Applying Frank-Wolfe based algorithms for Lasso and Latent Group Lasso regularizations

Francesca de Robertis, Gabriele Etta

September 2020

Abstract

In this essay, we propose a study about the family of Frank-Wolfe (FW) algorithms and their applications in the case of sparse convex optimization. To accomplish this, we focus on two important regularization penalties called Lasso and Latent Group Lasso. The term Lasso, commonly referred as L_1 penalty, is a mathematical tool adopted in the case of sparse regression analysis when it comes to regularize and perform variable selection. We describe its features and limitations, especially in the cases where groups of predictors are meant to be kept together. We then consider some alternatives that overcome this problem, focusing on the *Latent Group Lasso*. We conclude by studying the theoretical properties of the FW algorithms on a series of regression problems regularized by the two penalties proposed, comparing the results on artificial and real data scenarios.

1 Introduction

In the context of statistics and machine learning, the notion of sparsity has been central for a plenty of problems. Intuitively, sparsity is associated with the idea of having mathematical structures for which the number of zero elements is way larger than those that have a non-zero value. For example, a sparse matrix is a special type of matrix which contains most of the elements equal to zero or, equivalently, a sparse statistical model is a model which is represented at its best by a small number of predictors. The requirement of having a reduced number of elements that can still represent the original structure they belong has been useful for several applications. For example, in the *compressive sensing problem* we focus on obtaining the lowest number of coefficients for our unknown vector x i.e., the reconstruction of an input signal through an approximation obtained by a linear combination of elementary ones, under the case of *overcompleteness*. A solution to this kind of problem is to regularize the objective function with the l_1 norm, commonly

known as Lasso penalty [3] in order to obtain a convex problem which takes the name of *basis pursuit* [4], easier to solve with respect to the minimization of the initial support function $\min |supp(x)|$. From the first definition of Lasso regularization, the necessity of selecting groups of predictors together has raised out, bringing the concept of sparsity to another level such that [5] proposed the *Group lasso* regularization. The introduction of this penalty was a first step to manage structural sparsity but it led to situations for which, if some predictors of a group were not equal to zero, the entire group will not be zero as well. In the case of groups that share the same predictors, i.e., overlapping groups, this results in a problem for which sparsity is not guaranteed anymore. To overcome this, the concept of *Latent Group Lasso* [9] was introduced, moving the focus on the application of its norm to variables that are a decomposition of the original vectors.
The different sparsity problems that we have introduced have always been an object of study for the mathematical optimization theory, especially in the context of constrained programming for which the family of Frank Wolfe algorithms [1] were reported to be particularly outperforming. However, Frank Wolfe has shown some problems when dealing with large scale domains. To overcome this, some variants of this algorithm have been introduced in which the computation of the whole gradient is not needed. These are called *randomized variants* because they randomly sample some atoms leading to a reduction of the computational burden. In particular, in this work two randomized variants have been studied: Randomized Frank Wolfe (RFW) which is related to the classical Frank Wolfe algorithm and, in order to improve the convergence rate, the Randomized Away-step Frank Wolfe (RAFW).
The essay is organized as follows. In Section 2, we introduce the current state of the art for the topics that we analyze. Section 3 introduces the theory behind the concepts introduced in a more formal way in order to provide a foundation about the minimization problems considered and the algorithms chosen. Section 4 focuses on the description of the properties of the algorithms implemented. Section 5 applies the notions from the previous sections by running experiments in different contexts, comparing the results from some other works used as benchmark. Finally, Section 6 discusses the results obtained and the future works that may be carried.

2 Related Work

Given the purpose of this essay, in this section we highlight all the papers that served as an important foundation or benchmark for our work. We start by introducing a paper from [16] which was fundamental since our essay tries to replicate their results, and at the same time it is used as a proxy to discover the other related works. In fact, we referred [9] and [8]

for the basic notions about the different group lasso regularizations. In particular, the first work was really important since it introduced the concept of latent group lasso for the first time, for which we were able to extract meaningful theoretical insights that were brought into our experiments. From an algorithmic perspective, the different description of the Frank Wolfe and its variants provided by [10], [15], [13] were crucial in the development of the algorithms and in the convergence analysis.

3 Problem definition

In the regression framework, we are generally using linear models of the form

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon, \tag{1}$$

where Y is called response variable, and $X_1, \ldots, X_p$ are the set of our p dependent variables or predictors. Typically, the standard fitting procedure involves the usage of Least Square Estimates (LSE), however the trade-off between the interpretability of the model and its accuracy may lead us to introduce additional elements, such as the subset selection, the dimension reduction or, in our case, the regularization. By this term, we refer to those approaches that still fits a least squares model on its entire, i.e, by including all the p predictors. The difference arises about how coefficients are managed: the regularization introduces a shrinkage of those estimates towards or exactly to zero, depending on how the parameters of the applied regularization are set. This results in a lower variance obtained by the model, since we can obtain a model with less variables than their original formulation and/or with a series of values that are less predominant. For these reasons, shrinkage is also a method which performs variable selection.
In the literature, regularization is often associated with three methods: *Ridge Regression*, *Lasso* and *Elastic Net*. Despite there is an interconnection between them, we choose to only describe the Lasso since it is the foundation of the different methods that we encountered during the development of this project.

3.1 The Lasso Regression

In order to estimate the coefficients β_0 and β to achieve the most accurate approximation for the original response variable, we can choose among different methods. For the sake of simplicity and in order to be coherent with the way Lasso regression is usually introduced in the literature [11, 14], we use the so called *least square method* to fit the model, whose goal is to reduce the squared error between the real and the estimated values by minimizing the square-error loss on the coefficients β_0 and β:

$$\min_{\beta_0,\beta} \left\{ \frac{1}{2n} \|y - \beta_0 - \boldsymbol{X}\beta\|_2^2 \right\} \tag{2}$$

As explained in Section 3, this formulation may be subjected to the requirements of model interpretability and prediction accuracy. Those requirements lead to the extension of Eq.2 with the addition of a further constraint, called l_1-norm penalty, which introduces a new type of problem called the *Lasso* problem. With the same setting used for the LSE, we obtain the following formulation:

$$\min_{\beta_0,\beta} \left\{ \frac{1}{2n} \|y - \beta_0 - \boldsymbol{X}\beta\|_2^2 \right\} \quad \text{subject to} \quad \|\beta\|_1 \leq t. \tag{3}$$

As we can see, the minimization problem did not change on its core. We can only observe the addition of the l_1-norm norm constraint such that $\|\beta\|_1 \leq t$.

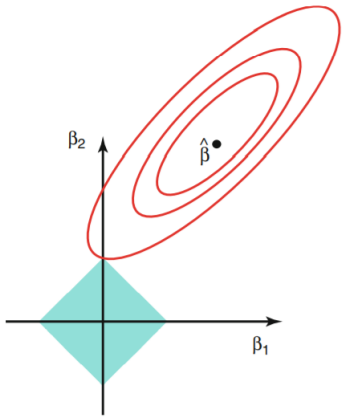


Figure 1: Graphical representation of the Lasso problem.

The presence of this norm strongly changed the estimation process of the parameters in our model. Fig.1 gives a 2D representation of what introducing the l_1 penalty means. In fact, the red ellipses centered around the LSE solution $\hat{\beta}$ represent the different combinations of values in β that shares the same RSS, whilst the sharp area in blue is our constraint $\|\beta\|_1 \leq t$. Given a reasonable value for t, the intersection between the ellipses and one of the vertices of the diamond brings the solution of the problem. In fact, unlike the Ridge estimator, which is represented with a circle, the sharp nature of the Lasso estimator will shrink one of the parameter value to zero, depending on the position where the intersection happened. For this reason, Lasso regression also performs variables selection, which makes it an important tool when it is required to obtain sparse solution.

For notational purposes, it is important to notice that we can refer to the lasso problem

by using its Langrangian representation, which is the following:

$$\min_{\beta_0,\beta} \left\{ \frac{1}{2\mathrm{n}} \|y - \boldsymbol{X}\beta\|_2^2 + \lambda \|\beta\|_1 \right\}. \tag{4}$$

In this form, we have also made some assumptions as in [14]. In fact, we assume that the predictors are standardized, so each column is centered, i.e., $\frac{1}{n}\sum_{i=1}^{n} x_{i,j} = 0$, with a unit variance, i.e., $\frac{1}{n}\sum_{i=1}^{n} x_{i,j}^2 = 1$. In addition to this, we also have that the dependent variable y has been centered, i.e., $\frac{1}{N}\sum_{i=1}^{n} y_i = 0$, which means that we can omit the intercept term β_0 in Eq.4.

3.2 Group Lasso and Sparse Group Lasso

Lasso regression is an important tool for regularization and variable selection, due to the properties previously explained. Despite this, there are cases for which a regression problem may come up with a certain structure on its features, which have to be selected together in order to maintain their significance. For the way that Lasso works, this is no guaranteed. In fact, the variables selection that it provides performs only at individual level, without looking at any particular structure of the given features. For this reason, [5] proposed a solution which takes the name of *Group Lasso*, which solves the LSE regression problem

$$\min_{\beta \in \mathbb{R}^p} \left\{ \left\| y - \sum_{l=1}^{L} X_l \beta_l \right\|_2^2 + \lambda \sum_{l=1}^{L} \sqrt{p_l} \|\beta_l\|_2 \right\}, \tag{5}$$

where L indicates the number of groups of features, X_l, β_l refer to the single feature and the parameter vector for each group, $\|\cdot\|_2$ is l_2 norm, λ the tuning parameter used to balance the trade-off between the group sparsity of the solution and the minimization of the loss function. $\sqrt{p_l}$ is a term that accounts for the size of groups which may vary according to different factors. The group Lasso penalty is also called l_1/l_2 penalty because it is defined as the sum (i.e. l_1 norm) of the l_2 norms of the restrictions for the parameters vectors in the model, related to the different groups of covariates. This formulation can be rewritten in a more general form:

$$\min_{\beta \in \mathbb{R}^p} \left\{ L(\beta) + \lambda \sum_{l=1}^{L} \sqrt{p_l} \|\beta_l\|_2 \right\} \tag{6}$$

where the least squares term is substituted with a general loss function $L : \mathbb{R}^p \to \mathbb{R}$. Fig.2, shows an example of comparison of the Lasso and Group Lasso regularization.

Despite this penalty gives the opportunity to select groups of variables together, it has to pay a trade-off for which it loses "sensitivity" on the individual level.

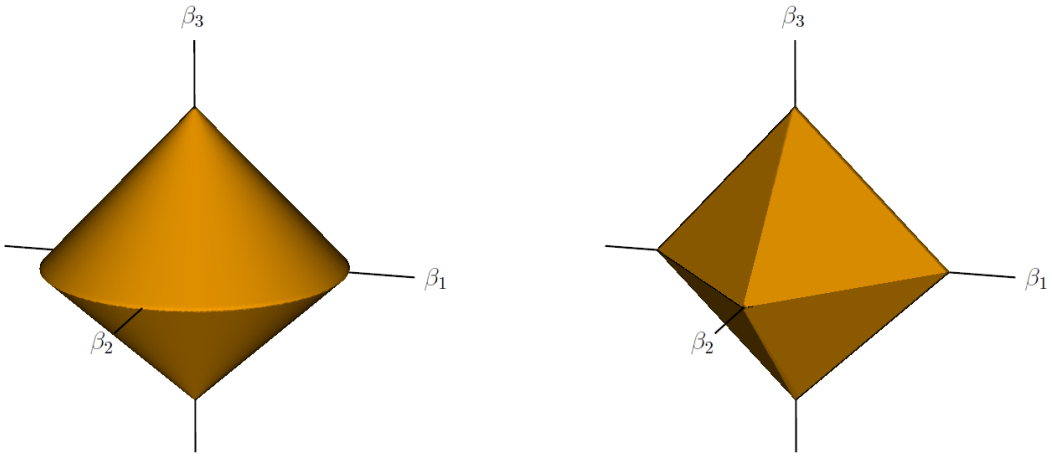


Figure 2: The group lasso ball (left) in $\mathbb{R}^3$, where the group is (β_1, β_2), compared to the L1 ball (right).

The group lasso does not account for the sparsity within a group. This limitation has been fixed by the introduction of a new penalty which is recognized under the name of *Sparse Group Lasso* (SGL) which solves a different minimization problem:

$$\min_{\beta \in \mathbb{R}^p} \left\{ \left\| y - \sum_{l=1}^{L} X_l \beta_l \right\|_2^2 + \lambda_1 \sum_{l=1}^{L} \|\beta_l\|_2 + \lambda_2 \|\beta\|_1 \right\} \tag{7}$$

with this formulation, we are able to have sparsity *between* and *within* the groups.

3.3 Latent Group Lasso

As previously introduced, the Group Lasso estimator defined in Eq.6 presents some problems when it comes to manage groups of overlapping covariates. The reason can be found on its mechanism. In fact, the goal of the Group Lasso is to shrink all the elements of a group to 0 and not to select them, as it could seem to be. It is easy to understand that in the presence of overlapping covariates, this modality may lead to set all the parameters of a group to zero even if some of them overlap with another group, which can introduce a sort of snowballing effect for which other groups may not be chosen as a consequence of this.

The introduction of the concept of *structured sparsity*, was therefore generalized in order to support the cases for which groups of covariates overlap for an arbitrary number of elements. This generalization results in a new penalty called *Latent Group Lasso* (LGL) [9].

3.3.1 Definition

Given a vector $w \in \mathbb{R}^p$ with a support $supp(w) \subset [1, p]$, i.e., the set of covariates $i \in [1, p]$ such that $w_i \neq 0$, we define a group of covariates as $g \in [1, p]$ and $\mathcal{G}$ the set of all groups, such that $g \in \mathcal{G}$. The notion of group can be applied to the vector $w \in \mathbb{R}^p$, obtaining $w_g \in \mathbb{R}^p$, which is a vector whose entries are shared with w only for the indices belonging to the group g and 0 otherwise. To avoid this sparsity, we will work with a set of vectors $\mathcal{V}_\mathcal{G} \subset \mathbb{R}^{p \times \mathcal{G}}$, where each vector $(v_g)_{g \in \mathcal{G}}$ has a support $supp(v_g) \subset g$, which means that its composed of only components related to the group indices that are not equal to 0. Since we will have to show the computation of the entire gradient and the one with each group, we refer to the first with $\nabla f(w) \in \mathbb{R}^p$ and $\nabla_g f(w) \in \mathbb{R}^g$, i.e., the subgradient of f related to the features in the group g.
We recall that the Latent Group Lasso regularization is obtained from the Group Lasso explained in Section 3.2. In fact, in the case of overlapping, w can be decomposed as $w = \sum_{g \in \mathcal{G}_1} v_g$, where $\mathcal{G}_1 \subset \mathcal{G}$ is the set of groups g for which $v_g \neq 0$. If w can be decomposed in that way, it also means that it would not have any coefficient equal to 0, so we can say that its support is composed by the indices of G_1, i.e., $supp(w) \subset \bigcup_{g \in \mathcal{G}_1} g$. This is enough to introduce the formulation of the Latent Group Lasso penalty:

$$\Omega^\mathcal{G}_\cup := \min_{v \in \mathcal{V}_\mathcal{G}} \sum_{g \in G} \|v_g\| \quad \text{s.t.} \boldsymbol{w} = \sum_{g \in G} v_g.$$

From this definition, we can now describe the way we built the structure of the algorithms for our experiments. First, we consider the following optimization problem which is referred in the literature [9] as an empirical risk optimization problem:

$$\min_{w \in \mathbb{R}^p} R(w) + \lambda \Omega^\mathcal{G}_\cup \tag{8}$$

where $R(w) = \frac{1}{2}\|Aw - b\|_2^2$ and therefore this becomes Latent Group Lasso Regression problem, for which we have to find the solutions of the problem:

$$\min_{w \in \mathbb{R}^p, \bar{v} \in \mathcal{V}_\mathcal{G}} \left\{ L(w) + \lambda \sum_{g \in \mathcal{G}} d_g \|v_g\|_2, \quad s.t. \quad w = \sum_{g \in \mathcal{G}} v_g \right\}. \tag{9}$$

The application of the group lasso norm on the latent decomposition has the effect to shrink some v_g to 0, while the others who have a different values satisfy $supp(v_g) = g$, i.e., the overlapping variables can be associated to a group. Therefore, if we define $\mathcal{G}_1 \subset \mathcal{G}$ the set of groups for which $\hat{v}_g \neq 0$, we can say that our solution $\hat{w} = \sum_{g \in \mathcal{G}_1} \hat{v}_g$, obtaining:

$$supp(\hat{w}) = \bigcup_{g \in \mathcal{G}_1} g. \tag{10}$$

We can further reformulate the second part of Eq.9 in order to define the *Latent Group Lasso* penalty as stated in [9]. Since the regularization involves only the penalty term and the constraints, we can treat it like a typical minimization problem, which gives us the formulation of the *Latent Group Lasso Penalization Problem*:

$$\min_{w \in \mathbb{R}^p} L(w) + \lambda \Omega^{\mathcal{G}}_{\cup}(w) \tag{11}$$

where the new term $\Omega^{\mathcal{G}}_{\cup}(w)$ is defined as:

$$\Omega^{\mathcal{G}}_{\cup}(w) = \min_{\bar{v} \in \mathbb{V}_{\mathcal{G}}, \sum_{g \in \mathcal{G}} v_g = w} \left(\sum_{g \in \mathcal{G}} d_g \left\| v_g \right\|_2, \right). \tag{12}$$

With this formulation, when the groups do not overlap and form a partition, we have a unique decomposition of $w \in \mathbb{R}^p$ as $w = \sum_{g \in \mathcal{G}} v_g$ with $supp(v_g) \subset g$. Therefore, we have that $v_g = w_g, \quad \forall g \in \mathcal{G}$, with no distinction between group lasso and latent group lasso penalties. In summary, it ensures that groups can be shrunk to zero and the parameters w of the covariates within can be set to zero if and only if all the group where it belongs are shrunk to zero, with the result of a support of w which is the union of the groups as expressed in Eq.10. Fig.3 summarizes what we have explained so far: from the example introduced in [9], for which we have $\mathcal{G} = \{\{1, 2\}, \{2, 3\}\}$ and consequently $\mathbb{R}^3$ unit balls, we can see how singularities in the Group Lasso ball (left figure) appear only when w_1 or w_2 is non zero, whilst in the Latent Group Lasso ball (middle figure), the singularities happen when only w_1 or only w_3 are zero.

From Fig.3 we can see that the unit ball of Ω is just the convex hull of a horizontal disk and a vertical one. It is possible to formalize this impression:

Lemma 3.3.1. *For any group $g \in \mathcal{G}$, define the hyperdisks $\mathbb{D}_g = \{v \in \mathbb{R}^p | v = v_g, \|v_g\| \leq \beta\}$. Then, the unit ball of Ω is the convex hull $\mathcal{A}$ of the union of hyper-disks $\cup_{g \in \mathcal{G}} \mathbb{D}_g$.*

There are several algorithmic approaches to solve Eq.11 depending on the structure of the groups. In [9] has been used an approach based on the reformulation by *covariate duplication* and applies an algorithm for the group Lasso in the space of duplicates, e.g., the block coordinate descent. Alternatively, there are efficient algorithms which do not require any work in the space of duplicated covariates, for example algorithms from the multiple kernel learning literature. In the Appendix (7), the reader can find additional information

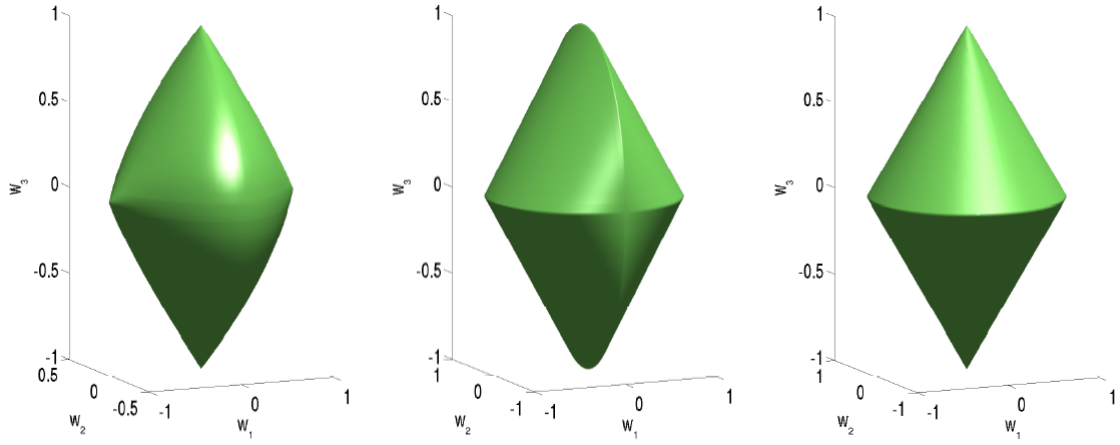


Figure 3: An example Group lasso with overlaps (left), Latent group Lasso (middle) and Group lasso without overlaps (right).

about the LGL, more precisely some properties of the LGL norm and a discussion about the choice of the groups weights that we have not use in our experiments.

4 Algorithms

In this section, we describe all the optimization algorithms used for the development of this project. Following [16] and [15], all the presented algorithms belong to the family of the Frank-Wolfe method [1], in the context of constrained optimization in which the goal is finding a solution of problems of the form:

$$\min_{x \in \mathcal{M}} f(x). \tag{13}$$

In Eq.13, $f : \mathbb{R}^p \to \mathbb{R}$ is a convex function with Lipschitz continuous gradient having constant $L > 0$ and $\mathcal{M} = conv(\mathcal{A})$ is a convex compact set, with a finite diameter

$$D = \max_{x,y \in M} \|x - y\|_2.$$

Since the work [9] that we analyzed brings theoretical concepts such as the duality gap or the application of FW algorithms for structured feasible sets, we decided to structure this section as follows. For the sake of completeness, we start by introducing the Frank-Wolfe algorithm on its general form. Then, we introduce one concept at time which will be useful to build the complete versions of the algorithms used in the work. In order to avoid

redundancy, the algorithms that will follow after the FW will have the previous notions already incorporated.

4.1 Frank-Wolfe Algorithm

The Frank-Wolfe method, also called *conditional gradient method*, is an iterative first-order optimization algorithm which has seen a recent interest due to its low memory requirement and projection-free iterations, which makes it a valid alternative of the projected gradient descent algorithm. The general schema is showed in Algorithm 1.

Algorithm 1 VANILLA FRANK-WOLFE ALGORITHM

Input: $\boldsymbol{x}_1 \in \mathcal{M}$

1: **for** $t = 1, ..., T$ **do**
2: $\quad \boldsymbol{s}_t = \min_{s \in \mathcal{A}} \nabla f(\boldsymbol{x_t})^T (\boldsymbol{s} - \boldsymbol{x_t})$
3: $\quad$ If $\boldsymbol{s_t}$ satisfies some specific condition, then STOP
4: $\quad$ Set $\boldsymbol{x_{t+1}} = \boldsymbol{x_t} + \gamma_t(\boldsymbol{s_t} - \boldsymbol{x_t})$, with $\gamma_t \in (0, 1]$ suitably chosen stepsize .

We start by choosing an initial point x_1 which is a feasible solution. At each iteration, we solve the linear minimization problem at Line 2 of the Algorithm 1 which, due the compactness of $\mathcal{M}$, gives us the solution $s_t \in \mathcal{A}$. Then, we check if the solution of the linear minimization satisfies some specific condition, which in the context of our work is referred to as the value of the dual gap, i.e., $-(\nabla f(x_t)^T (s_t - x_t)) < \epsilon$, where ϵ is a fixed small value. If it is satisfied, we stop and retrieve the last x_t, otherwise we can calculate the new point $x_{t+1} = x_t + \gamma_k d_t$ through the descent direction $d_t = s_t - x_t$ that we have obtained.
The first extension that we apply to the vanilla schema of the FW algorithm is the introduction of the *dual gap*. It relies on the definition, for each $x \in \mathcal{M}$, of the dual function

$$w(x) = \min_{z \in \mathcal{M}} f(x) + \nabla f(x)^T (z - x), \tag{14}$$

for which the minimum is guaranteed since $\mathcal{M}$ is compact and the linear function is continuous in z. Next, we can take advantage of a result from theory for which $w(x) \leq f(y) \quad \forall x, y \in \mathcal{M}$, which brings us to the definition of the duality gap:

$$g(x) = f(x) - w(x) = \max_{z \in \mathcal{M}} \nabla f(x)^T (x - z) = - \min_{z \in \mathcal{M}} \nabla f(x)^T (z - x). \tag{15}$$

Since x^* is unknown, we used this duality gap as our stopping criterion for our algorithms.

The second change on the algorithm affects the seeking of the linear solution s_t. In order to be coherent with many related works [16, 15, 10], we define the *Linear Minimization Oracle* (LMO) function as:

$$LMO(x_t, \mathcal{A}) = \arg\min_{s \in \mathcal{A}} \langle s, \nabla f(x_t) \rangle \tag{16}$$

The application of these algorithms in the context of the l_1 penalty reformulates the definition of the domain of our problem given by the feasible set $\mathcal{M}$. In the context of the l_1 ball, the following definition occurs:

$$\mathcal{M} = \{x \in \mathbb{R}^n : \|x\|_1 \leq 1\} = conv(\{\pm e_i, \quad i = 1, \ldots, n\}). \tag{17}$$

The formulation of this new polytope with a number of vertices equal to $2n$, since it must be symmetrical with respect to the axis involved, is illustrated in Figure 4. In addition to this, the linear minimization problem on this set mutates as well. The solution is now given by:

$$s_t = sign(-\nabla_{ik} f(x_k)) \cdot e_{ik} \tag{18}$$

with $i_k = \arg\max_i |\nabla_i f(x_k)|$, which establishes the new cost of this operation, $\mathcal{O}(n)$.

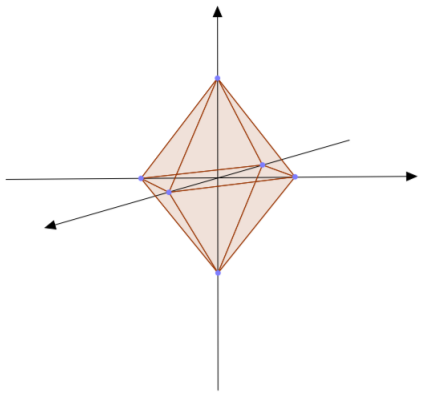


Figure 4: A 3D l_1-ball with a number of vertices equal to 6.

The last change in our algorithm, in order to make it comparable for our experiments, affects the choice of the stepsize. [16] proposes two variants: the first solves an exact line search problem and it is effective in the case of quadratic loss function, such as ours. The second one, instead, estimates the curvature constant C_f which, in the case is not possible,

relies on the bound $C_f \leq D^2 L$, where L is the Lipschitz constant of $\nabla f(x_t)$ and D is the diameter of our feasible set previously described. We decided to opt for the first because the other variants in that paper were reported with the stepsize computed only by the exact line search.
By including all the additions together, we obtain the Frank-Wolfe schema used in our experiments, which is also the foundation of the other variants. The schema is showed in Algorithm 2.

Algorithm 2 Applied Frank-Wolfe algorithm

Input: $\boldsymbol{x}_1 \in \mathcal{M}$
1: **for** $t = 1, ..., T$ **do**
2: **Let** $s_t = LMO_{\mathcal{A}}(\nabla f(x_t))$
3: **Compute the FW direction** $d_t = s_t - x_t$
4: **Compute the gap** $g = -\langle \nabla(x_t), d_t \rangle$
5:
6: **if** $\boldsymbol{g \leq \epsilon}$ **then return** x_t
7: **Perform Line-search:** $\gamma_t \in \arg\min_{\gamma \in [0,1]} f(x_t + \gamma d_t)$
8: **Set** $\boldsymbol{x_{t+1}} = \boldsymbol{x_t} + \gamma_t(\boldsymbol{s_t} - \boldsymbol{x_t})$

Regarding the convergence of the Frank-Wolfe algorithm, it has been proven that with a stepsize $\gamma_t = \frac{2}{t+1}$ theorem 4.1.1 holds:

Theorem 4.1.1. *Let f be a convex function with Lipschitz continuous gradient having constant $L>0$. The Frank-Wolfe method with stepsize $\gamma_t = \frac{2}{t+2}$ satisfies the following inequality:*

$$f(x_{t+1}) - f(x^*) \leq \frac{2LD^2}{t+1} \tag{19}$$

for all $t > 1$ and being D the diameter of our feasible set.

This sub-linear rate can be improved making stronger assumptions on the problem, such as feasible sets with special structure, a function that is σ-strongly convex or optimal solution in the interior.

4.2 Randomized Frank-Wolfe

The first variant that we introduce was proposed by [16] under the name of Randomized Frank-Wolfe (RFW). It distinguishes from the classic FW formulation for two aspects:

- The atomic set from which the LMO is computed is not the entire $\mathcal{A}$ anymore. Instead we rely on a random subset $\mathcal{A}_t \subseteq \mathcal{A}$ in which each atom can appear with the same probability, i.e., $P(v \in \mathcal{A}_t) = \eta$ for all $v \in \mathcal{A}$. This new parameter $\eta \in (0, 1]$

controls the size of the random domain considered by the LMO at each iteration, which tends to be equal to the one used by FW as η goes towards 1.

- The introduction of this new subsampling procedure does not always guarantee a descent direction. Therefore, the choice of the stepsize is not straightforward, but since we have already introduced the possible alternatives in the previous section, the choice of computing the stepsize through an exact-line search function remains feasible.

From a performance perspective, in the context of sparse domains such as the l_1, l_1/l_2 and $\Omega^{\mathcal{G}}_{\cup}$ balls, the computation of this randomized LMO reduces the gradient coordinates that have to be evaluated at each iteration.

As a final note, the computation of the gap in this case loses the property of being an upper bound of the primal error $f(x_t) - f(x^*)$. That is again correlated with the notion of subsampling, which affects the way s_t is obtained and, consequently, the computation of the gap $\langle -\nabla f(x_t), s_t - x_t \rangle$. To overcome this aspect, [16] proposes to compute a full LMO every $k\lfloor \frac{1}{\eta} \rfloor$ iterations, with $k \in \mathbb{N}^*$.

With these new features included, the schema is showed in Algorithm 3.

Algorithm 3 Randomized Frank-Wolfe algorithm

Input: $\boldsymbol{x}_1 \in \mathcal{M}$, sampling ratio $0 < \eta \leq 1$.

for $t = 1, \ldots$ **do**
  Choose $\mathcal{A}_t$ such that $\mathcal{P}(\boldsymbol{v} \in \mathcal{A}_t) = \eta$ for all $\boldsymbol{v} \in \mathcal{A}$
  Compute the random LMO $\boldsymbol{s}_t = \mathrm{LMO}_{\mathcal{A}_t}(\nabla f(\boldsymbol{x}_t))$
  Compute the RFW direction $d_t = s_t - x_t$
  if $t \mod k\lfloor \frac{1}{\eta} \rfloor = 0$ **then**
    Let $s_t = LMO_{\mathcal{A}}(\nabla f(x_t))$
    Compute the FW direction $d_t = s_t - x_t$
    Compute the gap $g = -\langle \nabla(x_t), d_t \rangle$

    if $\boldsymbol{g} \leq \boldsymbol{\epsilon}$ **then return** x_t
  Compute the exact line search $\gamma_t = \arg\max_{\gamma \in [0,1]} f((1-\gamma_t)\boldsymbol{x}_t + \gamma_t \boldsymbol{s}_t)$
  $\boldsymbol{x}_{t+1} = (1-\gamma_t)\boldsymbol{x}_t + \gamma_t \boldsymbol{s}_t$

It has been proven in [16] that RFW has a sub-linear rate of convergence, in particular Theorem 4.2.1 holds.

Theorem 4.2.1. *Let f be a function with bounded smoothness constant C_f and subsampling parameter $\eta \in (0, 1]$. Then Algorithm 3 (in both variants) converges towards a solution of*

problem 13. Furthermore, the following inequality is satisfied:

$$E[h(x_T)] \leq \frac{2(C_f + f(x_0) - f(x^*))}{\eta T + 2} \tag{20}$$

where E is a full expectation over all randomness until iteration T and the curvature constant is defined as:

$$C_f = \sup_{x,s \in \mathcal{A}, \gamma \in [0,1], y = x + \gamma(s-x)} \frac{2}{\gamma^2}(f(y) - f(x)\langle \nabla f(x), y - x \rangle)$$

.

We can conclude that both the randomized and the non-randomized FW variants achieves the same convergence rate. This is due to the fact that the cost of the LMO is reduced by the subsampling rate but is compensate by the larger number of iteration required by this new algorithm.

4.3 Away Step Frank-Wolfe (AFW)

The original proposal of the Frank-Wolfe algorithm established a behavior strictly correlated with the position of the optimal solution. In fact, if x^* lies on the boundary of the convex hull: the more we get closer to it, the more orthogonal the directions become with respect to the gradient. This causes the so called *zig-zagging phenomenon*, slowing the convergence rate. In order to address this problem, a variant of the original FW algorithm is proposed, called *Away-Step Frank-Wolfe* (AFW). The major change that was introduced is the following: at each iteration we compute a new direction called the away-step direction d_t^A, which is the difference between the current vertex x_t and v_t that maximizes the descent. In this way we can compare the two gaps, the original one and the one obtained from the away-step procedure, so that we can choose the direction according to the one that produced the lowest gap. This modification makes the algorithm linearly convergent for strongly convex functions and $\mathcal{M}$ a polyhedral. The result is showed in Algorithm 4.

As illustrated in the algorithm, the AFW carries another atom set, S_t, which contains all elements whose coefficients are positive. This implies a double search at each iteration: the first looks for s_t, the classical FW vertex, whilst the second finds the vertex v_t previously described. Despite this, the search for the latter is typically small, since it is computed over S_t and not in the entire domain. This means that the cost per operation is not $\mathcal{O}(n)$ as the LMO. Furthermore, the algorithm has not a fixed direction anymore, which now depends on the comparison between the FW and the AFW gaps, picking the best one. This also reflects the choice of the maximum stepsize, because especially in the AFW case we want to

Algorithm 4 Away-Step Frank Wolfe (AFW)

Input $\boldsymbol{x}_1 \in \mathcal{M}$, $\boldsymbol{x}_1 = \sum_{\boldsymbol{v}\in\mathcal{A}} \alpha^1_{\boldsymbol{v}}$ and $\mathcal{S}_1 := \{x_1\}$

for $t = 1, \ldots$ **do**
  Compute $\boldsymbol{s}_t = \mathrm{LMO}_{\mathcal{A}}(\nabla f(\boldsymbol{x}_t))$
  Let $\boldsymbol{d}_t^{\mathrm{FW}} = \boldsymbol{s}_t - \boldsymbol{x}_t$
  Compute $\boldsymbol{v}_t = \arg\max_{v\in\mathcal{S}_t} \langle \nabla f(\boldsymbol{x}_t), v\rangle$
  Let $\boldsymbol{d}_t^{\mathrm{A}} = \boldsymbol{x}_t - \boldsymbol{v}_t$
  if $g_t^{FW} := \langle -\nabla f(x_t), d_t^{FW}\rangle \leq \epsilon$ **then return** x_t
  if $\left\langle -\nabla f(\boldsymbol{x}_t), \boldsymbol{d}_t^{\mathrm{FW}}\right\rangle \geq \left\langle -\nabla f(\boldsymbol{x}_t), \boldsymbol{d}_t^{\mathrm{A}}\right\rangle$ **then**
    $\boldsymbol{d}_t = \boldsymbol{d}_t^{\mathrm{FW}}$ and $\gamma_{\max} = 1$
  else
    $\boldsymbol{d}_t = \boldsymbol{d}_t^{\mathrm{A}}$ and $\gamma_{\max} = \frac{\alpha^{(t)}_{\boldsymbol{v}_t}}{1-\alpha^{(t)}_{\boldsymbol{v}_t}}$
  Set γ_t **by line-search, with** $\gamma_t = \arg\min_{\gamma\in[0,\gamma_{\max}]} f(x_t + \gamma_t d_t)$
  $\boldsymbol{x}_{t+1} = \boldsymbol{x}t + \gamma_t \boldsymbol{d}_t$
  $\mathcal{S}_{t+1} = \{\boldsymbol{v} \in \mathcal{A} \text{ s.t. } \alpha^{t+1}_{\boldsymbol{v}} > 0\}$

guarantee that the active set S_t is still a convex representation, i.e., $S_t \subseteq \mathcal{A}$. For a FW step we have $S_{t+1} = s_t$ if $\gamma_t = 1$ and otherwise, $S_{t+1} = S_t \cup s_t$. The corresponding update of the weights is $\alpha_v^{(t+1)} = (1-\gamma_t)\alpha_v^{(t)}$ when $v \in S_t \backslash s_t$ and $\alpha_{s_t}^{(t+1)} = (1-\gamma_t)\alpha_{s_t}^{(t+1)} + \gamma_t$ otherwise. For away step we have the following update rule. When $\gamma_t = \gamma_{max}$, then $S_{t+1} = S_t \backslash v_t$. Combined with $\gamma_{max} < 1$ we call them bad drop step, as it corresponds to a situation in which we are not able to guarantee a geometrical decrease of the dual gap.

For AS in which $\gamma_t < \gamma_{max}$, the away atom is not removed from the current representation of the iterate. Hence $S_{t+1} = s_t$, $\alpha_v^{(t+1)} = (1-\gamma_t)\alpha_v^{(t)}$ for $v \in S_t \backslash s_t$ and $\alpha_{s_t}^{(t+1)} = (1-\gamma_t)\alpha_{s_t}^{(t+1)} - \gamma_t$ otherwise.

4.3.1 Randomized Away-Step Frank Wolfe (RAFW)

The second variant that we propose is an evolution of the RFW previously introduced which includes the properties of the AFW, with the name of Randomized Away-Step Frank Wolfe (RAFW). The major changes that come up with this version cover the ways how the FW LMOs are computed. In fact, the minimization originally computed in the AFW Algorithm [6] had been replaced by its randomized version, for which the s_t solution comes from a subsampled set $S_t \cup \mathcal{A}_t$, where $\mathcal{A}_t$ is a subset of size $\min\{p, \mathcal{A}\backslash S_t\}$, sampled uniformly at random from $\mathcal{A}\backslash S_t$, at each iteration.

The second LMO is still the same as the one in AFW, i.e., it is computed only on the active set S_t.
The introduced changes are summarized in Algorithm 5.

Algorithm 5 RANDOMIZED AWAY-STEP FW (RAFW)

Input $\boldsymbol{x}_1 \in \mathcal{M}$, $\boldsymbol{x}_1 = \sum_{\boldsymbol{v}\in\mathcal{A}} \alpha_{\boldsymbol{v}}^{(1)} \boldsymbol{v}, \mathcal{S}_1 := x_1$, with $|\mathcal{S}_0| = s$, and p a subsampling parameter s.t $1 \leq p \leq |\mathcal{A}|$.

1: **for** $t = 1, \ldots$ **do**
2: **Get** $\mathcal{A}_t$ **by sampling** $\min\{p, |\mathcal{A}\backslash\mathcal{S}_t|\}$ **elements uniformly from** $\mathcal{A}\backslash\mathcal{S}_t$
3: **Compute** $\boldsymbol{s}_t = \mathrm{LMO}_{\mathcal{S}_t\cup\mathcal{A}_t}(\nabla f(\boldsymbol{x}_t))$
4: **Let** $\boldsymbol{d}_t^{\mathrm{FW}} = \boldsymbol{s}_t - \boldsymbol{x}_t$
5: **Compute** $\boldsymbol{v}_t = \arg\max_{\mathcal{C}} \langle \nabla f(\boldsymbol{x}_t), v\rangle$
6: **Let** $\boldsymbol{d}_t^{\mathrm{A}} = \boldsymbol{x}_t - \boldsymbol{v}_t$
7: **if** $\left\langle -\nabla f(\boldsymbol{x}_t), \boldsymbol{d}_t^{\mathrm{FW}} \right\rangle \geq \left\langle -\nabla f(\boldsymbol{x}_t), \boldsymbol{d}_t^{\mathrm{A}} \right\rangle$ **then**
8: $\boldsymbol{d}_t = \boldsymbol{d}_t^{\mathrm{FW}}$ **and** $\gamma_{\max} = 1$
9: **else**
10: $\boldsymbol{d}_t = \boldsymbol{d}_t^{\mathrm{A}}$ **and** $\gamma_{\max} = \frac{\alpha_{\boldsymbol{v}_t}^{(t)}}{1-\alpha_{\boldsymbol{v}_t}^{(t)}}$
11: **Set** γ_t **by line-search, with** $\gamma_t = \arg\min_{\gamma\in[0,\gamma_{\max}]} f(x_t + \gamma_t d_t)$
12: $\boldsymbol{x}_{(t+1)} = \boldsymbol{x}_t + \gamma_t \boldsymbol{d}_t$
13: $\mathcal{S}_{t+1} = \{\boldsymbol{v} \in \mathcal{C} \text{ s.t. } \alpha_{\boldsymbol{v}}^{t+1} > 0\}$

Apart from the changes in the way the FW LMO is computed, the remaining parts shares the same structure with the AFW Algorithm 4 previously explained. What is interesting to notice is how the cost per iterations depends on the size of the active set, which varies throughout the iterations. However, for problems with sparse solutions, in [16] it has been observed empirically that the size of the active set remains small, making the cost of the second LMO and the comparison of Line 7 negligible compared to the cost of an LMO over the full atomic domain. Assuming also that the atomic domain has a sparse structure that allows gradient coordinate subsampling, RAFW can achieve a per iteration cost that is, like RFW, roughly $|\mathcal{A}|/p$ times lower than that of its deterministic counterpart.

Regarding the rate of convergence of RAFW, theorem 4.3.1 holds.

Theorem 4.3.1. *Consider the set $\mathcal{M} = conv(\mathcal{A})$, with $\mathcal{A}$ a finite set of extreme atoms, after T iterations of Algorithm 2 (RAFW) we have the following linear convergence rate:*

$$E[h(x_{T+1})] \leq (1 - \eta^2\rho_f)^{max\{0,\lfloor (T-s)/2\rfloor\}} \tag{21}$$

with $\rho_f = \frac{\mu_f^{\mathcal{A}}}{4C_f^{\mathcal{A}}}$, $\eta = \frac{p}{|\mathcal{A}|}$ and $s = |S_0|$.

This theorem shows that theoretically RAFW does not bring any computational advantage with respect to AFW as shown in Section 5.1.4 regarding the Frank Wolfe Variants experiments.

4.3.2 Pairwise Frank-Wolfe (PFW)

The last variant that we present is based on an algorithm originally proposed by Mitchell et al. [2] for the polytope distance problem and it is referred as *Pairwise Frank-Wolfe*. It extends the behavior of the AFW by avoiding the comparison between the Frank-Wolfe and Away-Step gap, moving the weight values related to the AS vertex v_t to the FW one, called s_t, instead. This operation is called *pairwise step* and it is really important in order to demonstrate how this algorithm can achieve a linear convergence rate that is more loose with respect to the AFW on the theory, but very effective in practice. The pairwise schema is described in Algorithm 6.

Algorithm 6 Pairwise Frank Wolfe (PFW)

Input $\boldsymbol{x}_1 \in \mathcal{M}$, $\boldsymbol{x}_0 = \sum_{\boldsymbol{v} \in \mathcal{A}} \alpha_{\boldsymbol{v}}^{(1)} \boldsymbol{v}$ and $\mathcal{S}_1 := x_0$

1: **for** $t = 1, \dots$ **do**
2: **Compute** $\boldsymbol{s}_t = \mathrm{LMO}_{\mathcal{A}}(\nabla f(\boldsymbol{x}_t))$
3: **Compute** $\boldsymbol{v}_t = \arg\max \mathcal{A} \langle \nabla f(\boldsymbol{x}_t) \rangle$
4: **Let** $\boldsymbol{d}_t^{\mathrm{PFW}} = \boldsymbol{s}_t - \boldsymbol{v}_t$
5: **Set** $\gamma_{\max} = \alpha_{v_t}$
6: **Set** γ_t **by line-search, with** $\gamma_t = \arg\min_{\gamma \in [0, \gamma_{\max}]} f(x_t + \gamma_t d_t)$
7: $\boldsymbol{x}_{(t+1)} = \boldsymbol{x}_t + \gamma_t \boldsymbol{d}_t$
8: $\mathcal{S}_{(t+1)} = \{\boldsymbol{v} \in \mathcal{A} \text{ s.t. } \alpha_{\boldsymbol{v}}^{(t+1)} > 0\}$

5 Experiments and Results

In the following section we describe the numerical experiments that tested the properties of both the algorithms and norms previously introduced. To achieve this, we relied on what was proposed in [16, 15], extending their experiments with the addition of some insights that we found meaningful to our goals. The results are presented with the following schema. At first, we consider the Lasso regularization, providing a characterization of the performance for the optimization algorithms in Section 4 on a synthetic and real dataset. Secondly, we move to the Latent Group Lasso regularization, but this time without considering a real world dataset but preferring an artificial one with bigger dimensions than the previous. From a technical perspective, the experiments have been coded in

two programming languages: Python and R. The reason behind this choice came up after some technical difficulties experienced by working on large scale datasets with R, for which we decided to look for an alternative that helped us to overcame this problem. As a consequence of this, we performed some experiments to assess which language is the most appropriated for this kind of computational problems.

5.1 Lasso regression problem

The first part of the results covers the Lasso regression problem introduced in Eq.4, for which we analyze:

1. The dual gap values for all the iterations;
2. The dual gap values versus the cumulative number of the coefficients computed of gradient per call to LMO;
3. The evolution of the number of the non-zero coefficients in the support;
4. The number of the recovered coefficients of the original solution related to our support;
5. The time of two RFW variants;
6. The cumulative running time of each algorithm on Python and R.

5.1.1 Artificial Dataset

The artificial dataset built for the first part of the experiment consists of a coefficient matrix $A \in \mathbb{R}^{200\times 500}$ whose values come from a Gaussian distribution and a solution vector $b \in \mathbb{R}^{200}$ such that $b = Ax^* + \epsilon$, where ϵ is a random Gaussian vector which represents our noise and x^* is a vector with 10% of nonzero coefficients and values in $\{-1, 1\}$. The l_1 regularization term of the problem was formalized as a l_1 ball with radius = 50, for which we can define the set of its vertices as $\mathcal{V} = \{\pm e_i : i = 1, \ldots, 500\}$.
Fig.5 compares the performances of Frank wolfe and Randomized Frank Wolfe with subsampling parameter $\eta = \frac{d}{|A|} = 0.05$ on the synthetic dataset. In the study of the RFW algorithm, two ways of subsampling are considered. In the first, we experiment a version with a non-continuous slicing, which consists of choosing randomly $d * \eta$ elements from the gradient over the 500 at each iteration, with an equal probability $P(v \in A_t) = \eta$. In the second version, the $d * \eta$ values are subsampled in a continuous way, for which the randomness consists only in the choice of the initial point. This choice seems to bring an advantage in terms of time of the iteration with respect to its non-continuous counterpart,

as upper right plot in Fig.5 shows. Apart from this, the results of the two versions of RFW are quite similar (we omit the plots in order to avoid being redundant), but in the end we consider the non-continuous variant since in [16] is stated that all the atoms should be chosen at random.

The upper left plot in Fig.5 shows the evolution of the FW dual gap for both the deterministic and the randomized FW. The reason why RFW requires more iterations to converge is that each call to the randomized LMO outputs a direction, likely less aligned with the opposite of the gradient than the FW direction. In terms of coefficient computed by the gradient at each time, the bottom left panel of Fig.5 shows how the RFW requires a smaller number than its deterministic counterpart. The reason is due to the randomized LMO which works only on the subsample and not on the entire domain. A larger number of cheaper iterations led to a sub-linear rate of convergence, like in the classical Frank Wolfe algorithm. In the end, the bottom plot of the figure shows the number of coefficients in the solutions that are in the same position as the one in the original solution. We can see how the FW outperforms RFW, and we may think that is due to the lack of complete domain available at each iteration that leads RFW to pick vertices that are not optimal.

5.1.2 Real World Dataset

The E2006-TFIDF [7] is a real dataset typically used in the context of large scale regression. It is composed of a set of financial reports from thousands of U.S. companies, each of them associated with an empirical measure of financial risk. The training set stores all these information from 2001 to 2005, resulting in $n = 16087$ training examples and $d = 150360$ features, while the test set contains the same features but only for the 2006 year, with a number of examples equal to 3308. In [16] it is reported that the number of features was sampled up to $d = 8000$ elements. Since it is not clearly stated which slice they chosen and which criterion they applied, we decided to consider a set of the features in which there is a large number of non-zero values in order to enhance the sparsity of the solution, choosing features from position 40000 to 48000.

Fig.6 compares the performances of Frank wolfe and Randomized Frank Wolfe with subsampling parameter $\eta = \frac{d}{|A|} = 0.06$ and a number of iteration equal to 1000. The results are coherent with the previous obtained in the synthetic dataset and in [16] . In the left panel, we illustrate the comparison of the FW dual gap for the two algorithms (top) and the evolution of the number of coefficients of the gradient with respect to the number of coefficients computed (bottom). This leads us to confirm that, like in Section 5.1.1, there is a trade-off between the number and the cost of each iteration. To end this part, in the right panel of Fig.6 we propose the same comparison of the two variants of RFW

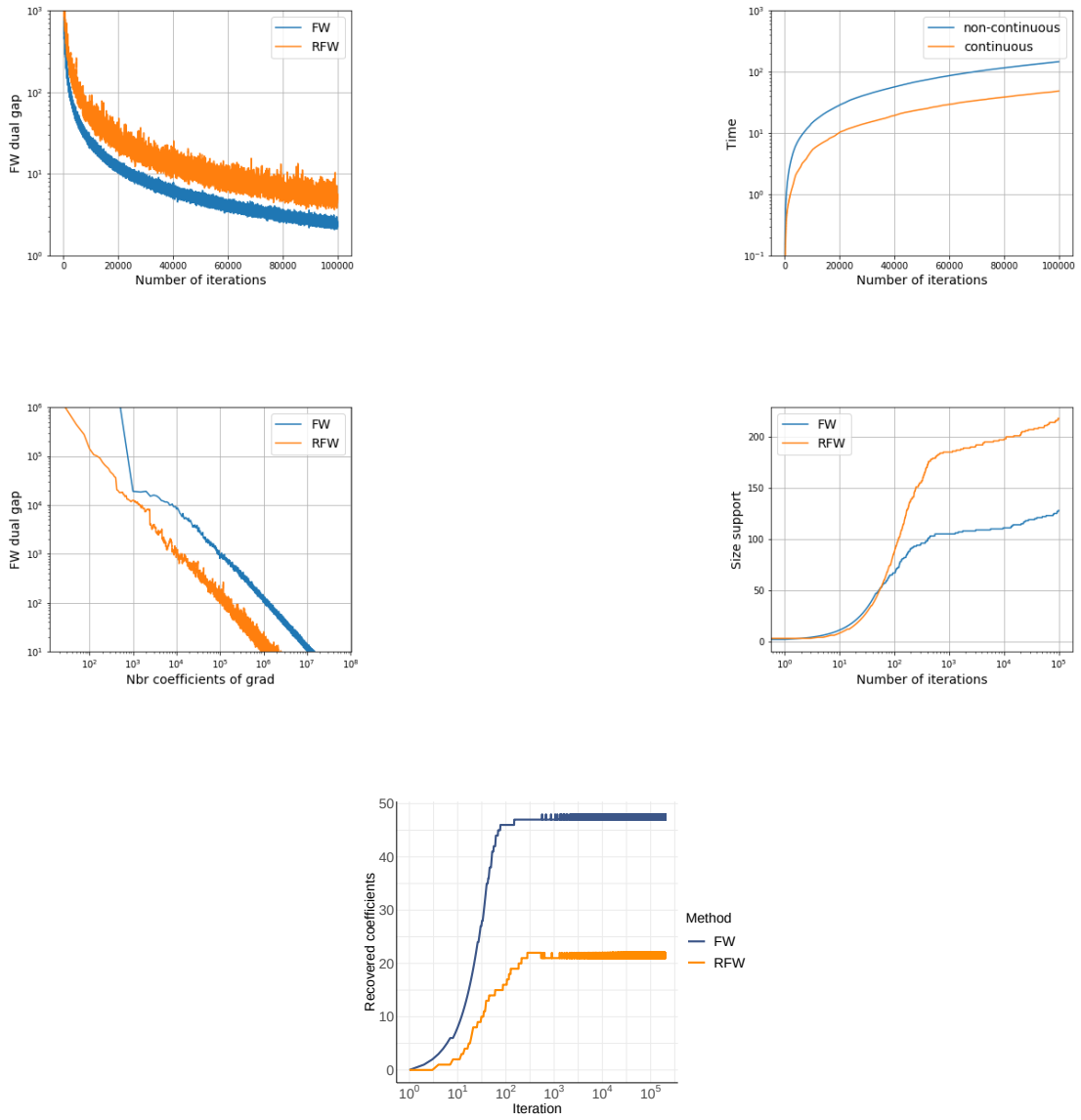


Figure 5: Comparison of FW and RFW on the Lasso problem using an artifical dataset. Upper left: evolution of FW dual gap with respect to the number of iterations. Lower left: progress of the cumulative number of computed coefficients. Upper right: comparison of the two ways of sampling. Lower right: study of the sparsity of the iterate. Bottom line: number of recovered coefficients belonging to the original solution. We did not include the RFW with continuous slicing since the performs were the same as the non continuous version.

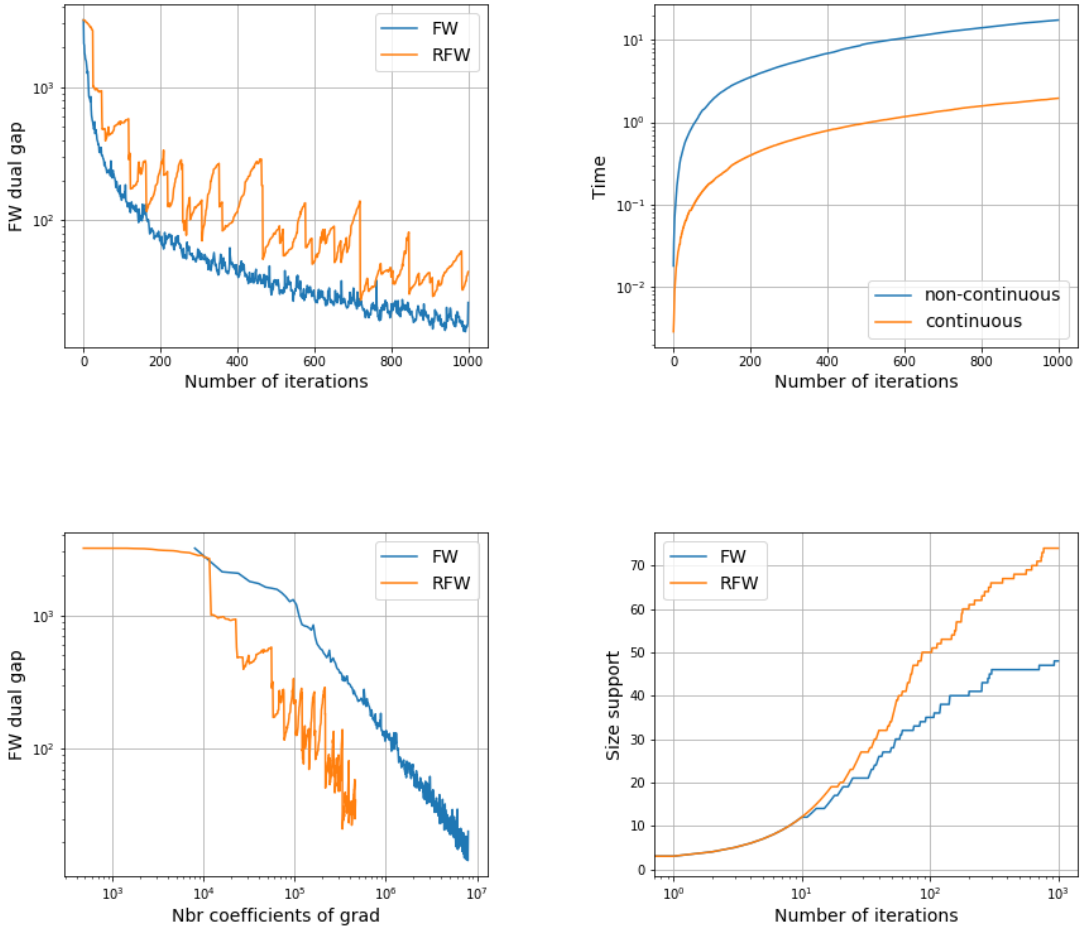


Figure 6: Comparison of FW and RFW on the Lasso problem using the E2006-TF-IDF dataset. Upper left: evolution of FW dual gap with respect to the number of iterations. Lower left: progress of the cumulative number of computed coefficients. Upper right: comparison of the two ways of sampling. Lower right: study of the sparsity of the iterate.

(top) and the size of the support (bottom). What was affirmed for the synthetic dataset seems to be true even in this case: RFW with continuous slicing performs faster than the one used in the paper while both the variants reduce the sparsity of the estimation due to its struggle to be aligned with the opposite of the gradient.

5.1.3 Time comparison

Fig.7 shows the cumulative computation time for the LMO in FW and RFW with non-continuous and continuous slicing in the case of the artificial dataset (left) and for the real

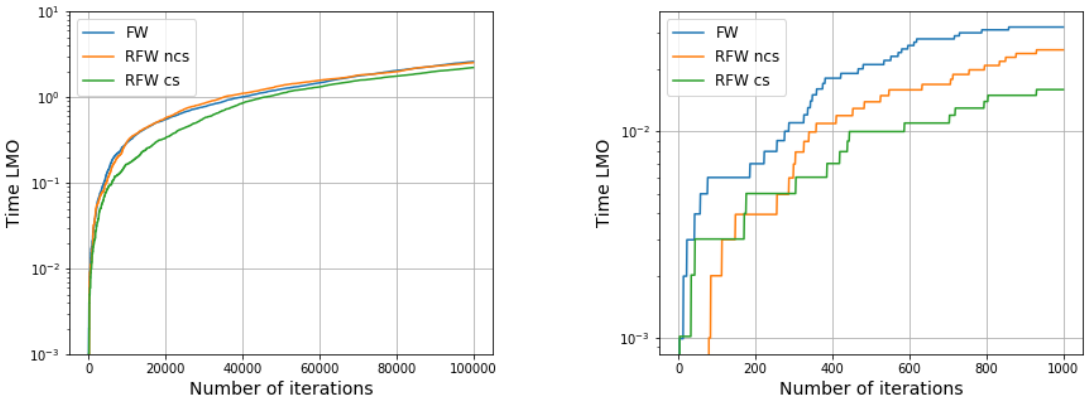


Figure 7: LMO time comparison of the algorithms on the artificial (left) and real dataset (right). FW scores the worst results due to the computation of the full gradient, whilst the RFW with continuous slicing may take advantage of the fact that continuous array vectors have more chance to be stored next to each other, reducing the access time in memory.

world dataset (right). As we can notice in the plot on the left, the curves overlap and the reduction of time for the randomized version is negligible. The green line representing the RFW with continuous slicing is the one obtaining the best results. We point that the reason behind this behavior can be found in the way arrays are represented in memory. In fact, by knowing only the begin and the end of the subset interval, the access time can be reduced by retrieving the desired chunk of memory corresponding to the column selected. The non continuous variant instead has to perform manual of indexing for each of the random index obtained, which are less likely to be next to each other, especially in the context of large scale dataset. This latter behavior seems to be confirmed in the right panel where, in the context of large scale dataset, the gap seems to increase between the two RFW variants.

Given the fact that the FW and RFW algorithms in this first part are coded in Python language whilst the variants are created in R, we have decided to introduce a further time comparison between the two languages. We performed the test on the synthetic dataset scenario with increased dimensions by setting $n = 2000$ and $d = 6000$, running FW and RFW in the two programming languages for a total of 10000 iterations.

Fig.8 shows the result of this comparison. For what concerns the execution of the FW, we can see how R ends the execution of the algorithm later than Python. The behavior seems not to be maintained for what concerns the RFW, where R outperforms Python. We think that this diversity is mainly due to the libraries that we used for the coding of

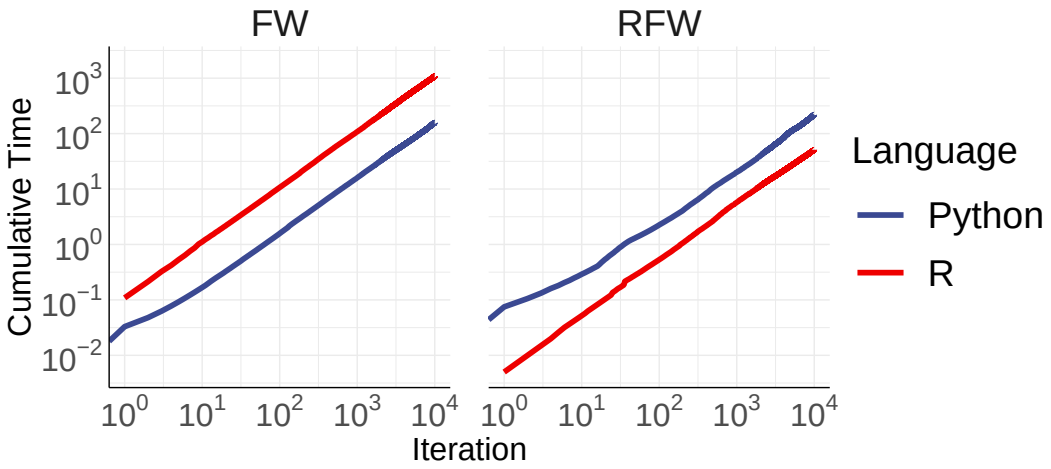


Figure 8: Comparison between FW (left) and RFW (right) algorithms coded in Python and R. The results show an insight about the two languages perform on different scales. In fact, the less computation time required from R in the case of RFW indicates that a better management of small datasets. When the entire domain is considered, as in the case of FW, Python seems to manage matrices better, thanks to the *numpy matrix*, with less computational time required.

the algorithms. In fact, in the case of FW, where we have to consider the entire domain size, the *numpy* and *scipy* packages provided by Python seem to manage matrices better. In R, instead, we represent the matrices as R does, without the usage of any specific library for scientific computation. However, in the case of small dataset, the scenario completely changes. Since we use the same η as the synthetic experiments, i.e., $\eta = 0.05$, the domain is severely reduced, and R seems to outperform Python. This comparison is important because the experiments on the FW variants have been run in R, and in fact we had to do some workarounds in the case of the real dataset due to the poor memory management provided by R when the matrix dimensions become important.

5.1.4 Frank Wolfe Variants

After having studied the behavior of FW and RFW for the Lasso problem, we shift the focus on their variants, i.e., FW, RAFW and PFW, replicating all those applications [16, 15] for which they are involved. At this time, the results that we obtain varies significantly between synthetic and real dataset. In the first one, we are able to display the linear convergence rate of all three algorithms as the upper left plot in Fig.9 shows. By looking at the type of steps computed in Fig.10, the slight difference of both gaps by the number of itera-

tions seems reasonable since the proportion between FW steps and Away Steps are around 50%. For what concerns the gap in relation with the number of coefficients computed in the gradient, we can see how the randomized version computation requires less elements, coherently for what we obtained in the RFW, resulting in a faster execution although than the AFW algorithm. This overperformance of RAFW goes away if we consider the sparsity of the solution provided. We can demonstrate this by looking right plot in Fig.9: RAFW provides a solution whose support is remarkably larger than the one obtained from AFW, including a fewer number of coefficients with a position corresponding to the one in the original solution. This behavior can be explained in the same way as we did for RFW, where its non random counterpart, in this case AFW, has a more conservative behavior which seems similar with the one obtained in [16].
The second part of the commentary focuses on the difference between PFW and the FW variants previously introduced and on the accordance of the results with the findings in [15]. As illustrated in the top left plot in Figure 9, PFW outperforms the other variants due to its rapid decrease of the dual gap. We explain this efficiency by referring to the *pairwise step* behavior, for which the coefficients are moved from the atom v_t to s_t, avoiding the choices between one kind of distance to another as the Away variants does, and thus by preserving its linear convergence at all without any FW step. Of course, this choice does not benefit of the advantage of a partial LMO computation provided by RAFW, but instead we obtain a solution that is more sparse than the others, even if it has the smallest number of original coefficients preserved.

The analysis of the variants then ends by comparing the same algorithms on the E2006-TF-IDF dataset. In this case, Fig.11 describes how the results obtained did not maintain any of the behaviors previously explained. Surprisingly, the advantages brought by the PFW seems not working in this case, and at the same time RAFW shows a zigzagging-like phenomenon which results in an unstable gap computation, establishing AFW as the best one in terms of stability/gap trade-off. Finally, we cannot state the results that we have obtained are coherent with the one in [16], and this may be due to the following reasons.

First, from the results in 8 we have understood how much R decreases in terms of performance as the size of the domain increases, and this was a problem to deal with since FW variants were analyze on this language despite the previous ones that were tested on Python. In addition to this, the computer where we have reproduced these tests was unable to manage a matrix with the dimension considered in the original paper, thus we reduced d from 16087 to 8000 to obtain a new $A \in \mathbb{R}^{8000\times 8000}$. We think that this choice have massively affected the computation of the gradient and the LMO because, by the results that we have obtained, it seems like the overall computation of the directions

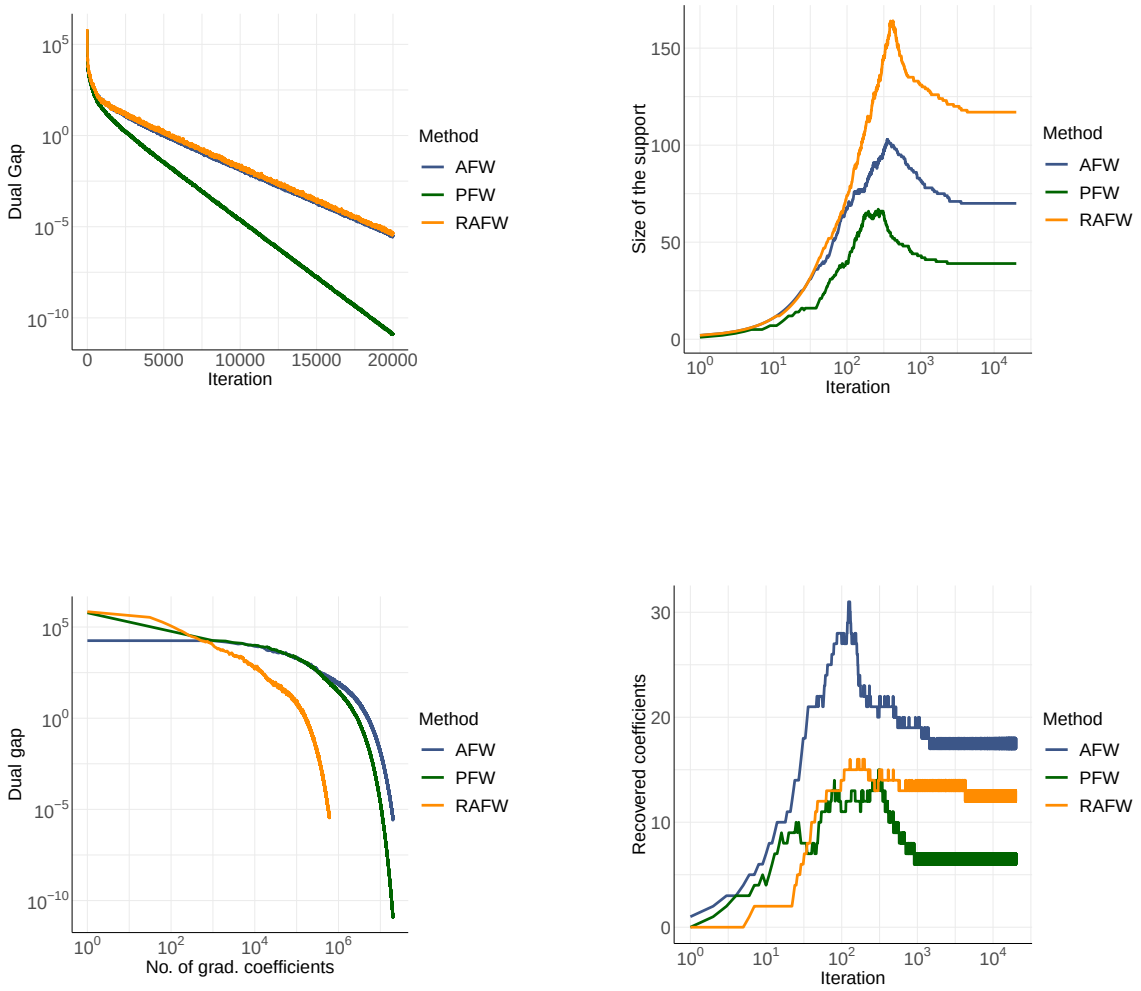
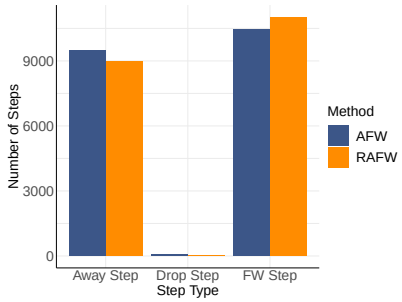


Figure 9: Comparison between the FW variants. Top left: dual gap values by the number of iterations. Bottom left: dual gap values by the cumulative number of coefficients computed by the gradient. Top right: size of the of the support during the iterations. Bottom right: number of recovered coefficients along the iterations. PFW outperforms the other variants in terms of dual gap by iteration and size of the support, but if we consider all the metrics, we find that AFW is a good compromise. In fact, if we do not consider the gap metrics, we find that it performs better than PAFW and RAFW, which confers it the best performance tradeoff.

missed some important atoms. This is supported by looking at the left plot in Fig.11. The gaps values that we have obtained do not reflect how they should be in relation with its



AFW	**Our Experiments**	**Kedreux et al.**
Away Step	9481	880
Drop Step	51	14
FW Step	10468	19106
RAFW	**Our Experiments**	**Kedreux et al.**
Away Step	8953	1242
Drop Step	19	37
FW Step	11028	18721

Figure 10: Comparison between the type of steps produced in our AFW and RAFW experiments for the synthetic dataset with the results obtained in [16]. Although the small difference in terms of gaps produced by the two algorithms should require statistical tests in order to assess if it does simply depend on the seed chose, we think that a reason for this behaviour can be found in the number away steps which is higher in the AFW. This means that the convergence should be faster than RAFW, which also implies a lower value of the gap produced.

benchmark. Also, our AFW does not end its computation after 60 iteration, unlike its benchmark. Furthermore, we think that the initial point, given the size of the dataset, has to be chosen properly, and since we miss this information, we have opted for a bias-free solution by choosing one atom at random, the same for every variant. In the end, the lack of information about the support lead us to not bring any information related too, because we are not sure if they still applied a threshold on the coefficients in order to bring them to 0, which may affect the results.

5.2 Latent Group Lasso

The second part of the analysis focuses on the Latent Group Lasso penalty, with the related theoretical aspects defined in Section [3.3]. In our experiment, we consider a synthetic

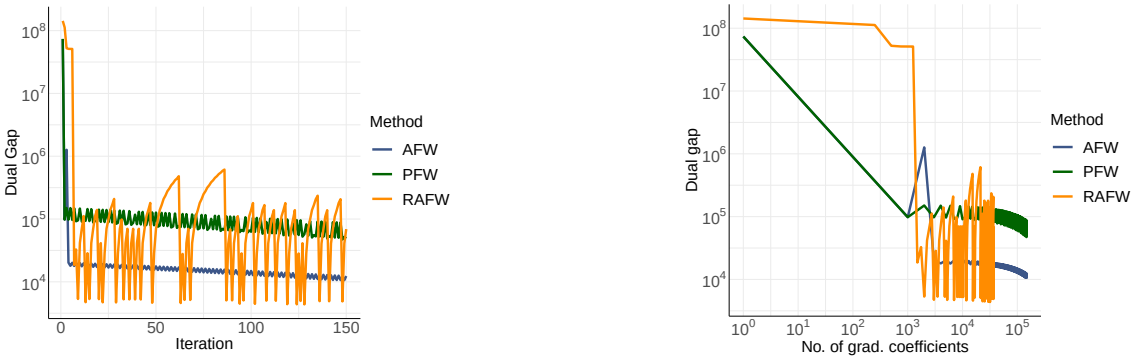


Figure 11: Experiment results from the application of FW variants on the 2006-TF-IDF dataset. On the left plot, representing the values of the dual gap by their iteration, AFW appeared to perform better than its competitors, with the lowest and stablest gap. In general, the results obtained from the synthetic scenario were not confirmed in the real one, which are actually the opposite. Right: dual gap trend by the cumulative number of coefficients computed at each iteration from the gradient. The most interesting result in this case comes from RAFW: it shows how around 10^3 coefficients are required in order to assist for a drop of the dual gap value. This may demonstrate the struggle to find a vertex that can lead to a descent direction that converges quickly.

scenario similar to the previous presented, following the setup in [16]. We then consider a matrix $A \in \mathbb{R}^{1000\times 10000}$ whose values are generated again from a Gaussian distribution. For the groups instead, our set G contains groups $g \in \mathcal{G}$ of size 10 with an overlap of 3, associated with the atomic set $\mathcal{A}$. The solution $w_0 \in \mathcal{A}$ has a number of 0.01 percentage of non zero coefficients.

The first technical aspect that we want to cover in this experiment is the creation of the dataset by looking at the way that we managed the overlapping between the groups. We approached it based on the statement in Section 8 of [6] for which, through covariates duplication, it is possible to see this problem like a Group Lasso problem. According to this, we refer to the definition of the duplication operator $\bigoplus : \mathbb{R}^p \to \mathbb{R}^{\sum_{g\in\mathcal{G}}|g|}$ presented in [9], such that $w \to \tilde{w} = \bigoplus_{g\in\mathcal{G}} (w_i)_{i\in g}$. This means that our $\tilde{w}$ has a dimension that is equal to the sum of the size of each group, thus replicating all those indexes of the elements that overlap. With the same way, for a vector $v \in \mathcal{V}_\mathcal{G}$, we can obtain its corresponding definition $\tilde{v} \in \mathbb{R}^{\sum_{g\in\mathcal{G}}|g|}$ without losing any information about the v_g because, as we have seen, their support is a subset of a group g, which means they components are not defined elsewhere. These considerations lead us to redefine the risk minimization problem as:

$$\min_{\tilde{v}\in\mathbb{R}^{\sum_{g\in\mathcal{G}}|g|}} \tilde{L}(\tilde{A}\tilde{v}) + \lambda \sum_{g\in\mathcal{G}} |\tilde{v}_g| \tag{22}$$

where $\tilde{A}$ is the $n \times (\sum_{g\in\mathcal{G}} |g|)$ matrix with a larger number of columns and $\tilde{v}_g$ are our previous $\tilde{v}$ but with the coordinates of the different groups g.

Roughly speaking, in this way we can manipulate the columns of $\tilde{A}$ in order to choose the columns whose indices are related to the group $\tilde{v}_g$. This introduces the idea to create, from a programming perspective, a data structure that is capable to store the indices related to each group, reproducing the purpose of $\tilde{v}_g$. In this way, in our algorithms we will do no more than picking each of these groups, getting the indices within to access to the subset of $\tilde{A}$. A representation of what we have just said is showed in fig. 12.

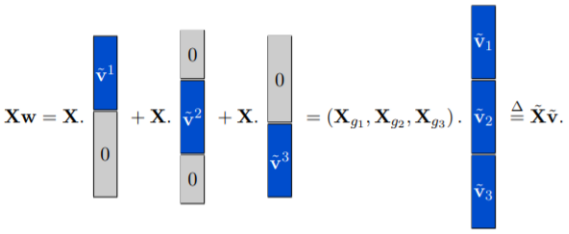


Figure 12: Application of the covariate duplication in a context of a risk minimization problem. The result is the product $\tilde{X}\tilde{v}$ which means that the space of the problem was extended such that we can operate on the groups indicated by the indices in $\tilde{v}_g$ without considering the fact that there are some covariates in common, reformulating the problem like a typical Group Lasso problem.

To realize this idea, we start by computing the number of groups $|G| = \frac{d}{group\_size - overlap}$. The reasoning behind is the following: the number of groups is equal to the number of distinct indices that it contains, which is given by the size of the group after removing removed the common variables. In our case, it is 7. After that, for each group we create a list of its indices by taking into account the overlap with its previous group in order to set the starting index coherently. Now that the groups are created, we pick a number of groups $n_g = d * 0.01$ whose coefficients will appear in the ground truth solution w_0. For each of those groups, we obtain the corresponding feature indices that we used to initialize w_0 with a Gaussian value.
The second aspect is related to the structure of the algorithms that we test, i.e., FW and RFW. Unlike the lasso, we have to manage the structural sparsity and therefore, we expect that the previous definition of the LMO has to be modified to suit our problem. [16]

proposed the following formulation for the RFW:

$$\text{LMO}(x_t, \mathcal{A}_t) \in \arg\min_{v \in \mathcal{A}_t} \big\langle v_{(g_p)}, \nabla_{(g_p)} f(\boldsymbol{x_t}) \big\rangle. \tag{23}$$

where $g_p = \bigcup_{g \in G_p} g$. This can be generalized as follows: the vertex that provides the better contribution for our gap comes from the group that minimizes the gradient at most, between the set of available groups $\mathcal{G}_p$. In particular, in the FW case, $\mathcal{G}_p = \mathcal{G}$ and the domain of vertex $\mathcal{A}_t = \mathcal{A}$, whilst in the RFW we have $\mathcal{G}_p$ a set of group indices randomly chosen, and $\mathcal{A}_t = \cup_{g \in \mathcal{G}_p} \mathbb{D}_g$ the set of vertices whose indices are retrieved from the groups g in the set of the random ones chosen $\mathcal{G}_p$.

From what concerns the computation of the LMO, the reformulation of the problem as a Group Lasso regression problem allowed us to refer on Lemma 1 in [12] to retrieve the vertex of our interest. Following this idea, at each iteration we compute the total gradient $\nabla f(x_t)$, from which we obtain $\nabla f_{(g_p)}(x_t)$ and its l_2 norm, corresponding to the groups considered. At the t^{th} iteration, we find the group i such that the corresponding block of the gradient $\nabla f_{(g_p)}(x_t)$ has the largest l_2 norm. As suggested in [12], the following step is to compute:

$$s_t = \frac{-\beta * \nabla f_{(g_p)}(x_t)}{\|\nabla f_{(g_p)}(x_t)\|_2}. \tag{24}$$

The analysis that we made are the same as [16]. We tested the trend of the gap with respect to the number of coefficients computed and the iteration time. The results described in Fig.13 differed from our benchmarks. From what concerns the gap by the iteration time, we think that the divergence are due to the way [16] build the model. In fact, they used a streaming architecture where, at each iteration, they only retrieved a single chunk of the dataset with size $n \times 500$. In our case, we stored everything in RAM and run the experiment on R, for which we have seen 8 that do not performs well in large scale contexts as this one. Despite the differences from our benchmark, we can see how FW performs better than RFW. In fact, the randomized version seems not capable to find a series of subsamplings that lead to a descent direction. This is explained by the trend of the gap: the instability it shows in its descent seems accounting to a zig-zagging phenomenon, and this is the reason for which FW performs generally better. For what concerns the gap versus the number of coefficients computed from the gradient, our results shows a similar trend, apart for the values in y axis that are much higher than the one in the benchmark. In addition to this, FW is the one that obtain a better final solution by looking at their final gap value. We think that this is mainly due to the way we implemented these algorithms and especially with the assumptions that we made for computing the dual gap.

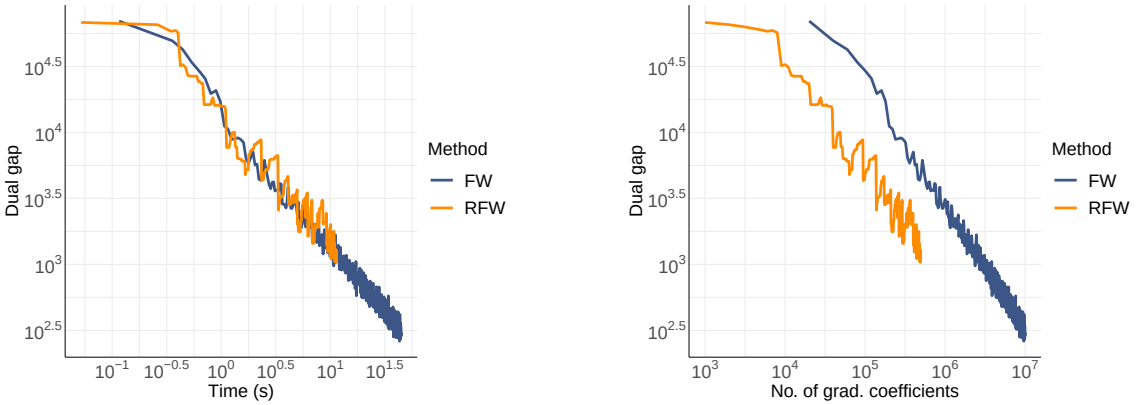


Figure 13: Results from Latent Group Lasso analysis on a synthetic dataset with large dimension. Left: Dual Gap values by the cumulative execution time. Right: Dual Gap values by the number of coefficients computed. The RFW shows a trend of the dual gap values in relation with the cumulative iteration time which shows a zigzagging phenomenon, making the FW as the best since it more stable and reaches a better final value. This is probably due to the lack of streaming model that our benchmark provided. For what concerns the gap by the number of coefficients computed, it performed better as expected, but this is mainly due to the subsampling which ended in a gradient and a LMO that considered less coefficients than the FW version.

6 Conclusion

In this project, we have studied different regularization related to problems whose solutions are affected from sparsity. We started by looking at Lasso, describing its properties and limitations in order to introduce more complex penalties that could improve the previous ones, such as Group Lasso and Latent Group Lasso. For the Lasso and the Latent Group Lasso, we minimized a regression problem subject to each of those regularizations by implementing constrained convex optimization algorithms belonging to the family of Frank Wolfe, discussing variants coming from the recent and the classic literature. For the Lasso regression problem, we run all the optimization algorithms both on an artificial and a real world scenario, discussing the results. In the artificial dataset, FW seems performing better in terms of the gaps obtained through the iterations and the size of the support produced, whilst RFW is better in terms of number of coefficients required, thanks to its subsampling, and in execution time. This behavior was reflected even in the real dataset. The analysis continued by studying the FW variants in the same way. We found that, in

the case of the synthetic dataset, PFW outperformed AFW and RAFW in terms of dual gap produced during the iterations, as well as in the size of the retrieved support. However, in terms of recovered coefficients in the same position as the original solution, it was the one performing the worst, from which RAFW seemed the method with best trade-off of all the metrics computed. The properties from the algorithms that resulted in scenario were not maintained in the case of the real dataset. The reasons behind this behavior are associated with the combinations of lack of resources in the computer for which these tests have been run and the language used to code these algorithms R. Therefore, we made a comparison between Python and R in terms of execution times, founding that Python outperforms R on large scale context, and thus confirming our hyphothesis about the lack of performance from R.

The experiments moved on the study of a regression problem regularized by Latent Group Lasso, describing all the properties which made us able to reformulate as it was a typical Group Lasso problem. We implemented the FW and RFW algorithms on a synthetic scenario with a larger dimension than the previous one, discussing how to implement the LMO since the problem had to maintain structural sparsity. We then tested the execution time of the algorithm and the dual gap values by the number of coefficients computed. The results that we have obtained differ from our benchmark. For what concerns the gap by the execution time, we obtained that RFW does not produce a lower dual gap than the FW, resulting in an opposite conclusion with respect to the benchmark results. This is due to the fact that we do not provide a streaming model like it has been done in the original paper, storing the entire dataset in RAM instead of retrieving only the parts of interest. For what concerns the gap versus the number of computed coefficients of the gradient instead, we found RFW working better than FW since it requires fewer computations at each LMO. In general, we think that the results could be improved keeping into account weights for each group.

References

[1] Philip Wolfe. *Integer and nonlinear programming.* North-Holland, 1970.

[2] B. F. Mitchell, V. F. Dem'yanov, and V. N. Malozemov. "Finding the Point of a Polyhedron Closest to the Origin". In: *SIAM Journal on Control* 12.1 (1974), pp. 19–26. DOI: 10.1137/0312003.

[3] Robert Tibshirani. "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1 (1996), pp. 267–288. ISSN: 00359246. URL: http://www.jstor.org/stable/2346178.

[4] Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. "Atomic Decomposition by Basis Pursuit". In: *SIAM Review* 43.1 (2001), pp. 129–159. ISSN: 00361445. URL: http://www.jstor.org/stable/3649687.

[5] Ming Yuan and Yi Lin. "Model selection and estimation in regression with grouped variables". In: *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B* 68 (2006), pp. 49–67.

[6] Laurent Jacob, Guillaume Obozinski, and Jean-Philippe Vert. "Group Lasso with Overlap and Graph Lasso". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pp. 433–440. ISBN: 9781605585161. DOI: 10.1145/1553374.1553431. URL: https://doi.org/10.1145/1553374.1553431.

[7] Shimon Kogan et al. "Predicting Risk from Financial Reports with Regression". In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Boulder, Colorado: Association for Computational Linguistics, June 2009, pp. 272–280. URL: https://www.aclweb.org/anthology/N09-1031.

[8] J. Friedman, T. Hastie, and R. Tibshirani. *A note on the group lasso and a sparse group lasso*. 2010. arXiv: 1001.0736 [math.ST].

[9] Guillaume Obozinski, Laurent Jacob, and Jean-Philippe Vert. *Group Lasso with Overlaps: the Latent Group Lasso approach*. 2011. arXiv: 1110.0413 [stat.ML].

[10] Martin Jaggi. "Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization". In: ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 1. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 427–435. URL: http://proceedings.mlr.press/v28/jaggi13.html.

[11] Gareth James et al. *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013. URL: https://faculty.marshall.usc.edu/gareth-james/ISL/.

[12] Ryan J. Tibshirani. "A General Framework for Fast Stagewise Algorithms". In: (2014). arXiv: 1408.5801 [stat.ML].

[13] Emanuele Frandi et al. *Fast and Scalable Lasso via Stochastic Frank-Wolfe Methods with a Convergence Guarantee*. 2015. arXiv: 1510.07169 [stat.ML].

[14] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman amp; Hall/CRC, 2015. ISBN: 1498712169.

[15] Simon Lacoste-Julien and Martin Jaggi. *On the Global Linear Convergence of Frank-Wolfe Optimization Variants*. 2015. arXiv: 1511.05932 [math.OC].

[16] Thomas Kerdreux, Fabian Pedregosa, and Alexandre d'Aspremont. *Frank-Wolfe with Subsampling Oracle*. 2018. arXiv: 1803.07348 [math.OC].

7 Appendix

Computation of the gradient for the regression function:

$$f(x) = \frac{1}{2} \|y - \beta X\|^2$$

$$f(x) = \frac{1}{2}(X\beta - y)^T(X\beta - y) = \frac{1}{2}[\beta^T X^T X\beta - \beta^T X^T y - y^T X\beta + y^T y] =$$

$$\frac{1}{2}[\beta^T X^T X\beta - 2\beta^T X^T y + y^T y]$$

.

$$\nabla f(x) = \frac{1}{2}[2X^T X\beta - 2X^T y] = X^T Xx - X^T y = X^T(X\beta - y)$$

7.0.1 Properties of the latent group lasso

In this section we introduce some properties of the latent group lasso, which will be useful to prove consistency results. We denote with V(w), the set of m tuples of vectors v.

Lemma 7.0.1. *For any w, $V(w)$ is non-empty, convex and compact. endlemma*

Lemma 7.0.2. *$w \mapsto \Omega(w)$ is a norm.*

Lemma 7.0.3. *Ω being a norm, the Fenchel dual norm Ω^* defined by:*

$$\forall \alpha \in R^p, \Omega^*(\alpha) = \sup_{w \in R^p} \{w^T \alpha | \Omega(w) \leq 1\}.$$

The Fenchel dual norm $\Omega*$ of Ω satisfies:

$$\forall \alpha \in R^p, \Omega^*(\alpha) = \max_{g \in G} d_g^{-1} \|\alpha_g\|$$

Lemma 7.0.4. ***(second variational formulation)*** *For any $w \in R^p$ we have:*

$$\Omega(w) = \max_{\alpha \in R^p} \alpha^T w \text{ s.t. } \|\alpha_g\| \leq d_g \text{ for all } g \in G.$$

Lemma 7.0.5. ***(third variational formulation)*** *For any $w \in R^p$ we have:*

$$\Omega(w) = \frac{1}{2} \min_{\lambda \in R_+^m} \sum_{i=1}^{p} \frac{w_i^2}{\sum_{g \ni i} \lambda_g} + \sum_{g \in G} d_g^2 \lambda_g$$

Lemma 11 A vector $w \in R^p$ is a solution of Eq.11.

- $-\nabla L(w)/\lambda \in \mathcal{A}(w)$
- w can be decomposed as $w = \sum_{g \in G} v^g$ for some $\bar{v} \in V_G$ with for all $g \in G$: either $v_g \neq 0$ and $\nabla_g L(w) = -\lambda d_g v^g / \|v^g\|$ or $v_g = 0$ and $d_g^{-1} \|\nabla L(w)\| \leq \lambda$.

7.0.2 Choice of weights

Even if in this work the weights have not been taken into account, we think that in general, they play an important role in these kind of problems and so, we breafly discuss about them. The choice of the weights d_g takes into account the discrepancies of sizes of the groups. [5], in the classical group lasso, used $d_g = \sqrt{|g|}$ which yields solutions similar to the ANOVA test under a certain design. In the case of latent group Lasso, since some groups overlap, the problem become complex. In [9] it has been studied that the weights should increase not too quickly with the size of the groups, sometimes larger groups are preferred over unions of smaller ones and they play an important role in the identification of relevant gropus and in the control of the false positive rate.