Primal-Dual Block Frank-Wolfe Baselines

Samuel Finestone
Department of Computer Science
McGill University
Montreal, QC H3A 0G4
samuel.finestone@mail.mcgill.ca

Cyril Yared
Department of Engineering
McGill University
Montreal, QC H3A 0G4
cyril.yared@mail.mcgill.ca

Eric Sun
Department of Computer Science
McGill University
Montreal, QC H3A 0G4
eric.sun@mail.mcgill.ca

Abstract

For this paper, we reviewed baseline algorithms and run times that were used to benchmark the Primal-Dual Block Frank-Wolfe Algorithm. The new algorithm, sought to reduce per-iteration cost, improving overall convergence, and when combined with Elastic Net Regularization created an algorithm that only depended on the sparsity of the solution set. We re-implemented five different baseline algorithms that were compared to the proposed algorithm, along with a sixth new baseline. We then ran the reproduction tests on four of the six data sets used by the original paper. With two of the data sets, Duke Breast Cancer, and RCV1, we conducted further optimization tests on the hyperparameters of the algorithms. When replicating baselines, we were largely able to confirm relative convergence rates between each respective algorithm. Under optimization tests however, we were able to speed up Accelerated Projected Gradient Descent significantly, making it faster than the paper's proposed Primal-Dual Block Frank-Wolfe algorithm for certain data sets.

1 Introduction

The paper of which we are reproducing baseline statistics for are focused on solving optimization problems of the form:

$$\min_{x \in C} : \sum_i f_i(a_i^\top) + g(x) \tag{1}$$

The authors of our reference paper Lei et al. [1] aim to improve on existing algorithms for these constrained optimization problems by focusing on problems of a special form, those with low-rank or sparse solutions. Specifically they focus on three properties.

1. Exploiting the function finite-sum form and simple structure of the solution.
2. Linear convergence on smooth and strongly convex problems.
3. Low-cost projection steps.

Currently, for large data sets, conventional algorithms will either require expensive matrix computations, or dominating projection step calculations. Frank-Wolfe type algorithms struggle with this as time complexity generally does not decrease asymptotically with the number of iterations, and offer

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

only sub-linear convergence. With the modifications proposed in Primal-Dual Block Frank-Wolfe, Lei et al. managed to utilize the special structure of sparse matrix solutions, reducing previously expensive matrix computations into dimensionless calculations. Additionally, Lei et al. proved that this improvement in complexity meant that for sparse solutions, Primal-Dual Block Frank-Wolfe achieved linear convergence.

We re-implemented, in Python, the five benchmark algorithms of the paper: Frank-Wolfe (FW), Accelerated Projected Gradient Descent (AccPG)[2], Stochastic Variance Reduced Gradient (SVRG)[3], Stochastic Conditional Gradient Sliding (SCGS)[4], and Stochastic Variance-Reduced Conditional Gradient Sliding (STORC)[5]. We also re-implemented the papers new algorithm Primal-Dual Frank-Wolfe (pdFW), due to the differences at runtime of Python and C++.

To extend the results we already had from the published paper, we also implemented a new algorithm: Stochastic Variance-Reduced Frank-Wolfe (SVRF) as an additional baseline to the work done in the paper. This new algorithm was mentioned with STORC and SVRG (Hazan & Luo, 2016) as an additional variation on optimization solvers, we decided that it would be applicable to test it on the sparse solution space problems proposed by Lei et al.

When verifying baselines, we found that our new implementations had trends that closely matched that found by the original paper, however, with further work we were able to optimize the hyperparameters of the algorithms, and achieved a faster convergence for the Accelerated Projected Gradient Descent algorithm on certain data sets.

1.1 Related Works

Constrained optimization problems form a solution structure that can often be applied to many different areas of machine learning. New research has also tackled applying these algorithms to different machine learning techniques (Vieillard, Pietquin, & Geist, 2019)[6], which has looked into using constrained optimization to complement dynamic programming in reinforcement learning models. Another paper published by (Sun, Liu, & Yang, 2019)[7] utilized constrained optimization to solve problems with hard to define objective functions.

2 Data Sets and Setup

The six benchmark algorithms were originally tested on 6 different data sets, each with sparsity ratios ranging from 0.1% to 5.9% (the number of non-zero values in the data set).

- `Duke breast-cancer`: Gene expression in breast cancer prediction. (5.9%)
- `RCV1`: Binary classification of news stories. (2.5%)
- `news20.binary`: Binary text classification. (0.1%)
- `MNIST.RB 0 vs 9`: Modified MNIST to differentiate between handwritten 0's and 9's. (0.9%)
- `ijcnn.RB`: Classifying bipolar output on time series of combustion engines. (1.2%)
- `cod-rna.RB`: Binary classification of RNA samples. (1.2%)

Due to computational constraints, we focused on the `Duke breast-cancer` and `RCV1` data sets. We ran the standard benchmark tests that matched the original paper on four data sets (`Duke breast-cancer`, `RCV1`, `MNIST.RB 0 vs 9`, and `cod-rna`), along with our new algorithm, while we conducted optimization and hyperparameter tests on just the two noted. While the Duke, RCV1, and news20.binary were unchanged from their source, the latter three data sets were modified via random binning to augment features before learning[8][9]. Hence, to ensure the most accurate comparisons for baselines, we utilized the preprocessed data provided by the authors in our experiments as well.

3 Proposed Approach

In this reproduction study, we attempted to reproduce all five baselines including Accelerated Projected Gradient (AccPG), Frank-Wolfe (FW), Stochastic Conditional Gradient Sliding (SCGS),

Stochastic Variance-Reduced Conditional Gradient Sliding (STORC), and Stochastic Variance Reduced Gradient (SVRG) on these data sets. We additionally proposed and implemented a Stochastic Variance Reduced Frank-Wolfe (SVRF) baseline due to its strong performance in Hazan & Luo which outperformed the baselines that were compared to it including STORC, SVRG and SCGS.

During these reproduction studies, the main variable that we studied was the rate of convergence with respect to time as was analyzed by the paper. However, we also chose to additionally investigate the true loss with respect to time due to the fact that as observed in Hazan & Luo, some algorithms may not converge as quickly as others however they may still achieve a lower loss within a shorter period of time.

In order to verify the strength of the findings, in addition to implementing SVRF, we also decided to modify the hyperparameters with a high focus on the learning rates to determine if better results could be achieved. For most of these algorithms, the learning rate tends to be one of the most important hyperparameters as it directly influences the rate of convergence. We modified the baselines for the Duke Breast Cancer data set and the Reuters Corpus Volume I data set since these data sets were smaller than the others and we could thus perform more intensive experiments on them with limited computational resources.

In order to reproduce the baselines of the various optimization algorithms, we first ran the provided code written in C++ to verify their findings. While we could have performed modifications to the hyperparameters and implemented a new baseline in C++, we decided to re-implement them in Python. In this way, we would be able to gain additional insight into how differences in implementation affect performance, especially because Python is an interpreted language compared to a compiled language. Thus, certain instructions may be faster in Python depending on how they are implemented with libraries while other series of computations or loops may run slower. We also re-implemented the paper's proposed Primal-Dual Block Frank-Wolfe algorithm in Python in order to be able to compare it with these baselines. However, we did not modify the hyperparameters of this algorithm.

Finally, as the variables measured are highly dependent on the computational resources, it is important to note that for a single test-run comparing algorithms, all algorithms were run on the same computer with approximately the same environment in order to ensure that the running speed would not be affected by external factors. However, as different test-runs were conducted on various computers, the main aspect observed is not the overall amount of time that it took an algorithm to converge however the relative performance of all of these algorithms.

4 Results

4.1 Reproduction

First, we ran the original C++ code provided by the authors of the paper for the four data sets and we observed similar trends to their findings without much variation except for slight differences in the overall time to convergence which is simply a result of running tests on a different computing platform. After re-implementing the six algorithms including Primal-Dual Block Frank-Wolfe (PDBFW) in addition to SVRF in Python and using the same hyperparameters used in the original study and using a learning rate for SVRF that we initially optimized through simple trial-and-error tactics, we obtained the convergence rate reported in Figure 1, and the losses reported in Figure 2.

4.2 Hyperparameter Tuning

We then attempted to adjust the hyperparameters of all of the baseline algorithms for the Duke Breast Cancer and the Reuters Corpus Volume I data set to attempt to achieve faster convergence times for such models as indicated in Table 1 for the Duke data set and Table 2 for the Reuters data set. Furthermore, for the SVRF algorithm, we changed the parameter from 10, which we had originally set, to 28 for the Duke data set and from 100 to 175 in the Reuters data set. We achieved the results in Figure 3 for both data sets for the rates of convergence and losses.

Because of the fact that tuning the hyperparameters for the some of the algorithms such as AccPG and STORC yielded significantly better results in the Python implementation, we also decided to change the hyperparameters in the published C++ code to determine if this is implementation specific and we obtained the results in Figure 4 for both data sets for the rates of convergence.

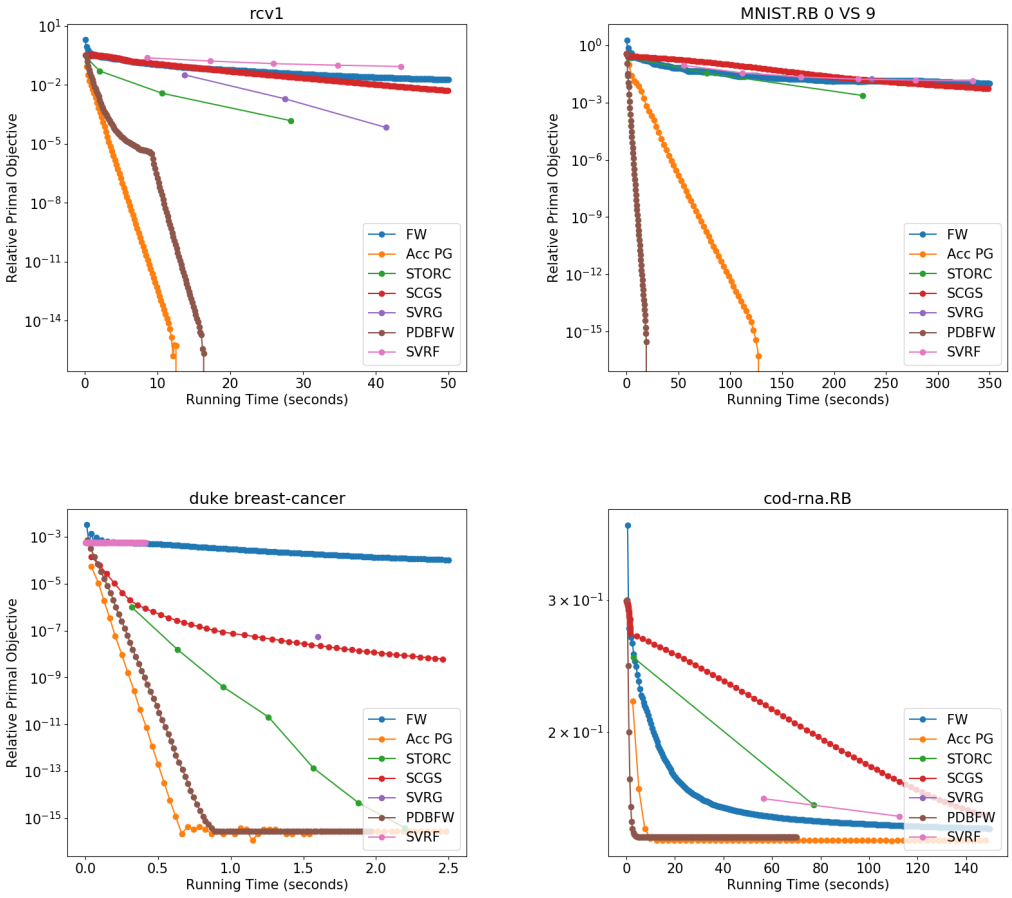

Figure 1: Reproduction of the convergence rates displayed for each baseline, for each data set with the H1 hyperparameters.

Table 1: The hyperparameters before and after tuning for the Duke data set.

Algorithm	H1	H2
AccPG	$\eta = 1.5$	$\eta = 4.5$
FW	$\eta = 0.5$	$\eta = 0.25$
SCGS	L = 0.5	L = 0.2
STORC	L = 0.1	L = 0.205
SVRG	$\eta = 5$	$\eta = 5$

Table 2: The hyperparameters before and after tuning for the RCV data set.

Algorithm	H1	H2
AccPG	$\eta = 100$	$\eta = 115$
FW	$\eta = 0.4$	$\eta = 0.25$
SCGS	L = 10	L = 5
STORC	L = 1	L = 0.75
SVRG	$\eta = 40$	$\eta = 50$

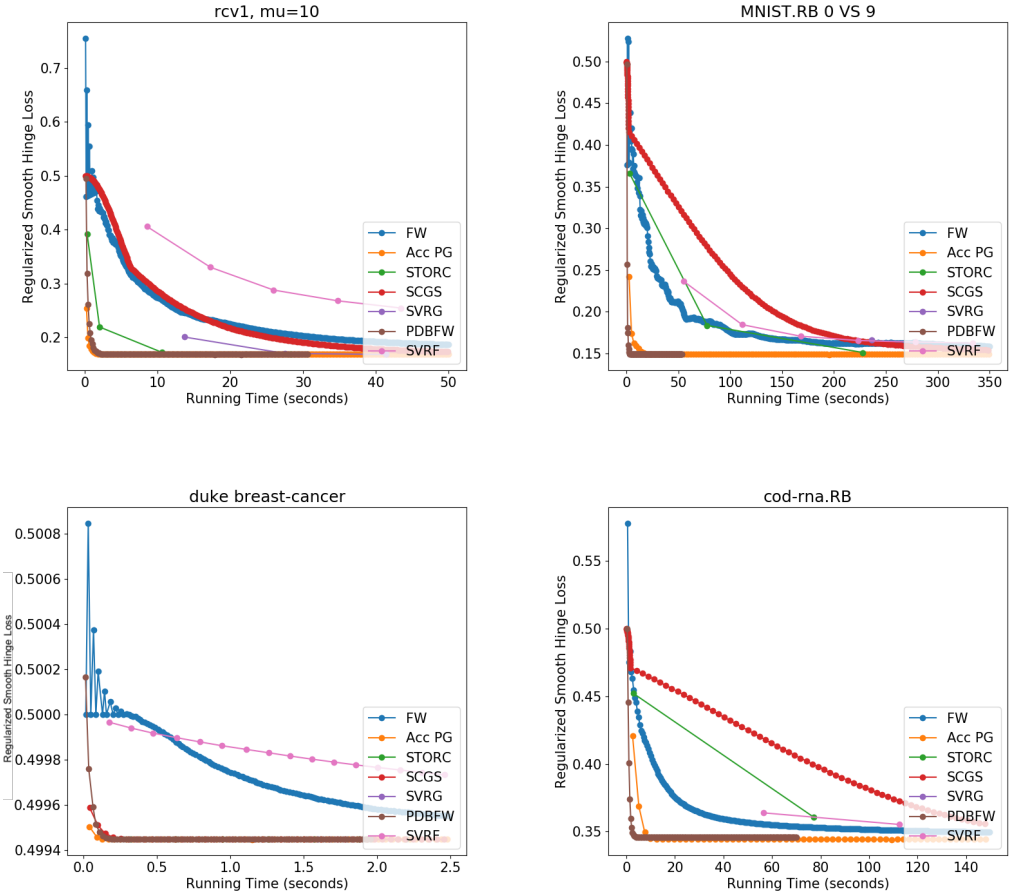

Figure 2: A comparison of the regularized loss function against running time on each data set with the H1 hyperparameters.

5 Discussion and Conclusion

When attaining results in this field, it is important to have rigorous and optimal baselines for evaluation. Comparing achieved results with baselines is an essential step towards understanding relative performance. Although, the variant of the Frank-Wolfe algorithm by Lei et al. reaches better results for larger data sets that are more sparse, our experiments with baselines indicate that further hyperparameter optimization could have been performed for smaller and less sparse data sets. In this paper, we extend the results obtained by Lei et al. by placing more focus on their benchmark comparison. Particularly, we implemented hyperparameter fine-tuning coupled with the additional SVRF benchmark, to effectively evaluate their variant algorithm.

During our exploration, we noted that tuning the learning rates had a bigger impact on the Duke Breast Cancer data than more sparse data sets. A less sparse data set, such as Duke, is more prone to changes in learning rates than sparser data sets, since the updates are generally larger. Consequently, the calculation of the gradient often involves a matrix multiplications between the sparse matrix and the current projected weights leading to potentially larger swings in the gradient itself which is multiplied by the learning rate, directly affecting convergence.

With the PDBFW algorithm, we would expect to see a greater decrease in convergence rate, since their method is founded upon a hinge loss regularization that takes solution sparsity as input rather than all the data points, explaining why it performed better on the larger and more sparse data sets like the MNIST or cod-rna data set. This can be seen in particular when looking at the new20.binary

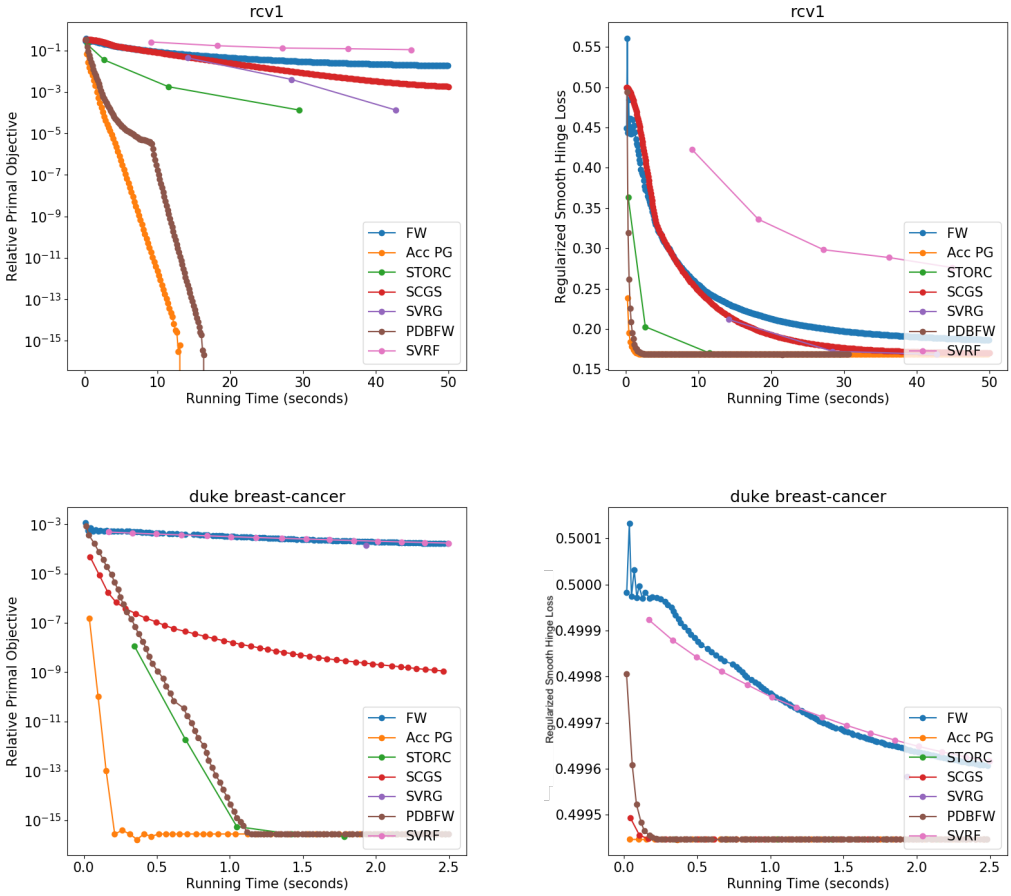

Figure 3: Comparison of the rate of convergence and the loss with each algorithms for both the RCV1 and the Duke data sets after hyperparameter optimization (H2).

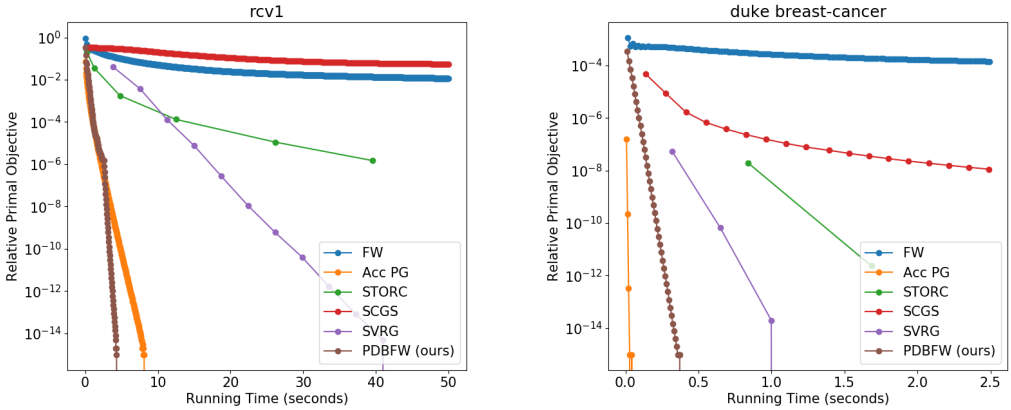

Figure 4: The rates of convergence obtained for the RCV1 and Duke breast-cancer data set with optimization of hyperparameters (H2) for the original C++ code.

graph in Figure 1 in Lei et al., where their algorithm benefits the most from the highest sparsity input and hence why it has the largest margin between itself and the second quickest performing algorithm. Despite our limited resources, we attained very similar results to those in the original paper and these can be seen when looking at Figure 1 in Lei et al. and Figure 1 in this paper. However, we do see a disparity between our optimized baselines and their original benchmark rates of convergence. While their algorithm still converges quickly relative to our optimized benchmarks, it is interesting to note that for the RCV1 data set, the margin between the rates of convergence between the AccPG and PDBFW is much smaller in the C++ implementation and the AccPG actually surpasses the performance of PDBFW in the Python implementation. Furthermore, for the Duke data set, both the Python and C++ implementation of AccPG converges quicker than PDBFW.

Both of these data sets are smaller and have less sparsity than the other two data sets where the PDBFW algorithm outperforms all other baselines. For the smaller and less sparse data sets, the reason why the Python implementation may perform better for AccPG compared to PDBFW is that AccPG relies on one outer loop with several matrix multiplications while PDBFW relies on a nested loop with partial updates. While the C++ compiler may be rather efficient at transforming the nested loop to several instructions, the Python interpreter may be slower when running large sequences in nested loops because every Python instruction must be independently interpreted as a sequence of assembly instructions yielding a higher number of instructions overall. Therefore, different implementations of the algorithms may significantly affect their overall performance on these smaller data sets.

The SVRF baseline taken from the paper by Hazan & Luo, while promising, did not perform as well as other baselines. This is noted particularly on smaller data sets, since it requires more computational cost. The SVRF use nested loops and calls upon the hinge loss function more times than simpler algorithms like AccPG. For larger data sets however, like cod-rna and MNIST, SVRF is a competitive algorithm. The reason that the performance may suffer for smaller data sets is the same reason that PDBFW does not perform as well as AccPG, especially in a Python implementation.

Overall, for large data sets that are sparse, the Primal-Dual Block Frank-Wolfe algorithm proposed by Lei et al. achieves quicker convergence than other algorithms. While we demonstrated that for smaller data sets, algorithms like Accelerated Projected Gradient may perform better, the performance for such data sets heavily depends on how the algorithm is implemented. We were able to successfully reproduce the performance of all of the baselines tested in the original paper on four of the data sets and we found some opportunity for tuning of hyperparameters for the smaller data sets that improved the performance of some of the baselines.

5.1 Further Work

Given more time and computational power, we could have performed hyperparameter optimization for the larger and more sparse data sets and conduct more rigorous testing to gain a more holistic view of their performance. Lei et al. investigates two additional data sets that are larger and sparser than the four that we experimented with. While we expect PDBFW to perform significantly better than all of the baseline algorithms for such data sets, there is an opportunity to attempt to optimize the hyperparameters of the baselines to compare the relative performance.

Additionally, we could try different implementations of the algorithms and attempt to further optimize the code in Python for PDBFW. As discussed above, the nested for-loops in Python yielded worse performance for the algorithm than that achieved in C++. Perhaps, by using lambdas or similar features in Python, we could improve the performance of PDBFW for smaller data sets.

6 Statement of Contributions

Cyril worked on the AccPG.py, SCGS.py, and main.py files as well as the DataLoader.py file. He also ran all the algorithms for the graphical comparisons and optimized the individual learning rates. Samuel worked on implementing FW.py, STORC,py and the PDBFW.py scripts. Finally, Eric wrote the code for both utilities.py, SVRG.py, and SVRF.py. We all worked together on writing different sections of the report.

References

[1] Qi Lei, Jiacheng Zhuo, Constantine Caramanis, Inderjit S. Dhillon, and Alexandros G. Dimakis. Primal-dual block frank-wolfe. *CoRR*, abs/1906.02436, 2019.

[2] Yurii E. Nesterov. *Introductory Lectures on Convex Optimization - A Basic Course*, volume 87 of *Applied Optimization*. Springer, 2004.

[3] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 315–323, 2013.

[4] Guanghui Lan and Yi Zhou. Conditional gradient sliding for convex optimization. *SIAM Journal on Optimization*, 26(2):1379–1409, 2016.

[5] Elad Hazan and Haipeng Luo. Variance-reduced and projection-free stochastic optimization. *CoRR*, abs/1602.02101, 2016.

[6] Nino Vieillard, Olivier Pietquin, and Matthieu Geist. On connections between constrained optimization and reinforcement learning. *CoRR*, abs/1910.08476, 2019.

[7] Chengjian Sun, Dong Liu, and Chenyang Yang. Model-free unsupervised learning for optimization problems with constraints. *CoRR*, abs/1907.12706, 2019.

[8] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM TIST*, 2(3):27:1–27:27, 2011.

[9] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1177–1184, 2007.