Overlapping Group Lasso Via ADMM in Python

Daniel Kessler

December 11, 2018

Abstract

In this report, we present a brief summary of the overlapping group lasso, showing how it can be motivated as an extension first of the regular lasso to a group setting, and then as a further generalization thereof. We then provide some background on the Alternating Direction Method of Multipliers (ADMM) algorithm, and show why it is a reasonable choice for solving overlapping group lasso problems. Next, we derive the ADMM algorithm for overlapping group lasso, and present a software implementation in python that implements this algorithm. Finally, we show some experimental results on synthetic data across a variety of parameter settings.

1 Notation

First, we fix notation. Since much of our derivations are based on [1], we will use different notation than is typically deployed in more statistics-oriented treatments of the lasso. Let $A \in \mathbb{R}^{m \times n}$ be a (fixed) design matrix, with m observations and n covariates, and $b \in \mathbb{R}^m$ a vector of observations. We assume that our b follows $b = Ax^\star + \epsilon$, where $x^\star \in \mathbb{R}^p$ is an unknown weight vector, and ϵ are independent and identically distributed errors (for a simple case, we can take them to follow $\mathcal{N}(0, \sigma^2)$ for some fixed, but unknown, σ^2). When norms are not otherwise specified, they are taken to be the 2-norm, i.e., $\|\cdot\| \triangleq \|\cdot\|_2$. We will generally be interested in minimizing the least squares loss, i.e., finding $\hat{x} \in \operatorname{argmin}_x \|Ax - b\|$. Note: A great deal of the treatment below, including that of the lasso, group lasso, overlapping group lasso, and ADMM is taken from [1]. We explicitly cite this text at key points, but we do not cite *every* claim which is based on [1] to avoid cluttering the text.

2 Background: Overlapping Group Lasso

In order to introduce the overlapping group lasso, we will first discuss the regular lasso and then show how it can be extended to the (non-overlapping) group lasso setting. The lasso is a highly popular method that is especially useful in high dimensional settings, i.e., where $n \gg m$. In this setting (presuming A is of full rank), the OLS estimate is no longer uniquely determined, as there exist infinitely many candidate $\hat{x}$ that yield zero loss. Instead, one can instead minimize a *regularized problem* in order to obtain a *sparse solution*. While various (essentially equivalent) formulations of the lasso objective exist, for our purposes we will define the primal lasso problem as

$$\min_x \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1, \tag{1}$$

where $\lambda > 0$ determines the amount of regularization. As $\lambda \to 0$, we see that (1) becomes the OLS problem, and as $\lambda \to \infty$, $x \to 0$. One advantage of the lasso is that it will typically recover a sparse solution, i.e., a solution where the minimizing $\hat{x}$ has many entries that are identically 0.

In the setting where covariates can be organized into groups, as may be natural in many applied settings (e.g., where the covariates are gene expression levels, and genes can be organized based on chromosome or location), we may wish not to simply select useful covariates, but instead to select useful *groups* of covariates. This motivates the use of *group lasso* [5], where we replace the objective in (1) with

$$\min_x \frac{1}{2}\|Ax - b\|_2^2 + \lambda \sum_{i=1}^{N} \|x_i\|_2, \tag{2}$$

where x_i is a sub-vector of x containing only the coefficients corresponding to the i'th group, with $i \in [N]$. Note that when extending to the group lasso, the penalty term no longer involves the 1-norm but instead has the 2-norm. Although this may seem surprising, the 1-norm is separable, i.e., $\left\| \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \right\|_1 = \|a_1\|_1 + \|a_2\|_1$, and this would devolve back to the original lasso. Critically, the 2-norm in the regularizer is not squared, which yields an analogous geometry to the lasso, with singularities corresponding to solutions that are group-sparse (see [5], Fig 1 for a helpful illustration of this phenomena). Note that for a singleton vector, $\|a\|_1 = \|a\|_2$, so when $N = n$, i.e., each feature is alone in its own atomic group, we can rewrite (2) as

$$\begin{aligned} \min_x \frac{1}{2}\|Ax - b\|_2^2 + \lambda \sum_{i=1}^{N} \|x_i\|_2 &= \min_x \frac{1}{2}\|Ax - b\|_2^2 + \lambda \sum_{i=1}^{N} \|x_i\|_1 \\ &= \min_x \frac{1}{2}\|Ax - b\|_2^2 + \lambda \|x\|_1, \end{aligned}$$

and we can recover the original lasso formulation in (1) as a special case of the group lasso.

Finally, the *group lasso* can be further extended to accommodate *overlapping groups* in the "overlapping group lasso" [6, 3]. In this setting, rather than partitioning x into disjoint sub-vectors, we let $G_i, i = [N]$ be an index set holding the indices of coefficients corresponding to the i'th group, i.e., x_{G_i} is a vector of coefficients for group i, x_{G_j} is a vector of coefficients for group j, and it may be the case that $G_i \cap G_j \neq \emptyset$. As a toy example, suppose $n = 3$, and in this simple setting we have two groups, with $G_1 = \{1, 2\}, G_2 = \{2, 3\}$, such that x_2 is common to both x_{G_1} and x_{G_2}, i.e., $G_1 \cap G_2 = \{2\}$. In this setting, the overlapping group lasso objective is given by

$$\min_x \frac{1}{2}\|Ax - b\|_2^2 + \lambda \sum_{i=1}^{N} \|x_{G_i}\|_2. \tag{3}$$

Of course, the (non-overlapping) group lasso discussed above is a special case of the overlapping group lasso, and by transitivity, since lasso is a special case of the group lasso, it is also a special case of the overlapping group lasso.

The geometry of this problem is rather complicated, and there is some work (e.g., [2]) that proposes addressing the overlapping group lasso through latent variables, in essence, performing variable duplication to render the problem non-overlapping, and then using the standard group lasso formulation in (2) to solve the problem. However, in the present work we will focus on directly optimizing the objective given in (3), although as we shall see in Section 4, our algorithmic approach

will involve a sort of variable duplication, but with an update step that pulls our duplicated variables back toward one another.

3 Background: ADMM

The Alternating Direction Method of Multipliers (ADMM) is an algorithmic approach to optimization well suited to solving problems that can be decomposed as the sum of two problems in distinct variables, subject to linear constraints. The background we provide here will closely follow [1], as this was our primary resource when endeavoring to learn the material. Our development here will be terse and limited, and we refer the reader to [1], as the exposition given below chiefly consists of key highlights from this very useful text.

ADMM is formulated to solve problems structured as

$$\begin{aligned} \min_{x,z} \quad & f(x) + g(z) \\ \text{subject to} \quad & Ax + Bz = c \end{aligned}. \tag{4}$$

It is closely related to the method of multipliers (a brief background is given in [1]) and proceeds by first constructing an augmented Lagrangian

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2, \tag{5}$$

where y is a dual variable, and the last term is the "augmenting" piece. Although augmenting may seem unnatural at first, we note that when the linear constraints are satisfied, this last term is identically 0 and thus inconsequential for the objective function at the optimum, and its inclusion makes the use of the `prox` operator natural during the optimization. ADMM is an iterative procedure, which given some initial values for x, z, y, proceeds as

$$x^{k+1} \leftarrow \operatorname*{argmin}_x L_\rho(x, z^k, y^k) \tag{6}$$

$$z^{k+1} \leftarrow \operatorname*{argmin}_x L_\rho(x^{k+1}, z, y^k) \tag{7}$$

$$y^{k+1} \leftarrow y^k + \rho(Ax^{k+1} + Bz^{k+1} - c). \tag{8}$$

Of course, the rub lies in actually solving the sub-problems given in (6)(7), but using the augmented Lagrangian L_ρ makes this tractable for certain problems.

4 Overlapping Group LASSO Via ADMM

There is treatment of lasso, group lasso, and (very brief) mention of overlapping group lasso in §6.4 of [1]. We noted in Section 2 that the group lasso, and in turn lasso, can be recovered as special cases of the overlapping group lasso. However, [1] attempts to make (both overlapping and non-overlapping) group lasso notationally consistent with their later section on consensus learning, which actually yields an algorithm that does not permit recovery of these simpler forms as special cases, since the roles of x and z are reversed. In our treatment here, we have eschewed this notational change and describe an approach that connects more naturally to lasso.

Our goal is to rewrite (3) in a form amenable to application of ADMM but that is still equivalent to the original optimization problem. In a similar spirit to the "latent" approach to overlapping group lasso [2, 4], we will create many *new* variables which in a strict sense do not overlap, but then use linear equality constraints to impose the requirement that their relevant components coincide via a common "anchoring" global variable, which will indirectly enforce equality in the shared components. In particular, we rewrite (3) as,

$$\begin{aligned} \min_{x, z_i, i \in [N]} \quad & \frac{1}{2}\|Ax - b\|_2^2 + \lambda \sum_{i=1}^{N} \|z_i\|_2, \\ \text{such that} \quad & \tilde{x}_i - z_i = 0, \quad \forall i \in [N] \end{aligned} \tag{9}$$

where $\tilde{x}_i \triangleq x_{G_i}$, i.e., the components of x corresponding to group i for $i \in [N]$, and the constraint ensures that the solution satisfies $z_i = x_{G_i}, z_i \in \mathbb{R}^{|G_i|}$ This is now in a form compatible with ADMM, where the second term is analogous to $g(z)$, where z is the concatenation of all the z_i's, and g very naturally decomposes across the z_i, which permits the update of the z_i to be done in parallel.

Now, our task is to give explicit forms for steps (6)(7). Our update can be performed as

$$x^{k+1} \leftarrow (A^T A + \rho I)^{-1}(A^T b + \rho(\bar{z}^{k+1} - \bar{u}^k)) \tag{10}$$

$$z_i^{k+1} \leftarrow \underset{z_i}{\operatorname{argmin}}(\lambda \|z_i\|_2 + \left(u_i^k\right)^T (z_i - \tilde{x}_i^k) + \frac{\rho}{2}\|z_i - \tilde{x}_i^k\|_2^2) \tag{11}$$

$$u_i^{k+1} \leftarrow u_i^k + \tilde{x}_i^{k+1} - z_i^{k+1}, \tag{12}$$

where $\bar{z}, \bar{u}$ are obtained by averaging over z_i, u_i at the relevant components. The operation at (11) is precisely the proximal operator, which for group lasso is the vector soft thresholding operation, as given in §6.4.2 of [1], i.e.,

$$z_i^{k+1} = (1 - \frac{\lambda}{\rho}\|\tilde{x}_i^{k+1} + u^k\|_2^{-1})_+(\tilde{x}_i^{k+1} + u^k) \tag{13}$$

We summarize the approach in Algorithm 1.

5 Software Implementation

We implemented the procedure of Algorithm 1 in python 3.7. The implementation, along with this report, is available on github at `http://github.com/dankessler/608a-project`. The dependency environment can be reconstructed using `pipenv` using the `Pipfile` and `Pipfile.lock` found at the root of the repository.

6 Experimental Results

We conducted numerical experiments in simulated data. A script named `simulations.py` reproducing these results is available on github in the root directory of our repository `http://github.com/dankessler/608a-project`. We generate synthetic data as $y = Ax^\star + \epsilon$, where

Algorithm 1 Overlapping Group LASSO Via ADMM

Require: $A, b, x^0 \in \mathbb{R}^n, \rho, \lambda, G$
$Q \leftarrow (A^T A + \rho I)^{-1}$
for $i \in [N]$ **do**
    $z_i \leftarrow x^0_{G_i}$
    $u_i \leftarrow \{0\}^{|G_i|}$
end for
$k \leftarrow 0$
while not converged **do**
    $x^{k+1} \leftarrow Q\left(A^T b + \rho(\bar{z}^{k+1} - \bar{u}\right)$
    for $i \in [N]$ **do**
        $z_i^{k+1} \leftarrow S_{\lambda/\rho}(\tilde{x}_i^{k+1} + u_i)$
    end for
    for $i \in [N]$ **do**
        $u_i^{k+1} \leftarrow u_i^k + \tilde{x}_i^{k+1} - z_i^{k+1}$
    end for
    $k \leftarrow k + 1$
end while

$\epsilon \sim \mathcal{N}_m(0, I_m), A \in \mathbb{R}^{m \times n}$ is a random matrix with iid standard normal entries, and for all simulations we fix $m = 50, n = 100$. Our three experiments below are under different settings for $x^\star$. In each case, we run our implementation of Algorithm 1 on the synthetic data with $\rho = 1$ and vary λ over a logarithmic grid. Each evaluation yields a regularized estimate $\hat{x}$, and with this we evaluate

1. The prediction error: $\|Ax^\star - A\hat{x}\|_2$
2. The accuracy of the estimated x: $\|x^\star - \hat{x}\|_2$
3. The precision of the recovery of the support: $\frac{\left|\{i:\hat{x}_i \neq 0\} \cap \left\{j:x_j^\star \neq 0\right\}\right|}{\|\hat{x}\|_0}$
4. The recall of the recovery of the support: $\frac{\left|\{i:\hat{x}_i \neq 0\} \cap \left\{j:x_j^\star \neq 0\right\}\right|}{\|x^\star\|_0}$

Because the x iterates in our algorithm never become *truly* sparse, as a heuristic after returning $\hat{x}$ for a fixed number of iterations (here one thousand) we set to 0 any entries of $\hat{x}$ with magnitude smaller than $\epsilon = .01$. This enables more meaningful interpretation for the latter two metrics (since otherwise the set of components of $\hat{x}$ that are nonzero will be all n. These results are depicted graphically as a function of λ in Figures 1, 2, and 3.

6.1 Lasso as a Special Case

As discussed earlier, if we place each covariate into its own atomic group, our problem devolves to the classic lasso. We set $x_i^\star = 10$ for $i \in [25]$, and $x_i^\star = 0$ for $i \in [50] \setminus [25]$. We conduct the simulation approach as described above, and display our results in Figure 1.

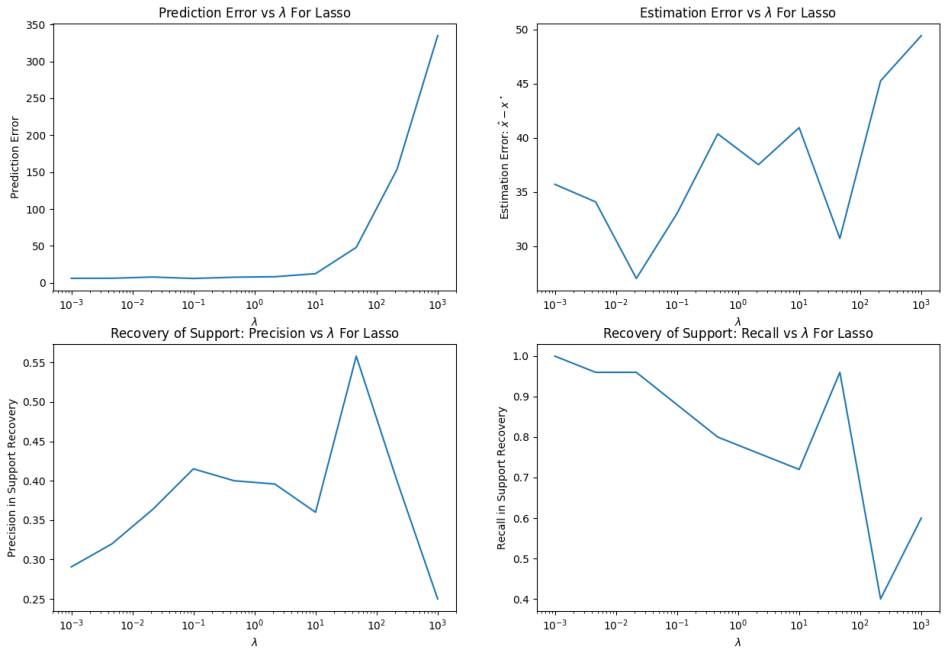

Figure 1: Performance metrics for simulations under lasso. Larger values of λ generally encourage a sparser solution, which seems to yield worse prediction error, estimation error, and recall in recovery of support. Precision initially improves, but this is likely due to a shrinking denominator.

6.2 Non-overlapping Group Lasso as a Special Case

Our set up for (non-overlapping) group lasso is reasonably straightforward. We construct two disjoint groups as $G_1 = [25], G_2 = [50] \setminus G_1$. We set $x_i^\star = 10$ for $i \in G_1$, and $x_i^\star = 0$ for $i \in G_2$. Note that this is the same $x^\star$ as in lasso above, but the penalization is different. The results are presented in Figure 2.

6.3 Overlapping Group Lasso

For overlapping group lasso, we construct 9 groups of equal size that overlap by the same amount. In particular, $G_i = \{5(i-1)+j : j \in [10]\}, i \in [9]$. For example, $G_1 = \{1,2,3,4,5,6,7,8,9,10\}$, $G_2 = \{6,7,8,9,10,11,12,13,14,15\}$, so $G_1 \cup G_2 = \{6,7,8,9,10\}$. As was discussed briefly earlier, our formulation of the overlapping group lasso, unlike that in [2, 4], has singularities corresponding to the complement of unions of groups. For this reason, we set $x_i^\star = 10$ for $i \in \{1,2,3,4,5,16,17,18,19,20\}$ and 0 otherwise. This corresponds to the complement of the union of groups $G_2, G_3, G_5, G_6, G_7, G_8$, and G_9. Results are presented in Figure 3.

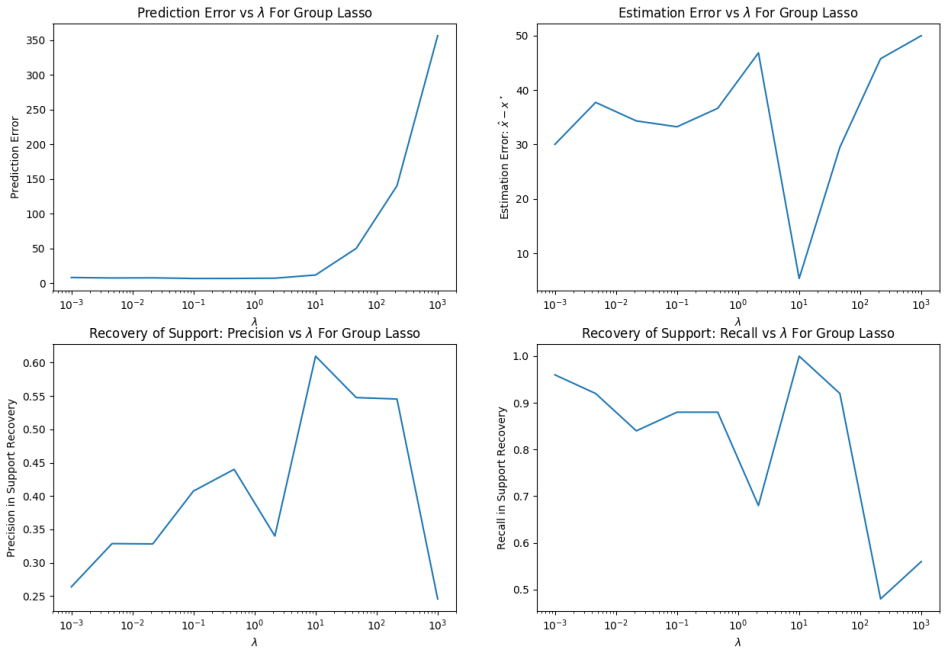

Figure 2: Performance metrics for simulations under (non-overlapping) group lasso. These plots are reasonably similar to those in Figure 1, which is unsurprising since the problem set up is very similar.

7 Conclusion and Future Directions

In this report, we provided a brief introduction to overlapping group lasso, reviewed requisite background materials on ADMM, and then showed how ADMM could be applied to the overlapping group lasso problem. We then conducted numerical experiments on simulated data under three different settings: lasso as a special case of overlapping group lasso, (non-overlapping) group lasso as a special case of overlapping group lasso, and finally a non-trivial overlapping group lasso problem. One important note is that our approach is in contrast to one favored by some statisticians [4, 2]. The reason for this is that the explicit formulation of the overlapping group lasso, as we have implemented, is well positioned for recovery in settings where the true support is the complement of a union of groups. Conceptually, this corresponds to "deactivating" a set of groups, and zeroing out all of their corresponding features, which in the case of overlaps, will result in some "active" groups only being active in components that are not shared with a deactivated group. In some applied settings this may be a reasonably accurate regime, but in settings where it is more natural to countenance the support being the union of *active* groups, an alternative formulation, described in [4, 2], proposes that variables be duplicated to render the problem non-overlapping, but does *not* involve the introduction of linear constraints that require that corresponding entries perfectly

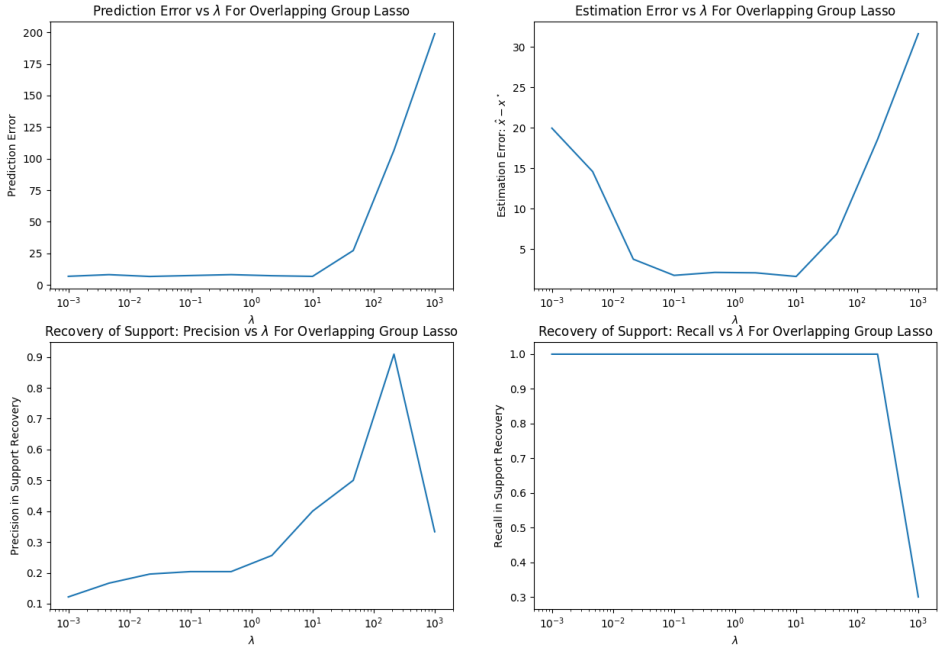

Figure 3: Performance metrics for simulations under overlapping group lasso. As with all our results, increasing λ hurts prediction error. Also, we can infer that we require high values of λ before we obtain solutions that are at all sparse, and this is why recall is initially so high. In terms of estimation error, extreme values of λ (too big or too small) give poor performance, but things are quite good in the middle.

match. This effectively renders the problem non-overlapping. For some of the applied settings in neuroscience where we propose to use our newly developed overlapping group lasso package in python, this latter formulation is more natural. However, the good news is that since we have demonstrated that our approach can solve *both* of these problems, as the latter is a special case of the former, the software that we have developed in this project can be used directly (with some wrapper functionality to handle the bookkeeping involved in variable duplication).

As a next step, we plan to validate the performance of our implementation against a reference implementation. While there exist many implementations of lasso and a few implementations of non-overlapping group lasso, we have yet to find a overlapping group lasso implementation that seems to give sensible results. Nonetheless, if we can validate our implementation against the simpler cases, we may be able to use our package as a benchmark against this other implementations to better understand their shortcomings.

References

[1] Stephen Boyd et al. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends® in Machine Learning* 3.1 (July 26, 2011), pp. 1–122. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000016. URL: http://www.nowpublishers.com/article/Details/MAL-016.

[2] Laurent Jacob, Guillaume Obozinski, and Jean-Philippe Vert. "Group Lasso with Overlap and Graph Lasso". In: *ICML2009*. ICML '09. New York, NY, USA: ACM, 2009, pp. 433–440. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553431. URL: http://doi.acm.org/10.1145/1553374.1553431.

[3] Julien Mairal et al. "Network Flow Algorithms for Structured Sparsity". In: *Advances in Neural Information Processing Systems 23*. Ed. by J. D. Lafferty et al. Curran Associates, Inc., 2010, pp. 1558–1566. URL: http://papers.nips.cc/paper/3965-network-flow-algorithms-for-structured-sparsity.pdf.

[4] Guillaume Obozinski, Laurent Jacob, and Jean-Philippe Vert. "Group Lasso with Overlaps: the Latent Group Lasso approach". In: *arXiv:1110.0413 [cs, stat]* (Oct. 3, 2011). arXiv: 1110.0413. URL: http://arxiv.org/abs/1110.0413.

[5] Ming Yuan and Yi Lin. "Model selection and estimation in regression with grouped variables". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.1 (2006), pp. 49–67. ISSN: 1467-9868. DOI: 10.1111/j.1467-9868.2005.00532.x. URL: http://onlinelibrary.wiley.com/doi/10.1111/j.1467-9868.2005.00532.x/abstract.

[6] Peng Zhao, Guilherme Rocha, and Bin Yu. "The composite absolute penalties family for grouped and hierarchical variable selection". In: *The Annals of Statistics* 37.6 (Dec. 2009), pp. 3468–3497. ISSN: 0090-5364, 2168-8966. DOI: 10.1214/07-AOS584. URL: http://projecteuclid.org/euclid.aos/1250515393.