Optimization in Data Analysis

Project R3

Binary Classification using pathwise coordinate method

Daniel Brito, Marcel Affi

K-6034,316056

MSc Data Science

Faculty of Mathematics and Information Systems

Warsaw University of Technology

2021

Table of Contents

List of Figures

Introduction

The purpose of this paper is to explore "one-at-a- time" coordinate-wise descent algorithms for these problems. The equivalent of a coordinate descent algorithm has been proposed for the L1-penalized regression (lasso) in the literature, but it is not commonly used. Moreover, coordinate-wise algorithms seem too simple, and they are not often used in convex optimization, perhaps because they only work in specialized problems. We ourselves never appreciated the value of coordinate descent methods for convex statistical problems before working on this paper

The aim of the project is to build a linear regression model for:

$$y = X\beta + e \tag{1.1}$$

where:

- $y \in \mathbb{R}^n$ is a vector of observations (empirical data) of endogenous variables
- $X \in \mathbb{R}^{N \times p}$ matrix of observations of exogenous variables
- $\beta \in \mathbb{R}^p$ vector of models parameters
- $e \in \mathbb{R}^N$ is the model's error

Firstly, we can build a linear regression model by solving the following optimization problem:

$$\min_{\beta} \frac{1}{2}\|y - X\beta\|_2^2 \text{ , subject to } \|\beta\|_0 \leq k \tag{1.2}$$

Where:

- k: natural number $\leq$ p

- $\|\beta\|_0$ denotes the number of nonzero elements of the vector β

We can get an approximate solution of this equation by solving the problem:

$$\min_{\beta} \left[\frac{1}{2} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \right] \tag{1.3}$$

Where:

- λ is a penalty parameter

Our project will consist in solving (1.3) by using the **pathwise coordinate** method.

Derivation of Solution

Let us start the derivation of the solution by displaying the loss function of the LASSO problem. We could have chosen many loss functions, but since we will be looking to minimize this cost function by deriving it and equalling to 0, we want a cost function which is derivable so that is why we use the Squared Loss, also because it is conceptually very simple and it fits our problem characteristics.

$$MSE(\beta) = \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1 \tag{2.1}$$

Now let's split this expression into the two terms that form it and focus on them separately:

$$\|y - X\beta\|_2^2 = \frac{1}{2}\sum_{i=1}^{m}\left[y_i - \sum_{j=0}^{n}\beta_j X_{i,j}\right]^2 \tag{2.2}$$

$$\frac{\partial}{\partial\beta_j}\|y - X\beta\|_2^2 = -\sum_{i=1}^{m} X_{i,j}\left[y_i - \sum_{j=0}^{n}\beta_j X_{i,j}\right] \tag{2.3}$$

$$= -\sum_{i=1}^{m} X_{i,j}\left[y_i - \sum_{k\neq j}^{n}\beta_k X_{i,k} - \beta_j X_{i,j}\right] \tag{2.4}$$

$$= -\underbrace{\sum_{i=1}^{m} X_{i,j}\left[y_i - \sum_{k\neq j}^{n}\theta_k X_{i,k}\right]}_{\rho_j} + \theta_j \underbrace{\sum_{i=1}^{m}(X_{i,j})^2}_{z_j} \tag{2.5}$$

Where:

- z_j: normalizing constant, it is just the squared norm of the vector $X_{*,j}$

Now let's turn our attention n to the term with the absolute value, our first thought when

seeing this term is to recognize that it is not derivable for $\beta = 0$, so we will have to make use of some mathematical tools in order to simplify this expression.

$$\lambda\|\beta\|_1 = \lambda\sum_{j=1}^{m}|\beta_j| \tag{2.6}$$

$$\frac{\partial}{\partial\beta_j}\lambda\|\beta\|_1 = \lambda\frac{\partial}{\partial\beta_j}\left(|\beta_1| + \cdots + |\beta_j|\right) \tag{2.7}$$

$$= \lambda\frac{\partial}{\partial\beta_j}|\beta_j| \qquad \lambda\frac{\partial}{\partial\beta_j}|\beta_k| = 0, k \neq j \tag{2.8}$$

Here is where the notion of sub-differential comes into play, we will use the definition of sub-differential of a function $f(x)$ at a point x_0, which is the interval $[a, b]$ whose limits are:

$$a = \lim_{x\to x_0^-}\frac{f(x) - f(x_0)}{x - x_0}$$
$$b = \lim_{x\to x_0^+}\frac{f(x) - f(x_0)}{x - x_0}$$

Which applied to ou case when the function we are dealing with is $|\beta_j|$ we get that at the point $\beta_j = 0$, the sub-differential is the interval with ends:

$$a = \lim_{\beta_j\to 0^-}\frac{|\beta_j| - 0}{\beta_j - 0} = \lim_{\beta_j\to 0^-}\frac{|\beta_j|}{\beta_j} = -1$$
$$b = \lim_{\beta_j\to 0^+}\frac{|\beta_j| - 0}{\beta_j - 0} = \frac{|\beta_j|}{\beta_j} = +1$$

While, if $\beta_j > 0$ the function $|\beta_j|$ will be differentiable with derivative 1, so the sub-differential for $\beta_j > 0$ is 1. For $\beta_j < 0$ the function $|\beta_j|$ will be differentiable with derivative -1, so the sub-differential for $\beta_j > 0$ is -1. Combining these 3 results we get the sub-differential for $|\beta_j|$ in the whole of $\mathbb{R}$.

$$\frac{\partial}{\partial\beta_j}|\beta_j| = \begin{cases} -1 & \text{if } \beta_j < 0 \\ [-1, 1] & \text{if } \beta_j = 0 \\ 1 & \text{if } \beta_j > 0 \end{cases}$$

And returning to the differentiation of $\lambda\|\beta\|_1$, at (2.8), we get:

$$\lambda \frac{\partial}{\partial \beta_j}|\beta_j| = \begin{cases} -\lambda & \text{if } \beta_j < 0 \\ [-\lambda, \lambda] & \text{if } \beta_j = 0 \\ \lambda & \text{if } \beta_j > 0 \end{cases}$$

Now, we will join both expressions back in the derivative of the loss function (2.5) and with a little abuse of notation we can write:

$$\begin{aligned} \frac{\partial}{\partial \beta_j}\|y - X\beta\|_2^2 &= \rho_j + \beta_j z_j + \begin{cases} -\lambda & \text{if } \beta_j < 0 \\ [-\lambda, \lambda] & \text{if } \beta_j = 0 \\ \lambda & \text{if } \beta_j > 0 \end{cases} \\ &= \begin{cases} -\lambda + \rho_j + \beta_j z_j & \text{if } \beta_j < 0 \\ [-\lambda - (\rho_j + \beta_j z_j), \lambda + (\rho_j + \beta_j z_j)] & \text{if } \beta_j = 0 \\ \lambda + \rho_j + \beta_j z_j & \text{if } {}_j > 0 \end{cases} \end{aligned}$$

Now let's take a look at the interval in case $\beta_j = 0$, the interval can be simplified since the term $\beta_j z_j = 0$, therefore the resulting closed interval is $[-\lambda - \rho_j, \lambda + \rho_j]$

And equalling to 0

$$0 = \frac{\partial}{\partial \beta_j}\|y - X\beta\|_2^2 = \begin{cases} -\lambda + \rho_j + \beta_j z_j & \text{if } \beta_j < 0 \\ [-\lambda - \rho_j, \lambda + \rho_j] & \text{if } \beta_j = 0 \\ \lambda + \rho_j + \beta_j z_j & \text{if } \beta_j > 0 \end{cases} \tag{2.9}$$

Now we want 0 to be inside the closed interval, so that $\beta_j = 0$ is a global minimum.

$$0 \in [-\lambda - \rho_j, \lambda + \rho_j] \iff \rho_j \in [-\lambda, \lambda]$$

So if we solve (2.9) for β_j we get:

$$\begin{cases} \beta_j = \frac{\rho_j + \lambda}{z_j} & \text{if } \rho_j < -\lambda \\ \beta_j = 0 & \text{if } \rho_j \in [-\lambda, \lambda] \\ \beta_j = \frac{\rho_j - \lambda}{z_j} & \text{if } \rho_j > \lambda \end{cases}$$

Since z_j just represents the normalization variable we can rewrite the update rule for the case in which the data we are working with is already normalized, or like we do in the implementation, the data is firstly normalized and then worked on. This results in

$$\begin{cases} \beta_j = \rho_j + \lambda & \text{if } \rho_j < -\lambda \\ \beta_j = 0 & \text{if } \rho_j \in [-\lambda, \lambda] \\ \beta_j = \rho_j - \lambda & \text{if } \rho_j > \lambda \end{cases}$$

Implementation

We can divide our implementation into 2 different approaches. The first will be based on the successive updates to the parameters (β) using the soft thresholding function.

```
def coordinate_descent_lasso(theta,X,y,lamda = .01, num_iters=100, intercept = False):
    '''Coordinate gradient descent for lasso regression - for normalized data.
    The intercept parameter allows to specify whether or not we regularize theta_0'''

    #Initialisation of useful values
    _,n_coordinates = X.shape
    X = X / (np.linalg.norm(X,axis = 0)) #normalizing X in case it was not done before

    #Looping until max number of iterations
    for i in range(num_iters):

        #Looping through each coordinate
        for j in range(n_coordinates):

            #Vectorized implementation
            X_j = X[:,j].reshape(-1,1)
            y_pred = X @ theta
            #We compute rho_j to later update the coordinate parameter theta_j
            rho = X_j.T @ (y - y_pred  + theta[j]*X_j)

            #Checking intercept parameter
            if intercept == True:
                if j == 0:
                    theta[j] =  rho
                else:
                    #theta_j = S(rho_j, lambda)
                    theta[j] =  soft_threshold(rho, lamda)

            if intercept == False:
                #theta_j = S(rho_j, lambda)
                theta[j] =  soft_threshold(rho, lamda)

    return theta.flatten()
```

Figure 3.1: coordinate descent LASSO

So in this function we declare 2 loops, in the first one we will iterate until we reach the maximum number of iterations, while in the second is where the Coordinate Descent comes into play, we will select each coordinate and ultimately update the value of theta (vector containing all of the parameters) using the above mentioned soft threshold function.

For next section we will compare 2 different approaches to solving the LASSO problem, firstly using a Sklearn approach, and secondly using Coordinate descent, using Cycling or randomized parameter choice.

Sklearn Approach

We will create a class that holds a data attribute where we will load in the data to try these approaches, the data set we use is hosted on d contains data about baseball hitters, such as the number of hits, home runs, runs, years, etc.

For the Sklearn approach we will very briefly explain each function since it is not the concern of our current project. We have *generate_simulate_data(self)* which as its name implies generates data using a Gaussian (normal) distribution to which it adds another small Gaussian distribution acting as random error. then it standardized this data and splits it into random train and test sets to train our data model, the train set will be 75% of the data set while the test set 25%. Function *process_data(self)* loads the data for the hitters, splits it into test and train sets and also transforms (normalizes) those sets. The final function is where the learning happens. *sklearn_training(self, x, y)* uses lasso cross validation to obtain the optimal λ as well as coefficient to minimize the lasso optimization objective function which is:

$$\frac{1}{2 * n_samples} * ||y - Xw||_2^2 + alpha * ||w||_1$$

Coordinate Descent Approach

In this section we won't dive deep into how we extract the update rules for coordinate descent since we have already explained it previously. The implementation doesn't differ much from that one, with only new addition is how we pick the coordinate we are minimizing for each time, since we introduce a random choice, which is not purely random since, to avoid it getting stuck in 1 coordinate over and over we force to change so the current coordinate we are minimizing for is not the same as the last coordinate we were using.

We will however create a function to visually compare the convergence of these 2 Coordinate approaches we mentioned. This function

objective_plot(self, betas_cyclic, betas_rand, x, y, shrink_axis = False)

Will receive a list of betas for cyclic and randomized coordinate descent, for which it will firstly calculate the objective function

Conclusion

Firstly let's observe the plot comparing cyclic coordinate descent and the randomized approach.

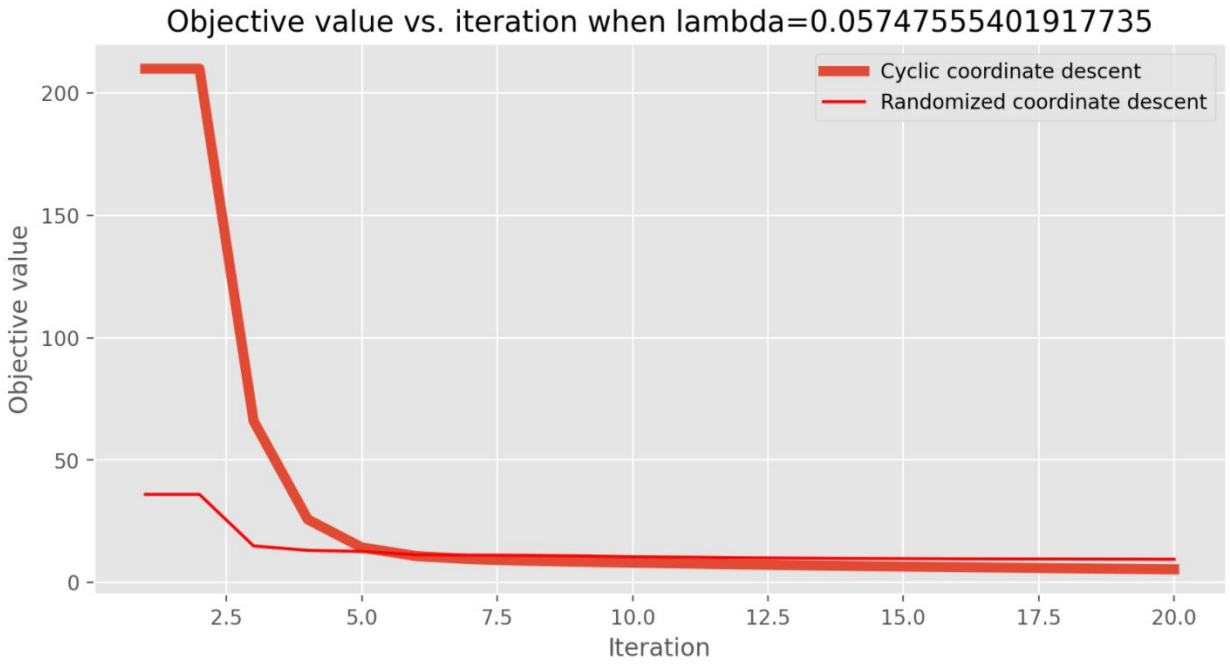

Figure 4.1: Cyclic coordinate descent vs randomized coordinate descent

As expected the results do not vary a great deal, especially once we have a few iterations, but the results at the very beginning suggests that the randomized approach could have been "lucky" picking a coordinate which affected the objective value more than the cyclic coordinate one which automatically picked the first coordinate.

Additionally we will also compare the amount of coefficients which after applying the 3 approaches went to 0, and those of which didn't, comparing the coordinate approached with the sklearn one.

Which yields the following results:

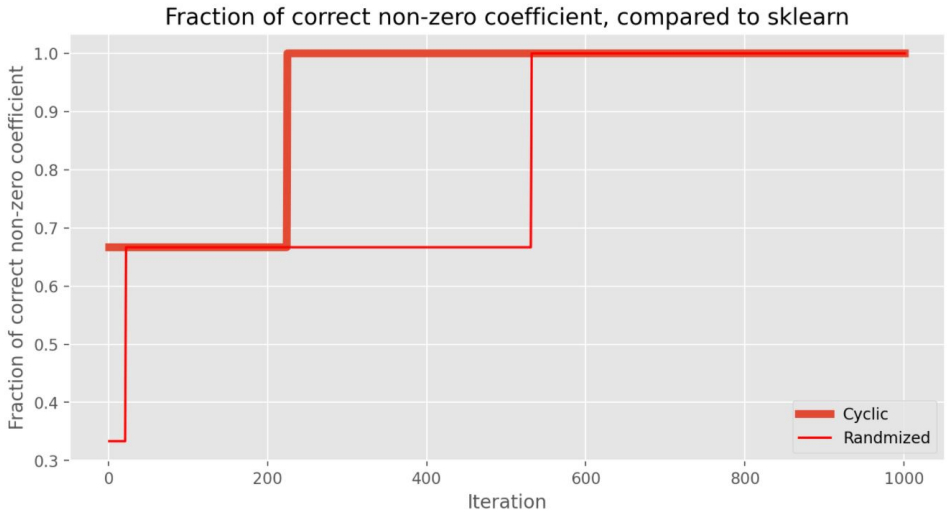

Figure 4.2: non zero coefficients compared to sklearn approach

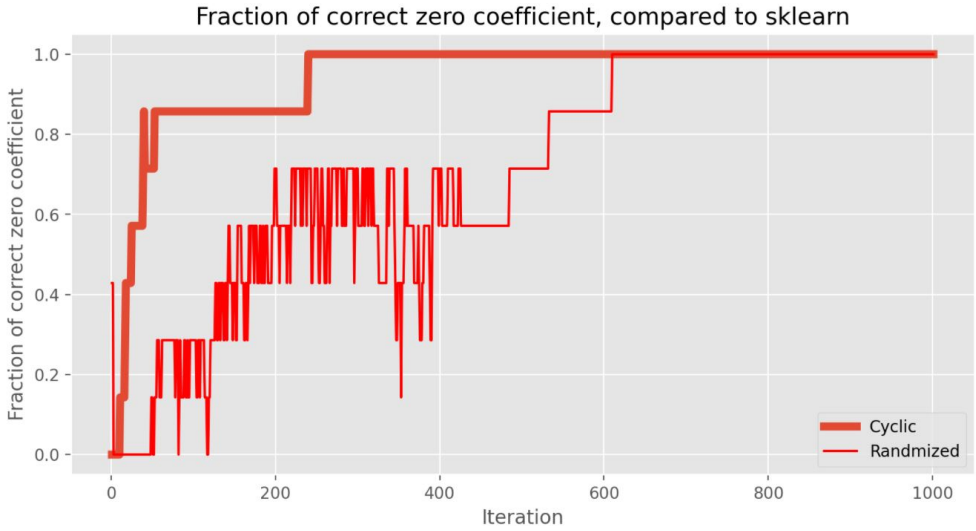

Figure 4.3: zero coefficients compared to sklearn approach

´

These plots give us further evidence that our coordinate approach is converging to the right results after a number of iterations.

Although these graphs suggest that the randomness of the randomized approach actually drives us away from our objective since we can see countless spikes going down, while the cyclic one converges much more "smoothly" to the sklearn solution.

Other Applications of Coordinate Descent

Another use of coordinate descent which is easy to visualize using 2 or 3-dimensional plots, is the method of coordinate descent to minimize a smooth function $f(x_1, x_2)$ (or in the case of 3-dimensional plot, a function which takes 3 variables $f(x_1, x_2, x_3)$)

In this case the algorithm will, in a very resumed manner, and after choosing (randomly or through user input) an initial vector (x_{10}, x_{20}) (or $(x_1^{(0)}, x_2^{(0)}, x_3^{(0)})$) we will:

- Choose the variable which we will be minimizing on (x_i)
- Choose a step size α through a user-preferred method (for example, line search like in gradient descent)
- update x_i of the vector with $x_i - \alpha \frac{\partial f}{\partial x_i}(x_1, \cdots, x_n)$
- Repeat this procedure until we have achieved convergence, or set a maximum number of iterations.

The result will be

We can clearly see the "steps" the algorithm takes until it reaches the minimum, each step signifies that the algorithm changed which variable it was minimizing for.

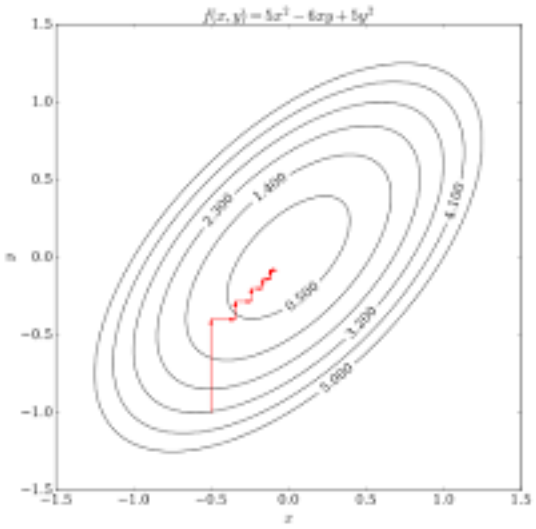

Figure 5.1: Coordinate descent of function $f(x, y) = 5x^2 - 6xy + 5y^2$