LASSO WITH CENTRALIZED KERNEL

JENNIFER KAMPE, MAYA BRODY, SOPHIA FRIESENHAHN, ADRINA GARRETT, MARISSA HEGY

Abstract. Least Absolute Shrinkage and Selection Operator (LASSO) regression utilizes an ℓ_1 coefficient constraint to produce high accuracy, sparse models of wide datasets with many regressors. Kernel-based models in the LASSO family offer a further improvement by providing a nonparametric method which can accommodate non-linear models. Roth (2004) proposes a kernelized LASSO using the robust Huber loss function. Our model builds upon this work by adding a centralized kernel. We implement this model as an algorithm including hyperparameter selection in R.

1. Introduction

Consider the regression problem of fitting a model to a large data set with many potentially correlated predictors. In this case, the Ordinary Least Squares (OLS) method will produce unnecessarily complex, overfitted models with poor generalizability. While unbiased, the resultant coefficients will have high standard errors and thus low accuracy. Additionally, a large number of predictors will be retained by the model, making interpretation difficult. The goal is to replace the problematic OLS method with one which produces sparse, prediction-optimized models.

Subset selection algorithms provide a number of methods for determining which variables to include in a regression model, yielding simpler and more easily interpretable models [7]. However, in situations with a large number of regressors, an exhaustive search of the potential feature space may be unfeasible — especially in cases with nonlinear models. Additionally, the coefficients produced by subset selection algorithms are only unbiased if the correct variables are discarded. However, the inclusion of non-predictive regressors increases the prediction error via overfitting, leading to a bias-variance trade-off.

Regularization refers to the technique of addressing overfitting by controlling the complexity of the models produced. Consider a data set $(X, \boldsymbol{y})$, where $X = (\boldsymbol{x}_{i1}, \ldots, \boldsymbol{x}_{ip})^T$ are the regressors, and each y_i is the response for the i^{th} observation, with $i = 1, \ldots, N$. The OLS solution is given by

$$\beta = \arg\min_{\beta} \|\boldsymbol{y} - X\boldsymbol{\beta}\|_2^2. \tag{1.1}$$

The regularization problem is obtained by adding to the above equation a penalty, λ on the regularization term (i.e., the specified measure of model complexity), L.

$$\beta = \arg\min_{\beta} \|\boldsymbol{y} - X\boldsymbol{\beta}\|_2^2 + \lambda L(\boldsymbol{\beta}). \tag{1.2}$$

Ridge regression is defined to be the ℓ_2 specification of the model complexity in the regularization expression above:

$$\beta = \arg\min_{\beta} \|\boldsymbol{y} - X\boldsymbol{\beta}\|_2^2 + \lambda L_2(\boldsymbol{\beta}). \tag{1.3}$$

where $L_2(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_2 = \sum_i (\beta_i)^2$.

Thus, the penalty in a ridge regression model is proportional to the norm of its coefficient vector. This technique shrinks redundant or non-predictive coefficients, but will not typically produce any coefficients exactly equal to zero [2]. This improves prediction accuracy relative to the OLS method, but fails to reduce the number of regressors or to produce a sparse solution.

The Least Absolute Shrinkage and Selection Operator (LASSO) provides a regularization method using an ℓ_1 complexity term.

$$\beta = \arg\min_{\beta} \|\boldsymbol{y} - X\boldsymbol{\beta}\|_2^2 + \lambda L_1(\boldsymbol{\beta}). \tag{1.4}$$

where $L_1(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_1 = \sum_i |\beta_i|$.

Introduced by Tibshirani in 1994, LASSO combines the strategies of the subset selection and ridge regression methods, to produce a method that takes into account both of the major drawbacks of the OLS method [10]. The geometry of the ℓ_1 constraint results in sparse solutions, with many coefficients going to zero; the model is simplified while still retaining prediction accuracy.

1.1. LASSO vs SVM

Both Support Vector Machines (SVM) and LASSO lead to sparse models, but the regularization method in each is very different. In SVM, the loss function does not reflect knowledge about the noise, as it is selected to enforce sparse solutions independent of the noise distribution. LASSO, on the other hand, uses a noise model to choose a loss function. Additionally, in SVM, the data points that are within the ϵ tube around the regression fit have weights of zero and thus do not influence the regression fit. In LASSO, the determination of the optimal coefficients depends on all of the input vectors, so every input vector has influence on the coefficients that are included in the model [10].

1.2. Paper Structure

In §2, we summarize the LASSO problem and introduce kernelized LASSO; we use a Bayesian specification to arrive at a robust loss function and state the resultant robust kernelized LASSO problem. In §3, we outline the K-LASSO algorithm proposed in [9] and provide our method of kernel centering. In §4, we describe model hyperparameters and propose a method for their optimization. In §5 we derive a method of generating predictions from a K-LASSO model. We conclude with suggestions for future work including a potential performance evaluation of the K-LASSO algorithm in comparison to related regression methods.

2. Background

2.1. Definition of LASSO

Let $(X, \boldsymbol{y})$ be a set of data where y_i are the responses for $i = 1, 2, \ldots, N$, and X is the matrix

$$X = \begin{Bmatrix} x_{11} & x_{12} & \ldots & x_{1p} \\ x_{21} & x_{22} & & \\ \vdots & & \ddots & \\ x_{p1} & & & x_{pp} \end{Bmatrix}$$

, previously shown as $X = (\boldsymbol{x}_{i1}, \boldsymbol{x}_{i2}, \ldots, \boldsymbol{x}_{ip})^T$. Further, assume the following: (i) either the observations are independent, or the y_i values are conditionally independent given the x_{ij} values, (ii)the x_{ij} values are standardized so that $\sum_{i=1} \frac{x_{ij}}{N} = 0$ and $\sum_{i=1} \frac{x_{ij}^2}{N} = 1$ [10].

The goal of regression is to construct a model $\boldsymbol{y} = X\hat{\boldsymbol{\beta}} + \boldsymbol{\epsilon}$ based on data. Let $\hat{\boldsymbol{\beta}} = (\hat{\beta}_1, \hat{\beta}_2, \ldots, \hat{\beta}_p)^T$. The LASSO estimate $(\hat{\beta}_0, \hat{\boldsymbol{\beta}})$ is defined by

$$(\hat{\beta}_0, \hat{\boldsymbol{\beta}}) = \min \left[\sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_j \beta_j x_{ij} \right)^2 \right] \quad \text{subject to} \sum_j |\beta_j| \leq t, \tag{2.1}$$

where $t \geq 0$ is a tuning parameter that can be estimated using cross validation [10]. (2.1) shows that to estimate the coefficients $\boldsymbol{\beta}$, the goal is to minimize the squared error of the model. Due to our assumption of standardization, the solution for β_0 is $\hat{\beta}_0 = \bar{y}$ for all values of t, and we can assume, without loss of generality, that $\bar{y} = 0$ so $\hat{\beta}_0 = 0$ and can be omitted. This simplification leaves us with

$$(\hat{\beta}_0, \hat{\boldsymbol{\beta}}) = \min \left[\sum_{i=1}^{N} \left(y_i - \sum_j \beta_j x_{ij} \right)^2 \right] \quad \text{subject to} \sum_j |\beta_j| \leq t \tag{2.2}$$

.

The tuning parameter t controls the amount of shrinkage that is applied to the coefficients in the model. The smaller the value of t, the tighter the constraint and larger the number of coefficients that will be shrunken to zero. As a result, fewer predictors will be included in the model.

2.2. Geometry of LASSO

The geometry of LASSO regression is important for understanding why LASSO often produces coefficients that are equal to zero, which rarely results from ridge regression. This happens because LASSO uses the ℓ_1 constraint $\sum_j |\beta_j| \leq t$, while ridge regression uses the ℓ_2 constraint $\sum_j \beta_j^2 \leq t$. We will use a model with two regressors x_1 and x_2 and their coefficients β_1 and β_2 as an illustrative example.

The criterion $\sum_{i=1}^{N} (y_1 - \sum_j \beta_j x_{ij})^2$ from (2.2) equals the quadratic function $(\boldsymbol{\beta} - \hat{\boldsymbol{\beta}})^T X^T X (\boldsymbol{\beta} - \hat{\boldsymbol{\beta}})$ plus a constant [10]. Fig. 1 shows the elliptical contours of this function in red, which are centered at the OLS estimates. On the left-hand graph, the shaded diamond is the constraint region determined by our LASSO constraint $\sum_j |\beta_j| \leq t$, which in this case is just $|\beta_1| + |\beta_2| \leq t$. Note that each corner of the diamond corresponds to a zero for one of the coefficients, meaning the coefficient is eliminated from the model. The LASSO solution is the first place that the red contours touch the diamond, which often occurs at a corner of the diamond.

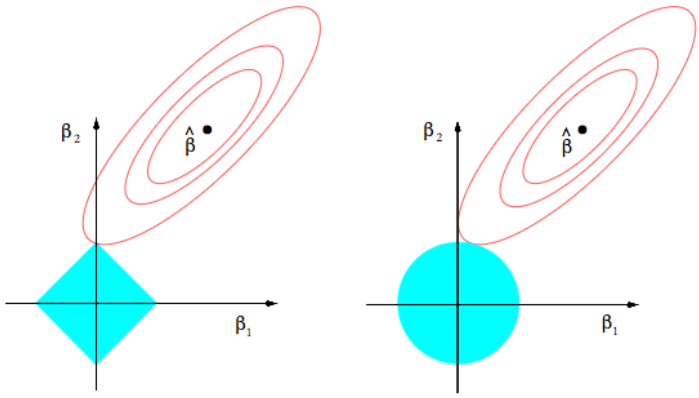

FIGURE 1. Solutions for LASSO (left) and ridge (right) regression models with two coefficients $\boldsymbol{\beta_1}$ and $\boldsymbol{\beta_2}$ [1].

In the right-hand plot, the shaded circle is the constraint region determined by the ridge regression constraint $\sum_j \beta_j^2 \leq t$. Because the ridge constraint region has no corners, solutions with a zero coefficient are much less likely to occur. For this reason, LASSO regression produces much sparser models than does ridge regression. For more complex models with more than two regressors, the constraint region for LASSO will be a hyper-diamond with many corners, while the constraint region for ridge will be a hyper-sphere with no corners.

2.3. Kernelized LASSO

Kernelized LASSO utilizes the same ℓ_2 norm as the traditional LASSO method, but with a kernel regressor matrix. The optimization problem for kernelized LASSO is

$$(\hat{\beta}_0, \hat{\boldsymbol{\beta}}) = \min \|\boldsymbol{y} - K\boldsymbol{\beta}\|_2^2 \quad \text{subject to} \|\boldsymbol{\beta}\|_1 \leq t, \tag{2.3}$$

where K is the kernel matrix

$$K = (k(\boldsymbol{x}_i, \boldsymbol{x}_1), k(\boldsymbol{x}_i, \boldsymbol{x}_2), \ldots, k(\boldsymbol{x}_i, \boldsymbol{x}_N)),$$

and for some kernel function ϕ,

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j})).$$

Like SVM, kernelized LASSO (K-LASSO) is a kernel-based learning algorithm that utilizes the kernel trick, where the dot product is taken between pairs of points mapped to a higher dimensional feature space without having to know the coordinates in the feature space. This method is less computationally intensive because it works in the lower dimensional input space. Note that in kernelized LASSO, the inner product is taken between rows of the matrix X to form the kernel matrix. In this paper, we will use the Radial Basis Function (RBF) kernel with the hyperparamter γ:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2), \qquad \gamma > 0.$$

K-LASSO offers several advantages to SVM, such as giving probabilistic outputs, which SVM lacks. K-LASSO can also handle very large data sets, while SVM suffers from a "steep growth of the number of support vectors" as the size of the training

set increases [9]. Finally, K-LASSO produces even sparser solutions than SVM does when dealing with large-scale problems, allowing for more efficient predictions.

2.4. Bayesian Framework

As proposed in [9], we will link the LASSO to a Bayesian regression model through the use of Automatic Relevance Determination (ARD) priors. Using this probabilistic framework makes it possible to estimate the prediction variance. In order to apply this Bayesian method to our LASSO regression problem, we must define a set of probabilistic models of the data.

Given a hypothesis, or prediction, H with prior probability $p(H)$, the likelihood of H is $p(H|D)$, where D is the data, given in pairs of inputs and outputs $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$. Assuming that the outputs y are generated by corrupting the values of the regression function f with Gaussian noise with variance σ^2, the likelihood of H is given by:

$$\begin{aligned}\prod_i p(y_i \mid \boldsymbol{x}_i, H) &= \prod_i \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{(y_i - f(\boldsymbol{x}_i))^2}{2\sigma^2} \right\} \\ &= (2\pi\sigma^2)^{-\frac{N+1}{2}} \exp\left\{ -\sum_i \frac{(y_i - f(\boldsymbol{x}_i))^2}{2\sigma^2} \right\}\end{aligned} \tag{2.4}$$

In regression problems, each H corresponds to a regression function $f(\boldsymbol{x})$, which in K-LASSO is $K\boldsymbol{\beta}$ [9].

The ARD prior is given by:

$$p(\boldsymbol{\beta} \mid \boldsymbol{\eta}') \propto \exp\left\{ -\sum_i \boldsymbol{\eta}_i' \boldsymbol{\beta}_i^2 \right\} \tag{2.5}$$

In this case, each expansion coefficient has its own prior variance $(\eta_i')^{-1}$ [9]. This prior and the likelihood of H give us a posterior that we will transform into a minimization problem by the following procedure.

$$\begin{aligned} M(\boldsymbol{\beta}) &= -\log(\text{likelihood} \cdot \text{prior}) \\ &= -\log\left((2\pi\sigma^2)^{-\frac{N+1}{2}} \exp\left\{ -\sum_i \eta_i' \beta_i^2 + \frac{(y_i - f(x_i))^2}{2\sigma^2} \right\} \right) \\ &= -\log\left(\exp\left\{ -\sigma^2 \sum_i \eta_i' \beta_i^2 + (y_i - f(x_i))^2 \right\} \right) \\ &= \sigma^2 \sum_i \eta_i' \beta_i^2 + (y_i - f(x_i))^2 \\ &= \sigma^2 \sum_i \eta_i' \beta_i^2 + \sum_i (y_i - f(x_i))^2) \\ &= \|\boldsymbol{y} - K\boldsymbol{\beta}\|^2 + \sigma^2 \sum_i \eta_i' \beta_i^2 \end{aligned} \tag{2.6}$$

We view the equivalence between LASSO and Adaptive Ridge Regression (AdR) in the following way. The AdR procedure minimizes (2.6) subject to the constraint

$$\frac{1}{N+1}\sum_{i=1}^{N+1}\frac{1}{\eta_i}=\frac{1}{\lambda}, \tag{2.7}$$

where λ is exogenous and $\eta_i=\eta_i^{'}\sigma^2$ [9]. Taking a Bayesian viewpoint of regression, the following marginalization procedure gives a LASSO-type functional in the form of an AdR problem in which the hyperparameters do not appear. Consider the exponential hyperpriors

$$p(\eta_i)=\frac{\rho}{2}\exp\left\{-\frac{\rho\eta_i}{2}\right\} \tag{2.8}$$

and the corresponding prior distributions

$$p(\beta_i)=\int_0^\infty p(\beta_i|\eta_i)p(\eta_i)\partial\eta_i=\frac{\rho}{2}\exp\{-\sqrt{\rho}|\beta_i|\}. \tag{2.9}$$

Recall the likelihood (2.4), which together with (2.9) gives the functional

$$\begin{aligned}
M^{LASSO} &= -\log\left(\left(\frac{1}{2\pi\sigma^2}\right)^N\exp\left\{\frac{\sum_{i=1}^N(y_i-f(x_i))^2}{2\sigma^2}\right\}\right)\\
&\qquad -\log\left(\frac{\sqrt{\rho}}{2}\exp\left\{-\sqrt{\rho}\sum_{i=1}^N|\beta_i|\right\}\right)\\
&= \frac{\sum_{i=1}^N(y_i-f(x_i))^2}{2\sigma^2}+\sqrt{\rho}\sum_{i=1}^N|\beta_i|\\
&= \frac{1}{2\sigma^2}\sum_{i=1}^N(y_i-f(x_i))^2+\sqrt{\rho}\sum_{i=1}^N|\beta_i|\\
&= \|\boldsymbol{y}-K\boldsymbol{\beta}\|_2^2+\tilde{\lambda}\|\boldsymbol{\beta}\|_1
\end{aligned} \tag{2.10}$$

Thus, with $\tilde{\lambda}=\sqrt{\rho}$, we obtain the result

$$M_{\text{LASSO}}(\boldsymbol{\beta})=\|\boldsymbol{y}-K\boldsymbol{\beta}\|_2^2+\sqrt{\rho}\|\boldsymbol{\beta}\|_1. \tag{2.11}$$

We can get rid of the constant values $(\frac{1}{2\pi\sigma^2})^N$, $\frac{1}{2}$, and $\frac{\sqrt{\rho}}{2}$ when we take the negative log because we only need the proportional equation [9]. Thus we can view the LASSO problem from a Bayesian perspective to see it as a type of AdR. We will use this in the following section to construct a robust version of the kernelized LASSO optimization problem.

2.5. Robust Kernelized LASSO

For practical application of the kernelized LASSO method, the true noise density of the data set used often cannot be assumed to have a Gaussian distribution and may only be assumed to be partially Gaussian and partially arbitrarily distributed. In such a case, it is preferable to use Huber's robust loss function,

$$L(z)=\begin{cases}c|z|-\frac{c^2}{2} & \text{for } |z|>c\\ \frac{z^2}{2} & \text{for } |z|\le c\end{cases} \tag{2.12}$$

as opposed to a quadratic loss function, as Huber's loss function penalizes large deviations only linearly while penalizing small deviations quadratically. Thus we will rewrite the kernelized LASSO problem to include Huber's loss function.

To do so, we first take the robust complement of (2.6):

$$M_{rob} = \sum_{i=1}^{N} L(y_i - (K_\tau \boldsymbol{\beta}_\tau)_i) + \boldsymbol{\beta}_\tau^T \operatorname{diag}\{\boldsymbol{\eta}_\tau\}\boldsymbol{\beta}_\tau. \tag{2.13}$$

Computing the partial derivatives of (2.13) obtains the following gradient:

$$\nabla_{\boldsymbol{\beta}} M_{rob} = K_\tau^T \Omega(\boldsymbol{\beta}_\tau) y - K_\tau^T \Omega(\boldsymbol{\beta}_\tau) K_\tau \boldsymbol{\beta}_\tau + 2 \operatorname{diag}\{\boldsymbol{\eta}_\tau\}\boldsymbol{\beta}_\tau \tag{2.14}$$

where $\Omega(\boldsymbol{\beta}_\tau) = \operatorname{diag}\{\omega([\boldsymbol{y} - K_\tau \boldsymbol{\beta}_\tau]_i)\}$ and $\omega(u) = \frac{\partial L(u)}{u \partial u}$. The gradient must vanish in the optimal solution, so setting (2.14) to zero and solving for $\boldsymbol{\beta}_\tau$ yields the updated estimate:

$$\boldsymbol{\beta}_\tau^{new} = [K_\tau^T \Omega(\boldsymbol{\beta}_\tau) K_\tau + 2 \operatorname{diag}\{\boldsymbol{\eta}_\tau\}]^{-1} K_\tau^T \Omega(\boldsymbol{\beta}_\tau) \boldsymbol{y}, \tag{2.15}$$

assuming $0 < \omega(u) \leq \omega(0) < \infty$ in order to enable convergence. Note that (2.15) defines the normal equations of a least squares problem with a design matrix $\tilde{K}_\tau = \Omega^{1/2} K_\tau$ and dependent variables $\tilde{\boldsymbol{y}} = \Omega^{1/2} \boldsymbol{y}$. Therefore the robust kernelized LASSO optimization problem is given by:

$$\min \|\tilde{\boldsymbol{y}} - \tilde{K}_\tau \boldsymbol{\beta}_\tau\|_2^2 \quad \text{s.t. } \|\boldsymbol{\beta}\|_1 \leq t, \tag{2.16}$$

where $(\tilde{\boldsymbol{y}}, \tilde{K_\tau}) = (\Omega^{1/2}\boldsymbol{y}, \Omega^{1/2} K_\tau)$ [9].

2.6. Standard Errors

Recall that statistical inference on an estimate requires the computation of Mean Squared Error (MSE) where:

$$MSE(\hat{\beta}_j) = E_{\hat{\beta}_j}\left[(\hat{\beta}_j - \beta_j)^2\right] = \operatorname{Var}_{\hat{\beta}_j} + \operatorname{Bias}\big(\hat{\beta}_j, \beta_j\big)^2.$$

Because penalized regression reduces variance at the expense of bias, the latter is likely to be a large component of MSE. Further, since there is no closed form for the LASSO estimators, the estimation of standard errors has proven difficult. There is no consensus on the appropriate standard error estimators to be used with the LASSO estimators although methods based on an approximated covariance matrix, bootstrapping, and the Empirical Bayesian approach have been proposed. Tibshirani (1996) suggests an approximate covariance matrix based on ridge regression. In this approach and many others based on the covariance matrix, however, the standard errors generated are not valid for coefficients with $\hat{\beta}_j = 0$, yielding a biased standard error identical to zero [10].

Given the difficulty of estimating the covariance matrix, bootstrap standard errors offer a potential solution which is distribution free, generated by sampling with replacement from the observed data. However, Leeb and Potscher find that in general, boostrap standard errors of shrinkage estimators like LASSO can be unreliable [6]. Additionally, the problem of biased standard errors for $\hat{\beta}_j = 0$ persists even when using bootstrapping [5]. Further, it is important to note that bootstrap standard errors can only estimate the variance of the LASSO estimators and cannot address bias, the other component of Mean Squared Error.

Given the drawbacks of each class of standard error estimators, it is suggested that they be omitted from penalized regression packages to avoid giving the user an inaccurate perception of estimate precision. Note that the `R` packages `GLMNET` and `penalized` indeed do not provide standard errors.

Roth (2004) proposes another method for statistical inference. Given an optimal $\bar{\boldsymbol{\beta}}$ found using the algorithm, the uncertainty of a prediction $\bar{f}(\boldsymbol{x}_*)$ given a new observation $\boldsymbol{x}_*$ with feature vector $\boldsymbol{\phi}_\tau(\boldsymbol{x}_*)$ can be measured based on the variance:

$$\text{var}[f(\boldsymbol{x}_*)] = \sigma^2 \boldsymbol{\phi}_\tau^T(\boldsymbol{x}_*)\big[\tilde{K}_\tau^T \tilde{K}_\tau + 2\text{diag}\{\boldsymbol{\eta}_\tau\}\big]\boldsymbol{\phi}_\tau(\boldsymbol{x}_*) \tag{2.17}$$

where σ^2 is an estimate of the variance of the Gaussian noise in the transformed problem [9].

3. Algorithm

To solve the optimization problem stated in (2.16), we implement an algorithm proposed in Roth (2004) [9]. In this algorithm, we incrementally add a vector $\boldsymbol{h}$ to the vector of estimated coefficients $\boldsymbol{\beta}$, indexed by N. Here $\boldsymbol{h}$ represents our search direction locally around our current estimation of $\boldsymbol{\beta}$. Because the goal is convergence, some elements of $\boldsymbol{h}$ will be zero as to leave our corresponding $\boldsymbol{\beta}$ elements unchanged. Thus, $\boldsymbol{h}$ is formally defined as $\boldsymbol{h} := P^T \begin{pmatrix} \boldsymbol{h}_\tau \\ \boldsymbol{0} \end{pmatrix}$, where P is a permutation matrix that is used to collect the $|\tau|$ non-zero elements of $\boldsymbol{\beta}$, i.e. $\boldsymbol{\beta} = P^T \begin{pmatrix} \boldsymbol{\beta}_\tau \\ 0 \end{pmatrix}$. Thus, our local optimization problem is:

$$\min_{\boldsymbol{h}} \|\tilde{\boldsymbol{y}} - \tilde{K}(\boldsymbol{\beta} + \boldsymbol{h})\|_2^2 \qquad \text{s.t. } \boldsymbol{\theta}_\tau^T(\boldsymbol{\beta}_\tau + \boldsymbol{h}_\tau) \le t \text{ and } \boldsymbol{h} = P^T \begin{pmatrix} \boldsymbol{h}_\tau \\ \boldsymbol{0} \end{pmatrix}. \tag{3.1}$$

where the linear constraint is proven to be equivalent to the typical ℓ_1 constraint in (3.4) for small enough steps in the direction $\boldsymbol{h}$.

By the Karush-Kuhn-Tucker conditions, we have

$$\begin{pmatrix} \tilde{K}_\tau^T \tilde{K}_\tau & \boldsymbol{\theta}_\tau \\ \boldsymbol{\theta}_\tau^T & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{h}_{\boldsymbol{\tau}} \\ \mu \end{pmatrix} \begin{pmatrix} \tilde{K}_\tau^T(\tilde{\boldsymbol{y}} - \tilde{K}_\tau \beta_\tau) \\ t - \boldsymbol{\theta}_\tau^T \boldsymbol{\beta}_\tau \end{pmatrix} \tag{3.2}$$

which we obtain in the following process. The Lagrangian of our objective function (3.1) can be given as:

$$\mathcal{L} = \|\tilde{\boldsymbol{y}} - \tilde{K}_\tau(\boldsymbol{\beta}_{\boldsymbol{\tau}} + \boldsymbol{h}_\tau)\|_2^2 - \mu'(t - \boldsymbol{\theta}_\tau^T(\boldsymbol{\beta}_\tau + \boldsymbol{h}_\tau)) \tag{3.3}$$

where $\mu' \ge 0$. Taking the partial in terms of $\boldsymbol{h}$ yields:

$$\partial \mathcal{L}_h = -\tilde{\boldsymbol{r}}\tilde{K}_\tau + \boldsymbol{h}_\tau \tilde{K}_\tau^T \tilde{K}_\tau + \mu \boldsymbol{\theta}_\tau^T \tag{3.4}$$

where $\mu = \frac{\mu'}{2}$ and $\tilde{\boldsymbol{r}} = \tilde{\boldsymbol{y}} - \tilde{K}\boldsymbol{\beta}$. By the KKT conditions, (3.4) must equal zero, so we can solve for $\boldsymbol{h}_\tau$ and μ:

$$\boldsymbol{h}_\tau = (\tilde{K}_\tau^T \tilde{K}_\tau)^{-1}(\tilde{K}_\tau^T(\tilde{\boldsymbol{y}} - \tilde{K}_\tau \boldsymbol{\beta}_\tau) - \mu \boldsymbol{\theta}_\tau^T) \tag{3.5}$$

$$\begin{aligned}
\boldsymbol{\theta}_\tau^T \boldsymbol{h}_\tau &= t - \boldsymbol{\theta}_\tau^T \boldsymbol{\beta}_\tau \\
\boldsymbol{\theta}_\tau^T (\tilde{K}_\tau^T \tilde{K}_\tau)^{-1}(\tilde{K}_\tau^T(\tilde{\boldsymbol{y}} - \tilde{K}_\tau \boldsymbol{\beta}_\tau) - \mu \boldsymbol{\theta}_\tau^T) &= t - \boldsymbol{\theta}_\tau^T \boldsymbol{\beta}_\tau \\
\boldsymbol{\theta}_\tau^T (\tilde{K}_\tau^T \tilde{K}_\tau)^{-1}(\tilde{K}_\tau^T \tilde{\boldsymbol{y}} - \tilde{K}_\tau^T \tilde{K}_\tau \boldsymbol{\beta}_\tau - \mu(\boldsymbol{\theta}_\tau^T) &= t - \boldsymbol{\theta}_\tau^T \boldsymbol{\beta}_\tau \\
\boldsymbol{\theta}_\tau^T ((\tilde{K}_\tau^T \tilde{K}_\tau)^{-1} \tilde{K}_\tau^T \tilde{\boldsymbol{y}} - \boldsymbol{\beta}_\tau - (\tilde{K}_\tau^T \tilde{K}_\tau)^{-1} \mu \boldsymbol{\theta}_\tau) &= t - \boldsymbol{\theta}_\tau^T \boldsymbol{\beta}_\tau \\
\boldsymbol{\theta}_\tau^T (\tilde{K}_\tau^T \tilde{K}_\tau)^{-1} \tilde{K}_\tau^T \tilde{\boldsymbol{y}} - \boldsymbol{\theta}_\tau^T \boldsymbol{\beta}_\tau - \mu(\boldsymbol{\theta}_\tau^T (\tilde{K}_\tau^T \tilde{K}_\tau)^{-1} \boldsymbol{\theta}_\tau) &= t - \boldsymbol{\theta}_\tau^T \boldsymbol{\beta}_\tau \\
\max\left(0, \frac{\boldsymbol{\theta}_\tau^T (\tilde{K}_\tau^T \tilde{K}_\tau)^{-1} \tilde{K}_\tau^T \tilde{\boldsymbol{y}} - t}{\boldsymbol{\theta}_\tau^T (\tilde{K}_\tau^T \tilde{K}_\tau)^{-1} \boldsymbol{\theta}_\tau}\right) &= \mu
\end{aligned} \tag{3.6}$$

where $\boldsymbol{\beta}$ is fixed, $\tilde{\boldsymbol{y}}$ and $\tilde{K}$ are defined with respect to $\Omega(\boldsymbol{\beta}+\boldsymbol{h})$, and $\boldsymbol{\theta}_\tau$ is the sign vector $\boldsymbol{\theta}_\tau := \operatorname{sign}(\boldsymbol{\beta}_\tau)$.

The first step of the algorithm is checking for sign feasibility. For $\boldsymbol{\beta^\dagger} := \boldsymbol{\beta}+\boldsymbol{h}$ to be sign feasible, we must have $\operatorname{sign}(\boldsymbol{\beta}^\dagger_\tau) = \boldsymbol{\theta}_\tau$. If this is not the case, the following steps in Part A are used to correct this.

Part A

(1) Find the smallest $\gamma \in (0,1)$ such that $0 = \beta_k + \gamma h_k$ for some $k \in \tau$ (i.e. $\beta_k \neq 0$) and define $\boldsymbol{\beta'} = \boldsymbol{\beta}+\gamma\boldsymbol{h}$. Essentially, this sets the element of $\boldsymbol{\beta}$ that was most likely causing the feasibility issue, β_k, to zero.
(2) Now set $\theta'_k = -\theta_{\mathbf{k}}$ (i.e. switch the sign of β_k) and compute a new value of $\boldsymbol{h}$, which we will label $\boldsymbol{h'}$ for clarity. If $\boldsymbol{\beta'}+\boldsymbol{h'}$ is sign feasible, set $\boldsymbol{\beta}^\dagger = \boldsymbol{\beta'}+\boldsymbol{h'}$ and move on to the second half of the algorithm (i.e. skip steps 3 and 4 and proceed to Part B. Otherwise, proceed to step 3.
(3) Remove the k used above from τ, and update β_k and θ_k accordingly.
(4) Recompute $\boldsymbol{h}$ for this revised problem and iterate on Part A until a sign feasible $\boldsymbol{\beta^\dagger}$ is obtained.

Part B

(5) Once a sign feasible $\boldsymbol{\beta}^\dagger$ is obtained, test its optimality by computing the following:
$$\boldsymbol{v^\dagger} = \frac{\tilde{K}^T\tilde{\boldsymbol{r}}^\dagger}{\|\tilde{K}^T_\tau\tilde{\boldsymbol{r}}^\dagger\|_\infty} = P^T\begin{pmatrix}\boldsymbol{v_1^\dagger}\\ \boldsymbol{v_2^\dagger}\end{pmatrix} \tag{3.7}$$
where $\tilde{\boldsymbol{r}}^\dagger = \tilde{\boldsymbol{y}}-\tilde{K}\boldsymbol{\beta}^\dagger$. Recall that $(\boldsymbol{v_1^\dagger})_i = \theta_i$ for $i \leq |\tau|$. If $-1 \leq (v_2^\dagger)_j \leq 1$ also holds for $1 \leq j \leq N-|\tau|$, then $\boldsymbol{\beta}^\dagger$ is optimal. Otherwise, continue to Part C. A more detailed explanation of this procedure can be found in §3.2.

Part C

(6) If $\boldsymbol{\beta}^\dagger$ was not the optimal solution, identify the most violated condition by finding the index s such that $(\boldsymbol{v}_2^\dagger)_s$ has the greatest absolute value.
(7) Update the index set τ by adding the indices from $1,\ldots,s$ that are in $N-|\tau|$ to it, and update $\boldsymbol{\beta}^\dagger_\tau$ and $\boldsymbol{\theta}_\tau$ accordingly. Append a zero to the end of $\boldsymbol{\beta}^\dagger_\tau$, and append $\operatorname{sign}(\boldsymbol{v}_2^\dagger)_s$ to the end of $\boldsymbol{\theta}_\tau$. Now update $\tilde{K_\tau}$ and $\tilde{\boldsymbol{y}}$ accordingly.
(8) Set $\boldsymbol{\beta} = \boldsymbol{\beta^\dagger}$, compute a new search direction $\boldsymbol{h}$, and iterate from the beginning of A.

3.1. Initialization

To initialize the algorithm and select the first index set τ, we choose the OLS estimate $\hat{\boldsymbol{\beta}}^\dagger = (\tilde{K}^T\tilde{K})^{-1}\tilde{K}^T\tilde{\boldsymbol{y}}$ as the initial $\hat{\boldsymbol{\beta}}^\dagger$ estimate. We then proceed immediately to Parts B and C of the algorithm to determine the index s, assuming that τ is initially empty. Once s is found, set $\tau = \{1,\ldots,s\}$, and update $\boldsymbol{\beta}^\dagger_\tau$ and $\boldsymbol{\theta}_\tau$ accordingly. As in Part C, append a zero to the end of $\boldsymbol{\beta}^\dagger_\tau$ and $\operatorname{sign}(\boldsymbol{v}_2^\dagger)$ to the end of $\boldsymbol{\theta}_\tau$. Compute $\boldsymbol{h}$ and proceed to Part A.

3.2. Further Explanation of Algorithm

Fig. 2 below presents the general organization of the algorithm.

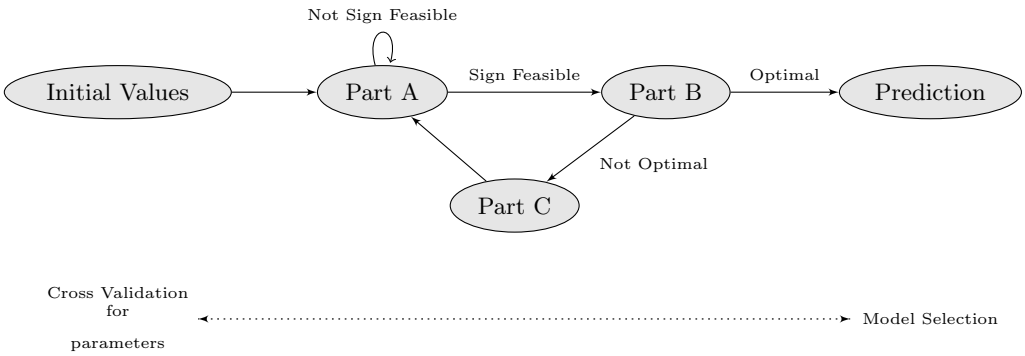

FIGURE 2. Algorithm outline.

Part B determines if $\boldsymbol{\beta}^{\dagger}$ is the optimal solution. We assume the loss function is differentiable, and therefore write the Lagrangian of the kernelized LASSO optimization problem as

$$\mathcal{L}(\boldsymbol{\beta}, \tilde{\lambda}) = \|\tilde{\boldsymbol{y}} - \tilde{K}\boldsymbol{\beta}\|_2^2 - \tilde{\lambda}(t - \|\boldsymbol{\beta}\|_1). \tag{3.8}$$

By the Karush-Kuhn-Tucker conditions, the partial derivative $\partial_{\boldsymbol{\beta}}\mathcal{L}$ must vanish at the optimal solution, i.e.

$$\partial_{\boldsymbol{\beta}}\mathcal{L} = -\tilde{K}^T\tilde{\boldsymbol{r}} + \tilde{\lambda}\boldsymbol{v} \stackrel{!}{=} \boldsymbol{a}, \tag{3.9}$$

$$\text{where } v_i = \begin{cases} \text{sign}(\beta_i) & \text{if } \beta_i \neq 0 \\ a_i \in [-1, 1] & \text{if } \beta_i = 0, \end{cases} \tag{3.10}$$

where $\tilde{\boldsymbol{r}} = \tilde{\boldsymbol{y}} - \tilde{K}\boldsymbol{\beta}^{\dagger}$. Note that $\|\boldsymbol{v}\|_\infty = 1$, which implies that $\tilde{\lambda} = \|\tilde{K}^T\tilde{\boldsymbol{r}}\|_\infty$.

3.3. Kernel Centering

For the above algorithm to be computationally feasible, we must center the kernel matrix and thereby eliminate the intercept β_0 because an intercept term produces an ill-conditioned kernel matrix. To center an $n \times n$ kernel matrix we must center the data set elements as they are mapped in the feature space as follows [3]:

$$K_{\text{center}} = (\boldsymbol{\phi} - \bar{\boldsymbol{\phi}}\mathbf{1}^T)(\boldsymbol{\phi} - \bar{\boldsymbol{\phi}}\mathbf{1}^T) \tag{3.11}$$

$$= (\boldsymbol{\phi} - \frac{1}{n}\boldsymbol{\phi}\mathbf{1}\mathbf{1}^T)^T(\boldsymbol{\phi} - \frac{1}{n}\boldsymbol{\phi}\mathbf{1}\mathbf{1}^T) \tag{3.12}$$

$$= (\boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T)\boldsymbol{\phi}^T\boldsymbol{\phi}(\boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T) \tag{3.13}$$

$$= H \times K \times H \tag{3.14}$$

where the centering matrix H is defined as

$$H = \boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T \tag{3.15}$$

where $\mathbf{1}$ is of length n. Expanding (3.11) we get

$$K_{\text{center}} = K - \frac{1}{n}\mathbf{1}\mathbf{1}^T K - \frac{1}{n}K\mathbf{1}\mathbf{1}^T + \frac{1}{n}\mathbf{1}\mathbf{1}^T K \mathbf{1}\mathbf{1}^T \frac{1}{n}, \tag{3.16}$$

which implies that the i, j^{th} element of K_{center} is

$$k_c(i,j) = k(i,j) - \frac{1}{\tau}\sum_{r=1}^{n} k(i,r) - \frac{1}{\tau}\sum_{r=1}^{n} k(j,r) + \frac{1}{\tau^2}\sum_{r=1}^{n}\sum_{s=1}^{n} k(r,s). \tag{3.17}$$

3.4. Convergence

Proof of the convergence of the algorithm consists of two cases:

If the current estimate $\boldsymbol{\beta}^\dagger$ is not sign feasible and is suboptimal, then in (3.1), it is implied that $\boldsymbol{h}$ is a descent direction. Thus, the objective function (3.1) is reduced in each step, and cycling cannot occur. This process must converge because the possible configurations of τ are finite. Therefore, the procedure must converge, and the final $\boldsymbol{\beta}$ must be sign feasible. Note that if $\boldsymbol{\beta}^\dagger$ is optimal for (3.1), then this part of the algorithm is bypassed[8].

If the current estimate $\boldsymbol{\beta}^\dagger$ is sign feasible and not optimal, then the augmented vector $(\boldsymbol{\beta}_\tau^\dagger, \mathbf{0})^T$ must also be suboptimal for (3.1). We update τ by adding s and augment $\boldsymbol{\theta}_\tau$ as $(\boldsymbol{\theta}_\tau, \theta_s)^T$. Then the solution for the augmented problem is $(\boldsymbol{h}_\tau, h_s)^T$, which is a descent direction for the augmented problem [8].

By definition of descent direction, we have:

$$\begin{aligned}
0 &> \begin{pmatrix} \boldsymbol{h}_\tau \\ h_s \end{pmatrix}^T (\tilde{\boldsymbol{y}} - \tilde{K}(\boldsymbol{\beta} + \boldsymbol{h}))(-2)(-\tilde{K}) \\
&> -\begin{pmatrix} \boldsymbol{h}_\tau \\ h_s \end{pmatrix}^T (\tilde{\boldsymbol{y}} - \tilde{K}\boldsymbol{\beta}^\dagger)(\tilde{K}) \\
&> -\begin{pmatrix} \boldsymbol{h}_\tau \\ h_s \end{pmatrix}^T (\tilde{\boldsymbol{y}} - \tilde{K}\boldsymbol{\beta}^\dagger) \begin{pmatrix} \tilde{K}_\tau \\ \boldsymbol{\phi}_s \end{pmatrix} \\
&> -\begin{pmatrix} \tilde{K}_\tau \\ \boldsymbol{\phi}_s \end{pmatrix}^T (\tilde{\boldsymbol{r}}^\dagger)^T \begin{pmatrix} \boldsymbol{h}_\tau \\ h_s \end{pmatrix} \\
&> -(\tilde{\boldsymbol{r}}^\dagger)^T \tilde{K}_\tau \boldsymbol{h}_\tau - (\tilde{\boldsymbol{r}}^\dagger)^T \boldsymbol{\phi}_s h_s \\
&> -\mu \boldsymbol{\theta}_\tau^T \boldsymbol{h}_\tau - \mu (v_2^\dagger)_s h_s \\
&> -\mu(\boldsymbol{\theta}_\tau^T \boldsymbol{h}_\tau + (v_2^\dagger)_s h_s)
\end{aligned} \tag{3.18}$$

.

For (3.1) to be feasible, we must have $\boldsymbol{\theta}_\tau^T \boldsymbol{h}_\tau + \theta_s h_s \leq 0$ [9]. We can multiply by μ and add (3.18) to obtain:

$$0 \leq \mu(\boldsymbol{\theta}_\tau^T \boldsymbol{h}_\tau + \theta_s h_s) - \mu(\boldsymbol{\theta}_\tau^T \boldsymbol{h}_\tau + (v_2^\dagger)_s h_s) \tag{3.19}$$

$$\leq \mu(\theta_s h_s - (v_2^\dagger)_s h_s) \tag{3.20}$$

$$\leq (\theta_s - (v_2^\dagger)_s) h_s. \tag{3.21}$$

To show that $\theta_s = \text{sign}(h_s)$, set $\theta_s = \text{sign}((v_2^\dagger)_s)$ in (3.21). First suppose $\text{sign}((v_2^\dagger)_s) = 1$. Then $\theta_s - (v_2^\dagger)_s < 0$ because $|\,(v_2^\dagger)_s\,| > 1$ by definition. Thus $h_s > 0$, and $\text{sign}(h_s) = 1$. Now suppose $\text{sign}((v_2^\dagger)_s) = -1$. By similar logic to the previous case, $\theta_s - (v_2^\dagger)_s > 0$, and therefore $h_s < 0$ and $\text{sign}(h_s) = -1$, which is the desired conclusion. Therefore, setting $\theta_s = \text{sign}((v_2^\dagger)_s)$ will always yield $\theta_s = \text{sign}(h_s)$. Thus, the linear constraint in (3.1) is equivalent to the ℓ_1 constraint

in (2.3) for small enough displacements in the step direction $(\boldsymbol{h}_\tau, h_s)^T$. Therefore, primal feasibility is maintained throughout the algorithm [9].

4. Hyperparameter Selection

In addition to the variables determined within our model (e.g. $\boldsymbol{\beta}$, $\boldsymbol{h}$, etc.), we also have three hyperparameters: c, γ, and t.

As indicated in §2.5, the hyperparameter c is utilized in Huber's loss function. Essentially, c is used here as a cut off between the "small" errors and the "large" errors. According to Huber, values of c between 1 and 2 are good choices, and for our purposes we use the computational standard of $c = 1.345$ [4].

Recall that the γ hyperparameter is used in the creation of the RBF kernel. Ultimately, γ represents a value inversely proportional to the standard deviation of the RBF kernel (σ). More specifically,

$$\gamma = \frac{1}{2\sigma^2}.$$

Therefore, a small gamma defines a Guassian function for the RBF kernel with a large variance, and a large gamma defines such a function with a small variance. Also recall that the hyperparameter t is the postively-valued constraint we place on our β values.

While Huber [4] identifies an optimal value for c, γ and t must be optimized for each data set. We implement this optimization via a grid search over all possible pairs of candidate values. Preliminary candidate values of γ and t are generated by stepping an initial value over powers of ten: $\{10^{-3}, 10^{-2}, \ldots, 10^2, 10^3\}$. The optimal pair (γ_0, t_0) is defined to be that which minimizes the 10-fold cross validation error of the algorithm in §3. Next, we perform a fine grid search around this optimal pair where candidate values of γ are elements of the set $\{0.2\gamma_0, 0.4\gamma_0, \ldots, \gamma_0, 2\gamma_0, 4\gamma_0, 8\gamma_0\}$ and candidate values of t are elements of the set $\{0.2t_0, 0.4t_0, \ldots, t_0, 2t_0, 4t_0, 8t_0\}$.

5. Predictions

Once an optimal $\boldsymbol{\beta}$ has been chosen via the algorithm given some t and σ, predictions can be made on the robustly transformed test data, $\tilde{\boldsymbol{y}}_{\text{test}}$, as follows. Compute $\hat{\tilde{\boldsymbol{y}}}_{\text{test}} = \tilde{K}_{\text{predict}}\boldsymbol{\beta}_\tau$, where K_{predict} is the $n \times \tau$ centered prediction kernel matrix, which, similar to (3.17), is constructed element-wise such that

$$k_c(i,j) = k(i,j) - \frac{1}{\tau}\sum_{r=1}^{\tau} k(i,r) - \frac{1}{\tau}\sum_{r=1}^{\tau} k(j,r) + \frac{1}{\tau^2}\sum_{r=1}^{\tau}\sum_{s=1}^{\tau} k(r,s), \tag{5.1}$$

where $i \in \{1, \ldots, n\}$ indexes the rows of the test data, and $j \in \{1, \ldots, \tau\}$ indexes the rows of the train data. It is then necessary to multiply K_{predict} by $\Omega^{1/2}$ to get $\tilde{K}_{\text{predict}} = \Omega^{1/2}K_{\text{predict}}$.

To measure the accuracy of the predictions, calculate the RMSE, given by

$$RMSE_i = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\tilde{y}_i - \hat{\tilde{y}}_i\right)^2}, \tag{5.2}$$

and note that a smaller RMSE indicates a more accuarate prediction.

6. Rcpp

The computational cost of using cross validation along with obtaining the predictions is expensive. R language is notorious for being slow with computationally intensive programs. Rcpp is a useful application programming interface that addresses many of these obstacles and improves the efficiency of this algorithm. Rcpp was created by Dirk Eddelbuettel and Romain Francois, in order to use C++ language with R. This allows R to take advantage of the advanced data structures and functions included within the standard template library of C++.

Rcpp accounts for many of the bottlenecks experienced in R language. Programs that use recursion, have loops that are difficult to vectorize, or call functions millions of times, would benefit from using Rcpp. In this algorithm, Rcpp is implemented in the cross validation for the hyperparameter selection, and could be utilized anywhere there are loops. Results show that the performance time of Rcpp is more efficient than R alone [11].

7. Discussion

Moving forward in our implementation of the kernelized LASSO algorithm, we hope to perform experimental validation of the models accuracy, sparsity, and efficiency. This will include a comparison to SVM and ridge regression for benchmark data.

Currently, our primary performance measurements of prediction accuracy are the root mean squared error (RMSE) on the test data and the prediction variance given in (2.17). By computing the RMSE over a test dataset, we are validating our model's accuracy and checking for overfitting. Additionally, we will consider the number of coefficients in our model because sparsity is desirable for prediction efficiency and for avoiding overfitting.

In addition to evaluating the relative predictive validity of our model, we will assess its computational efficiency relative to Ridge and SVM regression. We anticipate needing to optimize our algorithm. While our algorithm is implemented in `R`, it includes many loops which we hope to make more efficient via the use of `Rcpp`, which permits integration of `R` and C++.

Once we have improved the efficiency of our model and properly compared the accuracy and sparsity of its results with those of other regression methods, we hope to provide our algorithm for public use.

References

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. Second. Springer Series in Statistics. Data mining, inference, and prediction. Springer, New York, 2009, pp. xxii+745. DOI: 10.1007/978-0-387-84858-7. URL: https://doi.org/10.1007/978-0-387-84858-7.

[2] Arthur E. Hoerl and Robert W. Kennard. "Ridge Regression: Biased Estimation for Nonorthogonal Problems". In: *Technometrics* 12.1 (1970), pp. 55–67. ISSN: 00401706. URL: http://www.jstor.org/stable/1267351.

[3] P. Honeine. "An eigenanalysis of data centering in machine learning". In: *ArXiv e-prints* (July 2014). arXiv: 1407.2904 [stat.ML].

[4] Peter J. Huber. *Robust statistics*. John Wiley Sons, Inc., New York, 1981, p. 18. DOI: 10.1007/978-0-387-84858-7. URL: https://doi.org/10.1007/978-0-387-84858-7.

[5] Keith Knight and Wenjiang Fu. "Asymptotics for lasso-type estimators". In: *Ann. Statist.* 28.5 (2000), pp. 1356–1378. ISSN: 0090-5364. DOI: 10.1214/aos/1015957397. URL: https://doi.org/10.1214/aos/1015957397.

[6] Hannes Leeb and Benedikt M. Pötscher. "Model selection and inference: facts and fiction". In: *Econometric Theory* 21.1 (2005), pp. 21–59. ISSN: 0266-4666. DOI: 10.1017/S0266466605050036. URL: https://doi.org/10.1017/S0266466605050036.

[7] Alan J. Miller. "Selection of subsets of regression variables". In: *J. Roy. Statist. Soc. Ser. A* 147.3 (1984), pp. 389–425. ISSN: 0035-9238. DOI: 10.2307/2981576. URL: https://doi.org/10.2307/2981576.

[8] Michael R. Osborne, Brett Presnell, and Berwin A. Turlach. "On the LASSO and Its Dual". In: *Journal of Computational and Graphical Statistics* 9 (1999), pp. 319–337.

[9] Volker Roth. "The Generalized LASSO". In: *IEEE Transactions on Neural Networks and Learning Systems* 15.1 (2004), pp. 16–28. ISSN: 1045-9227. DOI: 10.1109/TNN.2003.809398. URL: https://www.researchgate.net/profile/Volker_Roth/publication/8328258_The_generalized_LASSO/links/0c960516f971a38e3a000000/The-generalized-LASSO.pdf.

[10] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *J. Roy. Statist. Soc. Ser. B* 58.1 (1996), pp. 267–288. ISSN: 0035-9246. URL: http://links.jstor.org/sici?sici=0035-9246(1996)58:1<267:RSASVT>2.0.CO;2-G&origin=MSN.

[11] Hadley Wickham. "High Performance Functions with Rcpp". In: (). URL: http://adv-r.had.co.nz/Rcpp.html.