Complexity Issues and Randomization Strategies in Frank-Wolfe Algorithms for Machine Learning

Emanuele Frandi ^{1*}
ESAT-STADIUS
KU Leuven, Belgium

Ricardo Ñanculef ^{2*}
Department of Informatics
Federico Santa María University, Chile

Johan Suykens ^{1*}
ESAT-STADIUS
KU Leuven, Belgium

Abstract

Frank-Wolfe algorithms for convex minimization have recently gained considerable attention from the Optimization and Machine Learning communities, as their properties make them a suitable choice in a variety of applications. However, as each iteration requires to optimize a linear model, a clever implementation is crucial to make such algorithms viable on large-scale datasets. For this purpose, approximation strategies based on a random sampling have been proposed by several researchers. In this work, we perform an experimental study on the effectiveness of these techniques, analyze possible alternatives and provide some guidelines based on our results.

1 Introduction

The Frank-Wolfe algorithm [7], hereafter denoted as FW, is a general method to solve

$$\min_{\alpha \in \Sigma} \; f(\alpha),$$

where $f : \mathbb{R}^m \to \mathbb{R}$ is a convex differentiable function, and $\Sigma \subset \mathbb{R}^m$ is a convex polytope. Given the current iterate $\alpha^{(k)} \in \Sigma$, a standard FW iteration consists of the following steps:

1. Define a search direction $d^{(k)}$ by optimizing a linear model:
$$u^{(k)} \in \operatorname*{argmin}_{u \,\in\, \Sigma} \, (u - \alpha^{(k)})^T \nabla f(\alpha^{(k)}) = \operatorname*{argmin}_{u \,\in\, \mathcal{V}(\Sigma)} u^T \nabla f(\alpha^{(k)}), \;\; d^{(k)} = u^{(k)} - \alpha^{(k)}, \quad (1)$$
where $\mathcal{V}(\Sigma)$ denotes the set of vertices of Σ.
2. Choose a stepsize $\lambda^{(k)}$, e.g. by a line-search: $\lambda^{(k)} \in \operatorname{argmin}_{\lambda \,\in\, [0,1]} f(\alpha^{(k)} + \lambda d^{(k)})$.
3. Update: $\alpha^{(k+1)} = \alpha^{(k)} + \lambda^{(k)} d^{(k)} = (1 - \lambda^{(k)})\alpha^{(k)} + \lambda^{(k)} u^{(k)}$.

Recently, the Optimization and Machine Learning communities have showed a renewed surge of interest in the family of FW algorithms [10, 9, 16]. They enjoy bounds on the number of iterations which are independent of the problem size, as well as sparsity guarantees [3, 10]. Furthermore, variants of the above basic procedure exist which attain a linear convergence rate [19, 8, 16, 13]. Such properties make FW a good choice for problems arising in a variety of applications [1, 5, 14].

Complexity of Frank-Wolfe Iterations. As the total number of FW iterations can be large in practice, devising a convenient way to find a solution to the subproblem (1) is often mandatory in order to make the algorithm viable. A typical situation arises when (1) has an analytical solution or the problem structure makes it easy to solve [16, 15]. Still, the resulting complexity can be impractical when handling large-scale data. As a motivating example, we consider the problem

$$\min_{\alpha \,\in\, \mathbb{R}^m} \;\; f(\alpha) = \tfrac{1}{2}\alpha^T K \alpha \quad \text{ s.t. } \sum_{i=1}^{m} \alpha_i = 1, \;\; \alpha \geq 0 \; , \qquad (2)$$

[*] Emails: ^{1}`{efrandi,johan.suykens}@esat.kuleuven.be`, ^{2}`jnancu@inf.utfsm.cl`

which stems from the task of training a nonlinear L_2-SVM model for binary classification [18, 4]. Here, K is a positive definite kernel matrix. In this case, $\mathcal{V}(\Sigma) = \{e_1, \ldots, e_m\}$, hence we have

$$u^{(k)} = e_{i_*^{(k)}}, \qquad \text{where} \qquad i_*^{(k)} \in \operatorname*{argmin}_{i=1,\ldots,m} \nabla f(\alpha^{(k)})_i = \operatorname*{argmin}_{i=1,\ldots,m} \sum_{j \,|\, \alpha_j^{(k)} > 0} K_{i,j} \alpha_j^{(k)} .$$

The theoretical cost of an iteration is therefore $\mathcal{O}(m|\mathcal{I}^{(k)}|)$, where $\mathcal{I}^{(k)} = \{i \,|\, \alpha_i^{(k)} > 0\}$, proportional to the number of examples.[1] In order to circumvent the dependence from the dataset size, the use of approximation strategies based on a random sampling has been proposed by several researchers [18, 5], but, up to our knowledge, never systematically studied on practical problems. We attempt to fill this gap by performing an experimental study on the effect of using such techniques.

2 Randomization Strategies and Possible Alternatives

In this section, we consider two different techniques to reduce the computational effort in each FW iteration, and try to identify the kind of problems where each can be applied effectively.

2.1 Random Working Set Selection

A simple and yet effective way to avoid the dependence on m is to explore only a fixed number of points in $\mathcal{V}(\Sigma)$. In the case of (2), this means extracting a sample $\mathcal{S} \subseteq \{1, \ldots, m\}$ and solving

$$i_{\mathcal{S}}^{(k)} \in \operatorname*{argmin}_{i \in \mathcal{S}} \nabla f(\alpha^{(k)})_i .$$

The iteration cost becomes in this case $\mathcal{O}(|\mathcal{S}||\mathcal{I}^{(k)}|)$. The following result motivates this kind of approximation, suggesting that it is reasonable to keep the samples very small, i.e. to pick $|\mathcal{S}| \ll m$.

Theorem 1 ([17], Theorem 6.33)**.** *Let $\mathcal{D} \subset \mathbb{R}$ be a set of cardinality m, and let $\mathcal{D}' \subset \mathcal{D}$ be a random subset of size r. Then, the probability that the smallest element in $\mathcal{D}'$ is less than or equal to $\tilde{m}$ elements of $\mathcal{D}$ is at least $1 - (\frac{\tilde{m}}{m})^r$.*

In the case of (2), where $\mathcal{D} = \{\nabla f(\alpha^{(k)})_1, \ldots, \nabla f(\alpha^{(k)})_m\}$ and $\mathcal{D}' = \{\nabla f(\alpha^{(k)})_i \,|\, i \in \mathcal{S}\}$, this means that, for example, it only takes $|\mathcal{S}| \approx 60$ to guarantee that, with probability at least 0.95 (and independently of m), $\nabla f(\alpha^{(k)})_{i_{\mathcal{S}}^{(k)}}$ lies between the 5% smallest gradient components.

Choice of the Stopping Criterion and Implications. The stopping criterion for FW algorithms is usually based on the duality gap [10]:

$$\Delta_d(\alpha^{(k)}) := \max_{u \in \Sigma} (\alpha^{(k)} - u)^T \nabla f(\alpha^{(k)}) \overset{(2)}{=} 2f(\alpha^{(k)}) - \nabla f(\alpha^{(k)})_{i_*^{(k)}} \leq \varepsilon .$$

This criterion, however, is not applicable without computing the entire gradient $\nabla f(\alpha^{(k)})$, which is not done in the randomized case. As a possible alternative, we can use the approximate quantity

$$\Delta_{\mathcal{S}}(\alpha^{(k)}) := 2f(\alpha^{(k)}) - \nabla f(\alpha^{(k)})_{i_{\mathcal{S}}^{(k)}} .$$

Since $\Delta_{\mathcal{S}}(\alpha^{(k)}) \leq \Delta_d(\alpha^{(k)})$, this simplification entails a tradeoff between the reduction in computational cost and risk of an anticipated stopping. Although this can be considered acceptable in contexts such as SVM classification, where solving the optimization problem with a high accuracy is usually not needed, it is important to make sure that the impact of this approximation can be kept to an acceptable level. The experiments in the next section aim precisely at investigating this issue.

2.2 Analytical Gradient Update

Another possibility to obtain a more efficient iteration is to exploit the structure of the problem to keep the exact gradient $\nabla f(\alpha^{(k)})$ updated at each iteration [12]. In the case of problem (2), this can be done in $\mathcal{O}(m)$ operations, since it is easy to see by using the formula for the FW step that

$$\nabla f(\alpha^{(k+1)})_i = (1 - \lambda^{(k)}) \nabla f(\alpha^{(k)})_i + \lambda^{(k)} K_{i, i_*^{(k)}} , \qquad i = 1, \ldots, m.$$

Compared to a naive implementation, we get rid of a factor $|\mathcal{I}^{(k)}|$ and, as an important by-product, we have that the duality gap can be updated exactly without any additional cost.

[1] More in general, it is proportional to $|\mathcal{V}(\Sigma)|$ and to the cost of computing $u^T \nabla f(\alpha^{(k)})$, with $u \in \mathcal{V}(\Sigma)$.

3 Numerical Results

In order to assess the effectiveness of the above implementations of the FW step, we conducted numerical tests on the benchmark datasets **Adult a9a** ($m = 32561$), **Web w8a** ($m = 49749$), **IJCNN** ($m = 49990$) and **USPS-ext** ($m = 266079$) [2, 6]. All the experiments were coded in C++, and executed on a 3.40GHz 4-core Intel machine with 16GB RAM running Linux.

Table 1 presents the statistics (averaged over 10 runs) for classification accuracy on the test set, CPU time, number of iterations and support vectors, obtained with samplings of increasing size. The tolerance parameter was set to $\varepsilon = 10^{-4}$, and a Gaussian kernel was used in all the experiments. An LRU caching strategy was implemented to avoid the computation of recently used entries of K [11].

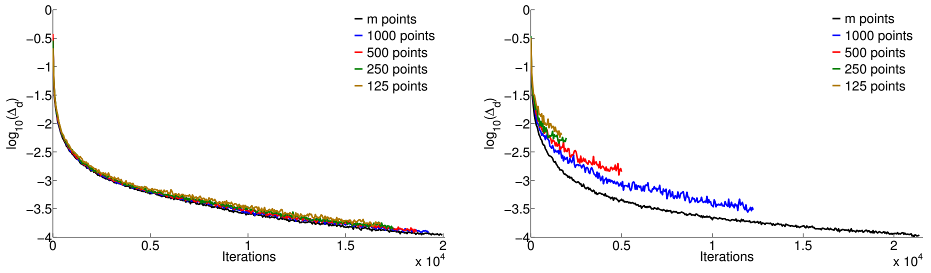
Dataset		m points	1000 points	500 points	250 points	125 points
Adult a9a	Test acc (%)	83.56	84.10	83.91	83.68	83.88
	Time (s)	$1.40e+02$	$4.26e+02$	$2.55e+02$	$1.62e+02$	$1.12e+02$
	Iter	$2.02e+04$	$1.94e+04$	$1.91e+04$	$1.85e+04$	$1.71e+04$
	SVs	$1.40e+04$	$1.37e+04$	$1.36e+04$	$1.34e+04$	$1.20e+04$
Web w8a	Test acc (%)	99.36	99.30	99.28	99.00	98.49
	Time (s)	$3.17e+02$	$2.50e+02$	$1.60e+02$	$5.55e+01$	$2.75e+01$
	Iter	$1.65e+04$	$1.39e+04$	$1.24e+04$	$4.63e+03$	$2.17e+03$
	SVs	$6.43e+03$	$6.33e+03$	$5.77e+03$	$2.82e+03$	$1.70e+03$
IJCNN	Test acc (%)	98.24	98.42	98.30	98.28	97.57
	Time (s)	$4.99e+01$	$1.19e+02$	$5.80e+01$	$3.12e+01$	$1.43e+01$
	Iter	$1.61e+04$	$1.46e+04$	$1.22e+04$	$9.91e+03$	$5.69e+03$
	SVs	$3.17e+03$	$3.59e+03$	$3.72e+03$	$3.84e+03$	$3.37e+03$
USPS-ext	Test acc (%)	99.52	98.90	98.88	99.50	99.45
	Time (s)	$1.77e+03$	$4.25e+02$	$2.83e+02$	$1.56e+02$	$4.97e+01$
	Iter	$2.05e+04$	$9.07e+03$	$4.32e+03$	$2.70e+03$	$1.65e+03$
	SVs	$3.94e+03$	$3.59e+03$	$3.00e+03$	$2.37e+03$	$1.60e+03$

Table 1: Average statistics with different sampling sizes.

First of all, note that the effect of sampling is substantially problem-dependent. On some datasets, such as **USPS-ext**, FW clearly encounters an early stopping even with a fairly large sampling size, while other results, such as those on **Adult a9a**, appear more stable. In some cases, e.g. on **Web w8a**, there seems to be a cutoff point after which the performance degrades considerably. Still, some general trends can be estabilished: the number of iterations decreases monotonically with $|\mathcal{S}|$, as expected from the observations in Section 2, and CPU times decrease accordingly. On the contrary, as seen from the results on **IJCNN**, the model size is not always monotonic with respect to $|\mathcal{S}|$. This arguably happens because solving (1) approximately can lead to spurious points being selected as FW vertices. Finally, note that the full sampling solution (which employs the strategy in Section 2.2) is very competitive on the smaller problems, while it is still very time consuming on the largest dataset **USPS-ext**. This intuitively suggests that a random sampling is computationally convenient when it can still produce a good solution with $|\mathcal{S}| \ll m/\mu_{|\mathcal{I}^{(k)}|}$, where $\mu_{|\mathcal{I}^{(k)}|}$ is an estimate of the average cardinality of $\mathcal{I}^{(k)}$ across iterations. Some of these conclusions are summarized in Table 2.

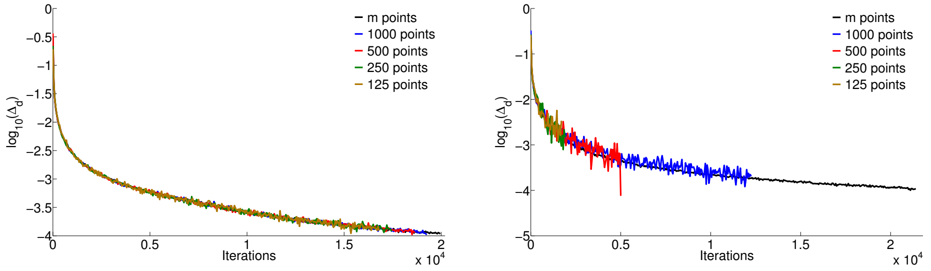
In the next experiment, we analyze, on the datasets **Adult a9a** and **USPS-ext**, the effect of sampling on the computation of the duality gap (and therefore on the stopping criterion) and on the minimization of the linear model. Figures 1 and 2 report, respectively, the exact gap Δ_d and the approximate gap $\Delta_{\mathcal{S}}$, plotted in logarithmic scale against the iteration number for various sampling sizes.

The figures shed light on the results in Table 1. On the dataset **Adult a9a**, the randomized strategy appears very effective: the duality gap does not deviate much from the ideal figure obtained with the full dataset, even for small sampling sizes. Furthermore, there are no significant differences between computing the exact and approximate duality gap. On the other hand, on **USPS-ext**, Δ_d is noticeably larger than its approximate counterpart, indicating that the algorithm is making less progress than predicted by $\Delta_{\mathcal{S}}$. Furthermore, the approximate gap exhibits large oscillations due to the random nature of the sampling, and it is possible that an "unlucky" iteration leads to a premature stopping, as can be seen from the figure. It is interesting to note that the degradation in optimization quality



(a) (b)

Figure 1: Exact duality gap path on datasets **Adult a9a** (a) and **USPS-ext** (b).



(a) (b)

Figure 2: Approximate duality gap path on datasets **Adult a9a** (a) and **USPS-ext** (b).

(as measured by Δ_d) is not reflected in this case by a corresponding loss in test accuracy, which is a phenomenon typical of classification problems. However, this is not true in general, as other applications such as function estimation are known to be more sensitive to a less accurate solution.

Randomized Working Set Selection	- Applicable whenever Σ is a polytope - Large computational gain when $\lvert\mathcal{S}\rvert \ll m/\mu_{\lvert\mathcal{I}^{(k)}\rvert}$ - Performance depends on the problem
Analytical Gradient Update	- Convenient for structured f (e.g. quadratic) - Saves a factor $\lvert\mathcal{I}^{(k)}\rvert$ at each iteration - Deterministic results

Table 2: Some recommendations on the implementation of the FW step.

Adaptive Strategies. Taking into account all the above, one would ideally want to be able to select an optimal strategy automatically, based on the data and the actual performance. Provided both strategies can be applied to the problem at hand, one could for example start by performing a fixed number $\bar{k} > 0$ of iterations using both, and then devise some criterion based on the difference in duality gap to decide whether the approximation is adequate. However, a discussion on how to effectively implement such a strategy would be nontrivial, and as such is deferred to a separate work.

4 Conclusions

Using SVM classification problems as a motivation, we have performed an experimental study on the effectiveness and impact of some techniques designed to alleviate the computational burden of the optimization step in a FW iteration. Our results suggested that, while it comes with some caveats, a random sampling technique may be the most viable choice on very large-scale problems. On the other hand, when the problem size is not prohibitive (e.g. batch training tasks with medium to large datasets), fast updating schemes which exploit the problem structure might provide a better choice.

Acknowledgments

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC AdG A-DATADRIVE-B (290923). This paper reflects only the authors' views and the Union is not liable for any use that may be made of the contained information. Research Council KUL: GOA/10/09 MaNet, CoE PFV/10/002 (OPTEC), BIL12/11T; Flemish Government: FWO: projects: G.0377.12 (Structured systems), G.088114N (Tensor based data similarity); PhD/Postdoc grants; iMinds Medical Information Technologies SBO 2014; IWT: POM II SBO 100031; Belgian Federal Science Policy Office: IUAP P7/19 (DYSCO, Dynamical systems, control and optimization, 2012-2017).

References

[1] A. Argyriou, M. Signoretto, and J. A. K. Suykens. Hybrid algorithms with applications to sparse and low rank regularization. In J. A. K. Suykens, M. Signoretto, and A. Argyriou, editors, *Regularization, Optimization, Kernels, and Support Vector Machines*, chapter 3. Chapman & Hall/CRC (Boca Raton, USA), 2014.

[2] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2011.

[3] K. Clarkson. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. *ACM Transactions on Algorithms*, 6(4):63:1–63:30, 2010.

[4] E. Frandi, M. G. Gasparo, S. Lodi, R. Ñanculef, and C. Sartori. A new algorithm for training SVMs using approximate minimal enclosing balls. In *Proceedings of the 15th Iberoamerican Congress on Pattern Recognition, Lecture Notes in Computer Science*, pages 87–95. Springer, 2010.

[5] E. Frandi, M. G. Gasparo, S. Lodi, R. Ñanculef, and C. Sartori. Training support vector machines using Frank-Wolfe methods. *International Journal of Pattern Recognition and Artificial Intelligence*, 27(3), 2011.

[6] A. Frank and A. Asuncion. *The UCI KDD Archive. http://kdd.ics.uci.edu*, 2010.

[7] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 1:95–110, 1956.

[8] J. Guélat and P. Marcotte. Some comments on Wolfe's "away step". *Mathematical Programming*, 35:110–119, 1986.

[9] Z. Harchaoui, A. Juditski, and A. Nemirovski. Conditional gradient algorithms for norm-regularized smooth convex optimization. *Mathematical Programming*, 13(1):1–38, 2014.

[10] M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

[11] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. *IEEE Transactions on Neural Networks*, 11(1):124–136, 2000.

[12] P. Kumar and A. Yildirim. A linearly convergent linear-time first-order algorithm for support vector classification with a core set result. *INFORMS Journal on Computing*, 23(3):377–391, 2011.

[13] S. Lacoste-Julien and M. Jaggi. An affine invariant linear convergence analysis for Frank-Wolfe algorithms. *arXiv.org*, December 2013.

[14] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate Frank-Wolfe optimization for structural SVMs. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

[15] G. Liuzzi and F. Rinaldi. Solving l_0-penalized problems with simple constraints via the Frank-Wolfe reduced dimension method. *Optimization Letters (in press)*, 2014.

[16] R. Ñanculef, E. Frandi, C. Sartori, and H. Allende. A novel Frank-Wolfe algorithm. analysis and applications to large-scale SVM training. *Information Sciences (in press)*, 2014.

[17] B. Schölkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, USA, 2001.

[18] I. Tsang, J. Kwok, and P.-M. Cheung. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

[19] P. Wolfe. Convergence theory in nonlinear programming. In J. Abadie, editor, *Integer and Nonlinear Programming*, pages 1–36. North-Holland, Amsterdam, 1970.