Optimizing Recommender Systems: A Small Study on the Application of Frank-Wolfe and Away-Step Frank-Wolfe Algorithms

Riccaro Carangelo

Abstract

Recommender Systems are tools playing a crucial role for providing suggestions for items that a user may find useful, in order to assist users in various decision-making processes, enabling individualized user experiences across several areas. The optimization approaches used to enhance these systems frequently confront computational efficiency and scalability issues. This small work delves into the Frank-Wolfe and Away-Step Frank-Wolfe algorithms as reasonable optimization strategies for improving the efficacy of Recommender Systems. The study provides a first theoretical part with the aim of introducing the mathematical basis behind the Frank-Wolfe approach. A second part is dedicated to showing the algorithms built and used, with a performance comparison discussion on the same dataset.

1. Introduction

In recent years, Recommender Systems have become significant tools in several fields, leading to deep changes in the way in which users discover products and services, offering great efficiency in suggestion tasks. On the other hand, Frank-Wolfe algorithms consist in a family of iterative first-order optimization tools specifically thought for constrained convex optimization and originated from a first proposed algorithm, developed in 1956 by Marguerite Frank and Philip Wolfe. The main feature of the Frank-Wolfe algorithm is the exploitation of a linear approximation of the objective function. At each iteration, the algorithm takes into account such linearization and moves towards a minimizer of this linear function.

The objective of this paper is to introduce the Frank-Wolfe algorithm and a specific variant, i.e., the Away-Steps Frank-Wolfe algorithm. The paper starts with a theoretical introduction with the aim of showing the mathematics behind the Frank-Wolfe algorithm and its Away-Steps variant. The intent of this part is to briefly describe and explain the methods. A second, more empirical, section is dedicated to implement the two algorithms in order to test them on a specific recommendation task. This practical step involves providing results of the empirical analysis and a comparison analysis between the two algorithms.

2. Frank-Wolfe Algorithm

The original Frank-Wolfwe algorithm aims to solve a constrained convex optimization problem of the following form:

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

Where $f : \mathbb{R}^n \to \mathbb{R}$ is a convex and continuously differentiable objective function, while the domain $\mathcal{D} \subseteq \mathbb{R}^n$ is a compact convex subset belonging to a given vector space. The original form of the Frank-Wolfe algorithm is shown in Algorithm [1].

At every iteration, the Frank-Wolfe algorithm takes into account a linear subproblem. In particular, it seeks to solve the linear approximation of the objective function f within the boundaries of the feasible set $\mathcal{D}$. Although the Frank-Wolfe algorithm is known for its good scalability and for its ability to handle the constraints when dealing with machine learning applications, one of the most important features of the the Frank-Wolfe algorithm is its ability of keeping the iterates as a convex combination of only a small amount of “atoms” $\boldsymbol{s}$, a fundamental property known as sparsity.

Algorithm 1 Frank-Wolfe Algorithm [1]

```
1: Let $\mathbf{x}^{(0)} \in \mathcal{D}$
2: for $k = 0...K$ do
3:     Compute $\mathbf{s} := \underset{\mathbf{s}\in\mathcal{D}}{\operatorname{argmin}} \left\langle \mathbf{s}, \nabla f(\mathbf{x}^{(k)}) \right\rangle$
4:     Update $\mathbf{x}^{(k+1)} := (1-\gamma)\mathbf{x}^{(k)} + \gamma\mathbf{s}$ where $\gamma := \frac{2}{k+2}$
```

3. Away-Steps Frank-Wolfe Algorithm

The Away-Steps algorithm [2] is a variant of the original Frank-Wolfe algorithm. The latter can show a slow convergence rate in cases in which the solution is at the boundary. In this specific context, the Away-Steps variant can provide a way of solving this problem by adding the possibility for the algorithm to take a step away during the optimization task. In this case

The main idea is that, at each iteration it is possible to add a new atom $\boldsymbol{s}$ or, also, possibly remove an atom that can be bad for the optimization task. Using this variant can improve sparsity and faster solutions ”at-the-boundary” for accounting problems.

Algorithm 2 Away-Steps Frank-Wolfe Algorithm [2]

```
1:  Let $x^{(0)} \in \mathcal{A}$ and $S^{(0)} := \{x^{(0)}\}$
2:  for $t = 0...T$ do
3:      Let $s_t := \text{LMO}_{\mathcal{A}}(\nabla f(\boldsymbol{x}^{(t)}))$ and $\boldsymbol{d}_t^{\text{FW}} := \boldsymbol{s}_t - \boldsymbol{x}^{(t)}$
4:      Let $v_t \in \underset{\boldsymbol{v}\in\mathcal{S}^{(t)}}{\text{argmax}} \left\langle \nabla f(\boldsymbol{x}^{(t)}, \boldsymbol{v}) \right\rangle$ and $\boldsymbol{d}_t^{\text{A}} := \boldsymbol{x}^{(t)} - \boldsymbol{v}_t$
5:      if $g_t^{\text{FW}} := \left\langle -\nabla f(\boldsymbol{x}^{(t)}), \boldsymbol{d}_t^{\text{FW}} \right\rangle \leq \epsilon$ then return $\boldsymbol{x}^{(t)}$
6:      if $\left\langle -\nabla f(\boldsymbol{x}^{(t)}), \boldsymbol{d}_t^{\text{FW}} \right\rangle \geq \left\langle -\nabla f(\boldsymbol{x}^{(t)}), \boldsymbol{d}_t^{\text{A}} \right\rangle$ then
7:          $\boldsymbol{d}_t := \boldsymbol{d}_t^{\text{FW}}$ and $\gamma_{\max} := 1$
8:      else
9:          $\boldsymbol{d}_t := \boldsymbol{d}_t^{\text{A}}$ and $\gamma_{\max} := \frac{\alpha_{v_t}}{(1-\alpha_{v_t})}$
10:         end if
11:     Line-search: $\gamma_t \in \underset{\gamma\in[0,\gamma_{\max}]}{\text{argmin}} f(\boldsymbol{x}^{(t)} - \gamma \boldsymbol{d}_t)$
12:     Update $\boldsymbol{x}^{(t+1)} := \boldsymbol{x}^{(t)} + \gamma_t \boldsymbol{d}_t$
13:     Update $\mathcal{S}^{(t+1)} := \{\boldsymbol{v} \in \mathcal{A} \text{ s.t. } \alpha_v^{(t+1)} > 0\}$
14:     end for
```

4. Methods

In this work the two variants are tested on a recommender system problem. For this purpose, two Python 3 functions (`Frank_Wolfe` and `away_steps_FW`) were built and a pre-processing step was necessary before testing the algorithms

4.1. *The dataset*

The dataset used in this work is the MovieLens dataset (ml-25m, available at https://grouplens.org/datasets/movielens/), in which a CSV table file named "ratings.csv" shows users' and movies' IDs, and a column of ratings in a 5-stars system. The dataset contains 25000095 ratings across 62423 movies, while users were selected at random for inclusion, only considering users with at least 20 rated movies.

4.2. *Pre-Processing*

In order to to provide some further information about the data and to process them for the following Frank-Wolfe algorithms tests, the dataset has been pre-processed. From this phase, it was possible to get a distribution of the ratings (shown in Figure 1).

Also, the dataset contains a total movies count of 59047 and a total users count of 162541, without NaN values. The CSV table organizes the recommendation data in a classical way, showing three columns that are useful for the task: "userId", "movieId", and "rating". The table has been processed to make a pivot table in which user IDs are placed in the first column, while movie IDs are placed in the first row. In this way, a sparse matrix was generated, containing the ratings and several NaN values (converted to zero entries). However, generating a such a sparse matrix from the whole dataset would lead to a non-addressable problem, due to huge size of the pivot table. This is why the decision of pruning the original dataset was made, by just retaining the top 93 % most active users (with a total count of 704 user IDs) and the top 50 % most rated movies (with a total count 654 movie IDs). In this way sparsity has been reduced to 20.88 %.

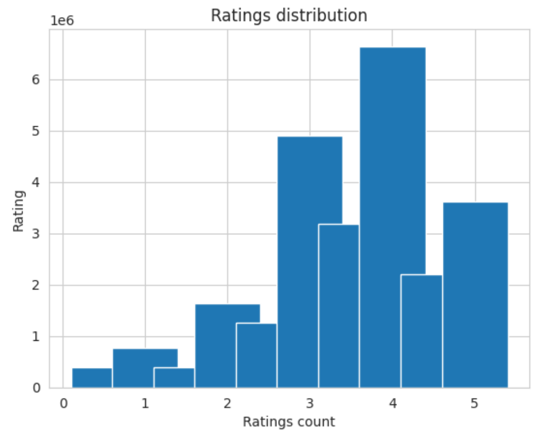

Figure 1: Ratings distribution. Positive ratings constitute a larger component of the total votes.

4.3. *Cope with sparsity*

A first important problem was the sparsity of the matrices used for testing the algorithms. Sparsity is a common problem in almost every recommendation task and it can become a serious problem, because of the calculation time and the risk of corrupting solutions. In order to ensure that null entries were considered as missing data, the SciPy library was used to convert the matrices into CSR format, i.e., a format that detects null entries and only considers the actual data for calculations. In this way, it was possible to efficiently cope with the problem of matrix sparsity, by addressing it both from the point of view of correctness of calculations and of computational time.

4.4. *How about the constraints?*

Frank-Wolfe algorithms are optimization algorithm for constrained optimization. For this problem, a nice set for a suitable constraint is the nuclear norm-ball, i.e., the convex surrogate for the rank constraint, defined as follows:

$$\begin{aligned}\{X \in \mathbb{R}^{m\times n} \; \|\|X\|_{\text{nuc}} \leq 1\} \\ = \text{conv}\{uv^T | u \in \mathbb{R}^m, v \in \mathbb{R}^n, \\ \|u\|_2 = \|v\|_2 = 1\}\end{aligned}$$

Where $\|X\|_{\text{nuc}}$ = is the nulcear norm of the matrix X, given by the some of the singular values of X. A linear minimization in this context only requires a truncated SVD, since:

$$\begin{aligned}\underset{\|X\|_{\text{nuc}}\leq 1}{\text{argmin}} \langle X, Y\rangle &= \underset{\|u\|_2=\|v\|_2=1}{\text{argmax}} \; \text{tr}((uv^T)^T(-Y)) \\ &= \underset{\|u\|_2=\|v\|_2=1}{\text{argmax}} \; u^T(-Y)v \\ &= uv^t\end{aligned}$$

Where u and v are the top left and right singular vectors of $-Y$, respectively. The Singular Value Decomposition (SVD) is a powerful matrix factorization technique which is widely used in recommendation systems, since it allows a dimensionality reduction that uncovers latent factors in user-item interactions. In this case, this particular decomposition is useful for the linear minimization step.

4.5. Algorithm's tests

In this work, f is the RMSE function, defined as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

The two functions `Frank_Wolfe` and `away_steps_FW` takes as inputs the pivot matrix described above, a number of epochs for performing the training, the ball radius for the nuclear norm-ball, the loss function (in this case the RMSE), and the function for calculating the gradient. For both algorithms the outputs are the predicted ratings pivot matrix and a list containing the chronologically sorted gaps, used for making the plots in shown in Figure 2 and in Figure 3. Furtherome, the gaps (used also as a stopping criterion) have been calculated for each epoch, using $g(\boldsymbol{x}) = \langle \boldsymbol{x} - \boldsymbol{s}, \nabla f(\boldsymbol{x})\rangle$.

5. Results

When comparing the two algorithms, the first difference that stands out immediately is in the time required for calculation, since the classical Frank-Wolfe algorithm was, in general, faster and showed a constant speed when training through the epochs, while the Away-Steps variant progressively decreased its speed. The general slowness is due to the extra complexity added in the Away-Steps algorithm, while the Away-Steps' progressive slowness is due to the fact that the active set results more and more populated and big when going through the epochs, slowing down the away direction calculation step progressively.

Figure 2 and in Figure 3 show the gaps of the two algorithms, with, as expected, a better performance for the Away-Steps Frank-Wolfe variant. Although the gap trend of the Away-Steps variant results quite compressed when compared to the gap trend of the original Frank-Wolfe algorithm (Figure 4)), it is easy to observe that the Away-Steps variant outperforms the simple Frank-Wolfe, giving a more stable and smooth trend, which is also lower during all the training phase.

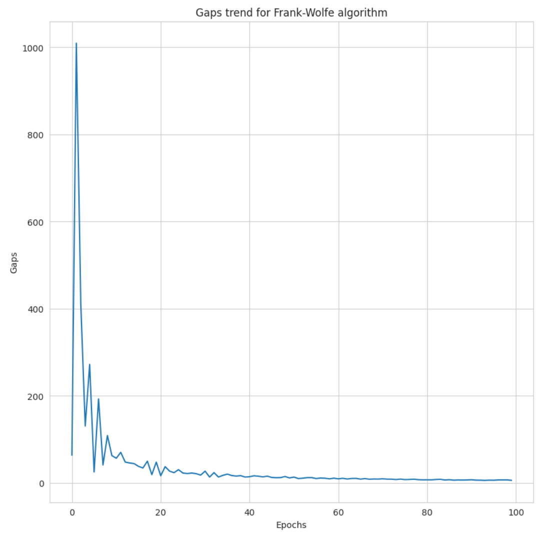

Figure 2: Gap convergence during training of the original Frank-Wolfe algorithm.

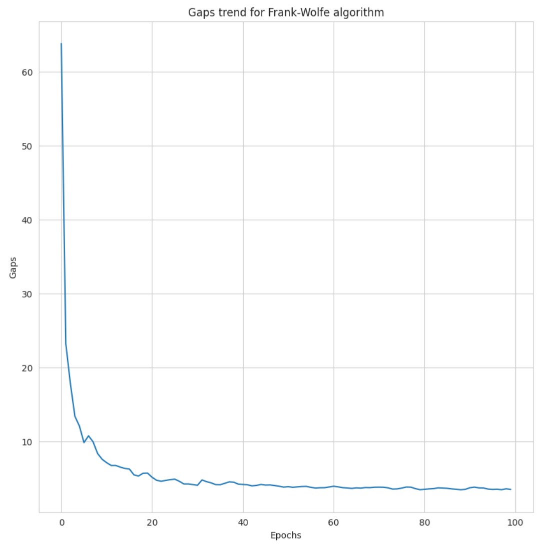

Figure 3: Gap convergence during training of the Away-Steps Frank-Wolfe algorithm.

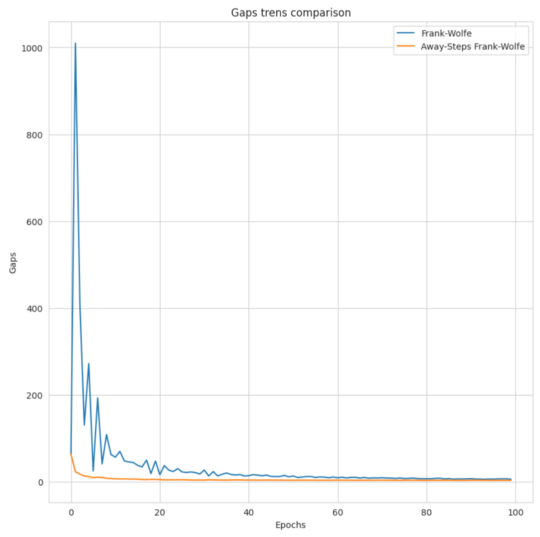

Figure 4: Gaps trends comparison for the two algorithms, Away-Steps Frank-Wolfe and simple Frank-Wolfe. The former outperforms the latter.

6. References

Combettes, C. W., & Pokutta, S. (2021). Complexity of linear minimization and projection on some sets. Operations Research Letters, 49(4), 565-571.

Jaggi, M. (2013, February). Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In International conference on machine learning (pp. 427-435). PMLR.

Lacoste-Julien, S., & Jaggi, M. (2015). On the global linear convergence of Frank-Wolfe optimization variants. Advances in neural information processing systems, 28.

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. ¡https://doi.org/10.1145/2827872¿