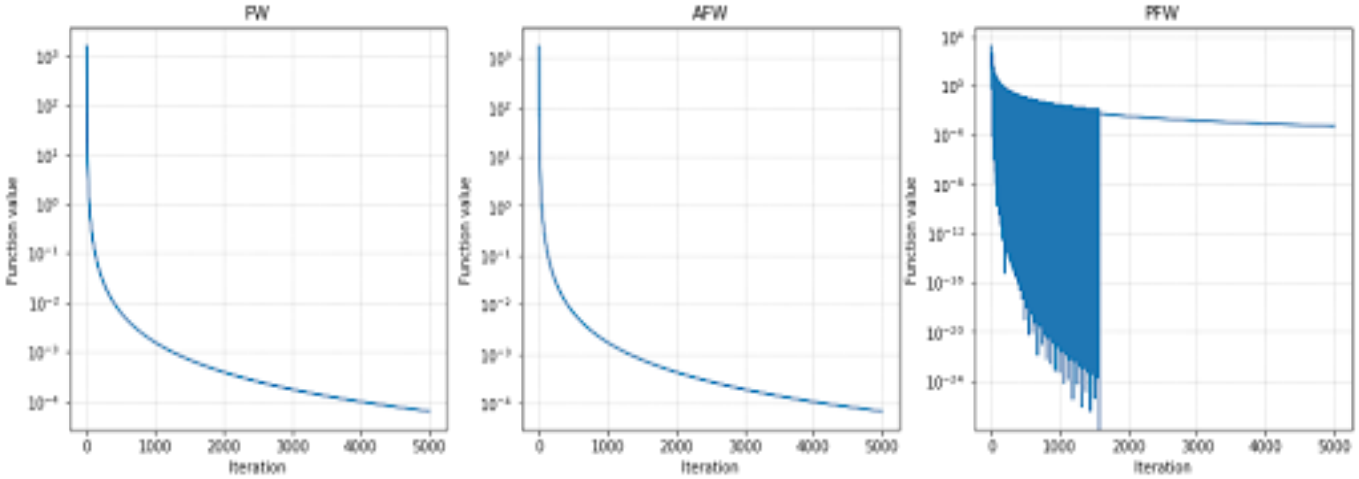


Figure 4. Function value with splice

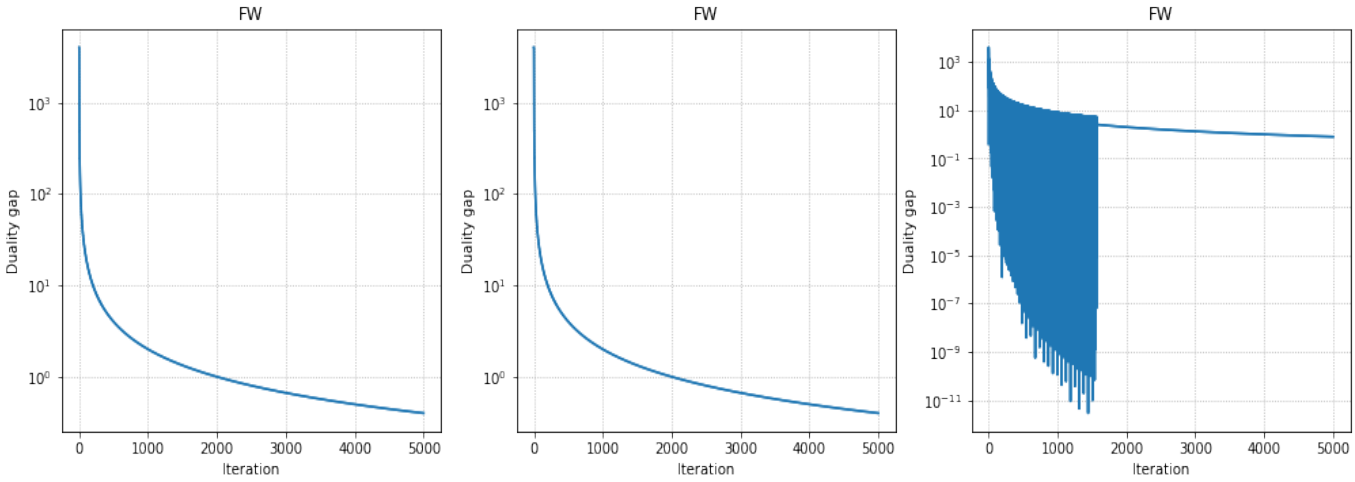


Figure 5. Duality gap with svmguide1

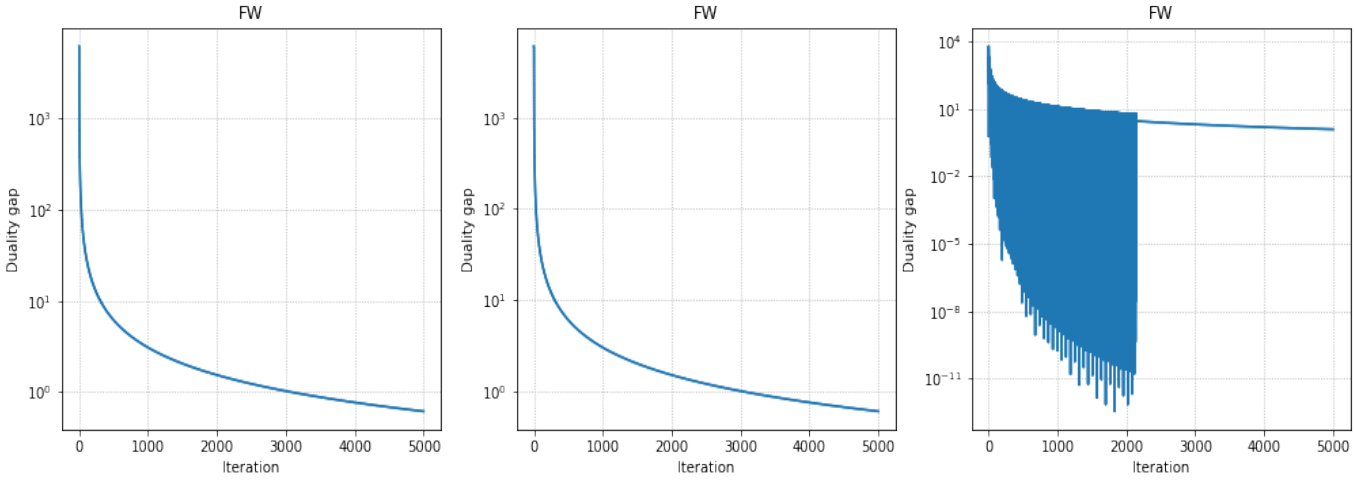


Figure 6. Duality gap with splice

References

[1] M. Jaggi. An equivalence between the lasso and support vector machines. *Eprint Arxiv*, 2013.

[2] Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *International Conference on Machine Learning*, pages 427–435. PMLR, 2013.

[3] Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of frank-wolfe optimization variants. *Advances in neural information processing systems*, 28, 2015.