THE UNIVERSITY OF TEXAS AT AUSTIN

McCOMBS SCHOOL OF BUSINESS

RM 294 - Optimization I

Project 3 - Non-Linear Programming

Group 3

Mihir Deshpande (MD46487)

Pratik Gawli (PBG397)

Soumya Agrawal (SA55638)

Tanushree Devi Balaji (TB33857)

1. Introduction

Big data is integral to many business decisions being made on a daily basis. While its abundance helps generate confidence in the approach, the downside is redundancy. Most times, non-essential features do more bad than good, making the model more complex without adding to its predictive abilities. It is, hence, useful to resort to utilizing only a subset of highly important features that elevate model accuracy and interpretability. Without handpicking useful attributes, models could suffer from the curse of dimensionality, latency or loss of predictive power. This is why feature selection or extraction is sought after and one of its popular types is Lasso regularization. Lasso is an acronym for "Least Absolute Shrinkage and Selection Operator", which modulates complexity by imposing an L1 penalty on the coefficients in the loss function.

While there are many benefits to using Lasso, it is notorious for its instability. Given the reliability of direct variable selection through cross-validation and recent developments in mixed integer quadratic optimization techniques, wielding this method could potentially improve model reproducibility and add value in the long run. However, direct variable selection does come with its fair share of computational challenges. The objective of this project is to weigh the pros and cons of these two methods against each other and pick the one that best suits our needs.

Approach:

To compare the two techniques, we will be running a regression on the dataset given, which has been split into training and test datasets. Following are the steps to be followed:

1. Develop an optimization model to be run in Gurobi. The objective of the model would be to find the right parameters or "betas" that lower the loss of regression with 'k' predictors. The ideal k would be "directly" picked through trial and error (i.e) cross-validation and subsequently used to find the test mean squared error.
2. Run Lasso regression using the training set. The objective is to find the best value of 'lambda' through cross-validation that will "indirectly" reduce the dimensionality and also provide the lowest mean squared error. Validate the model on the test set by calculating its mean squared error.
3. Compare & contrast both methods to come up with insights to help make a decision about switching from Lasso to Direct variable selection

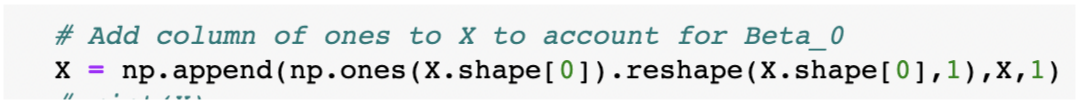
2. Direct Variable Selection – MIQP Problem

Data Setup:

We start off by setting up the data for MIQP. What we will essentially do is solve for the decision variables (i.e) ‘m’ betas through matrix multiplication involving X, the matrix with the predictors and y, the matrix with the prediction variable, both of which are ‘n’ rows long with one for each data point.

<u>To set up the data:</u>

Add a column of ones to the X matrix, since we will be including an intercept in our regression. Thus, X is an n * (m+1) matrix, with one column corresponding to each beta value



```
# Add column of ones to X to account for Beta_0
X = np.append(np.ones(X.shape[0]).reshape(X.shape[0],1),X,1)
```

Once we have this, the matrix form would be X * β = y

The data is already split into training and test data sets. We will be using the training set for estimation and the test set for validation. In our case, we have 50 coefficients and 1 intercept.

Modeling Approach:

Objective:

We are trying to minimize the squared loss incurred during regression, which is given by:
This function above can be represented as

$$\min_{\beta,z} \sum_{i=1}^{n} (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_m x_{im} - y_i)^2$$

$$(X\beta - y)^{\mathsf{T}} * (X\beta - y)$$

And subsequently as,

$$\beta^{\mathsf{T}}(X^{\mathsf{T}} X)\,\beta + (-2\,y^{\mathsf{T}}X)\,\beta$$

We also need to take into account 50 ‘z’ variables, one for each regression coefficient, that will select our desired variables. To accommodate these changes, we redefine our objective by introducing two new matrices that will give us our final quadratic objective:

Quadratic Matrix:

Matrix Q_obj of size 2m+1 * 2m+1 = $X^T X$ is the upper left corner and the rest zeros

```
# Get quadratic term of objective
Q = X.T @ X

# We need Quad objective to be of dimensions (2m+1)x(2m+1)
Q_obj = np.zeros((2*m+1,2*m+1))
Q_obj[:m+1,:m+1] = Q
```

Linear Term:

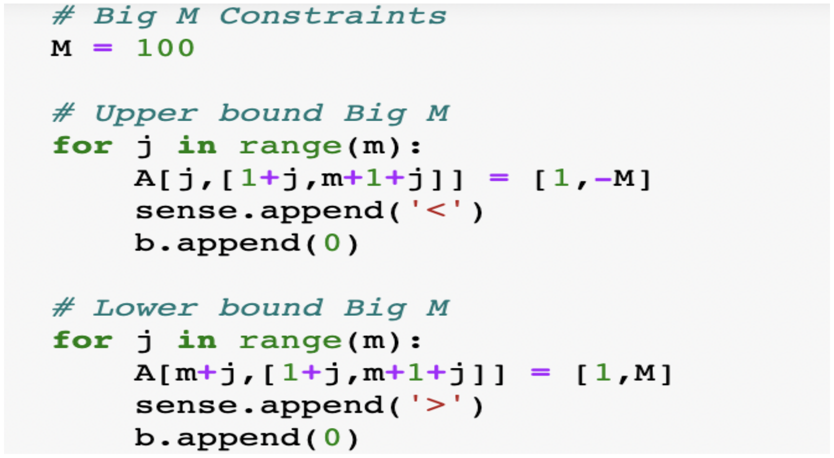
Matrix L_obj of size 2m+1 where the first m components are $-2\ y^T X$ and the rest zeros

```
# Get Linear term of objective
L = y.T @ X
#print(L)
L = -2*L
#print(len(L))

# We need linear objective to be of length 2m+1
L_obj = np.zeros(2*m+1)
L_obj[:m+1] = L
```

Constraints:

1. The z variables tell us whether a beta gets picked or not. Thus, a beta can exist only if z is non-zero. This can be formulated with the help of big M constraints as follows:
   where M is a very large value. In our case, we have chosen M to be 100.

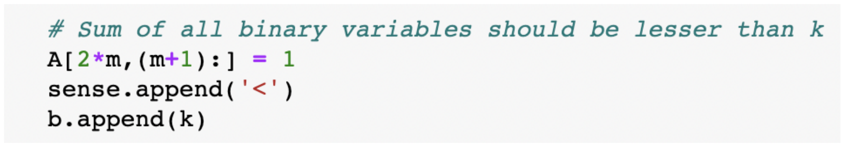
$$s.t. -Mz_j \leq \beta_j \leq Mz_j \quad for\ j = 1, 2, 3, \ldots, m$$



```
# Big M Constraints
M = 100

# Upper bound Big M
for j in range(m):
    A[j,[1+j,m+1+j]] = [1,-M]
    sense.append('<')
    b.append(0)

# Lower bound Big M
for j in range(m):
    A[m+j,[1+j,m+1+j]] = [1,M]
    sense.append('>')
    b.append(0)
```

2. The number of variables to be chosen is given by k and hence,

$$\sum_{j=1}^{m} z_j \leq k$$



```
# Sum of all binary variables should be lesser than k
A[2*m,(m+1):] = 1
sense.append('<')
b.append(k)
```

3. A feature can either be present in the model or not and hence, the z variables are binary.

With all of this in mind, we have **101 decision variables** and **101 constraints**

Modeling:

To run the model, we will perform 10-fold cross-validation to test different values of k.

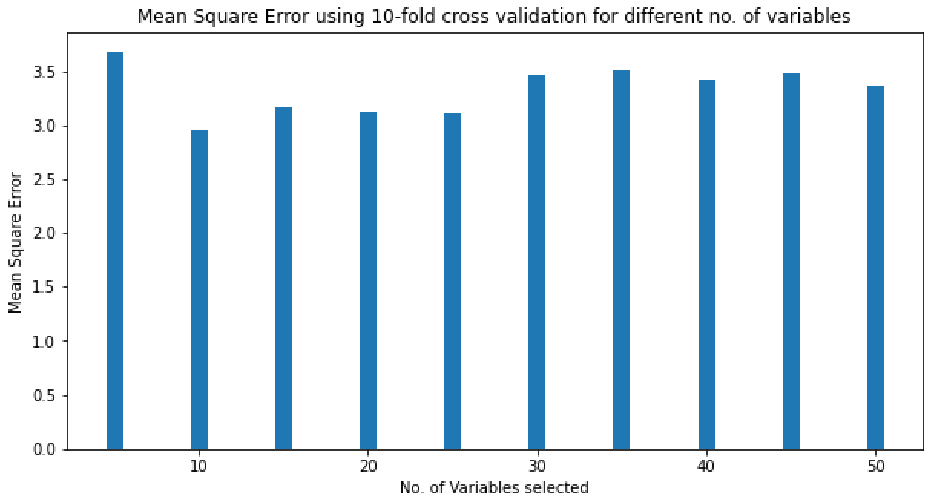
1. Pick a value of k
2. Use the random_choice function to split the indices into 10 equal parts
3. Choose 9 of these parts to be your training set and the one left out to be your holdout or validation set. This way, you can create 10 batches of train-holdout set combinations.
4. On each of these 10 batches, find betas for k variables and the corresponding mean squared error on the holdout set. Now, retain the average of all these errors.
5. Repeat this process for k = 10, 15, … 50.
6. Pick the k which gives the lowest error of all.
7. Use that k to run the model on the full training set and find test errors.

```python
# Randomly shuffle data
ind = np.random.choice(list(X_train.index),len(X_train),replace=False)
ind_shuffled = np.array_split(ind,k_fold)

# Initialize arrays to store values of output
mse_kfold = []
betas_kfold=[]
val_kfold=[]

# Iterate through all k-folds
for ind_kfold in ind_shuffled:

    X_kfold_val = X_train.iloc[ind_kfold,:]
    y_kfold_val = y_train.iloc[ind_kfold]
    X_kfold_train = X_train.drop(ind_kfold,axis=0)
    y_kfold_train = y_train.drop(ind_kfold,axis=0)
```

Results:

Below are the Mean squared errors for all values of k for which the model was run. **k = 10** gives the best results for cross-validation with an average MSE of 2.96. This implies that using fewer variables than all the 50 variables actually yields better results while predicting 'y' variable as the average MSE by using 50 variables is 3.68, much higher than 2.95 on the validation set.

MSE	No. of Variables
2.958498	10
3.103878	25
3.124743	20
3.173731	15
3.371867	50
3.418577	40
3.468592	30
3.483805	45
3.505869	35
3.679512	5



After we trained the model with the ideal number of variables(10) on the **entire** training data and estimate the results for the test data, we get an MSE of **2.336.** Following are the corresponding beta values for these 10 variables and an intercept. As none of the Beta values is equal to 'M', we do not need to re-run the model with higher values of M.

	Beta Values
0	0.972524
9	-2.308207
15	-0.518326
16	-0.204162
23	-1.559143
24	0.866973
26	-1.311919
34	0.408165
45	1.781475
47	0.887383
48	-0.282292

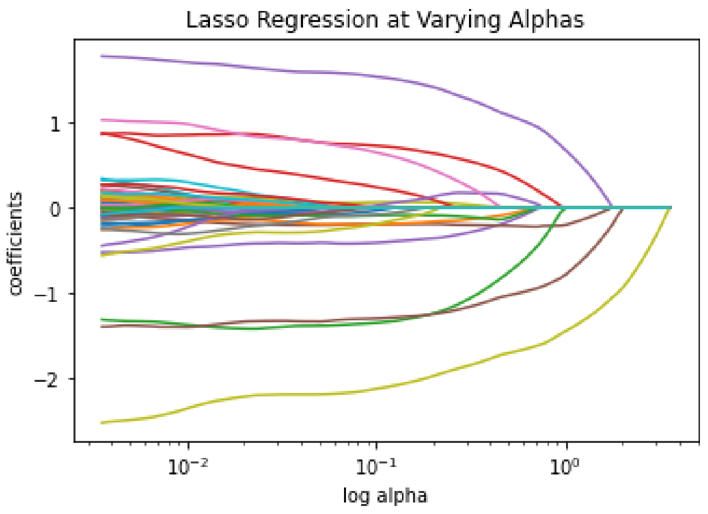
As we can see out of all the 50 variables, only X9, X15, X16, X23, X24, X26, X34, X45, X47 and X48 are significant enough to give a minimum MSE of 2.336. Meaning a better performance was obtained using just 20% of the original variables.

3. Indirect Variable Selection – LASSO

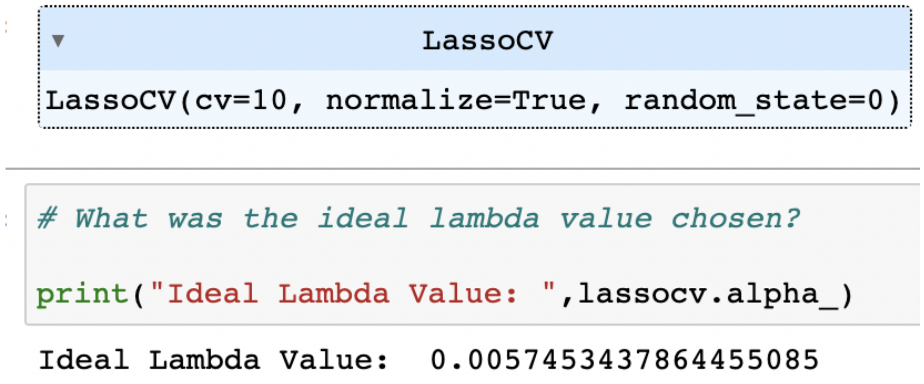
Modeling Approach:

1. To perform feature selection we used cross-validation(LassoCV) to find the best value of **lambda,** the penalty parameter. This can be achieved quite easily through Scikit learn.
2. Set 'normalize=True' to normalize the data to get appropriate results using Lasso
3. Once lambda is obtained, fit the model to the training set to find betas.
4. Use this to predict Test MSE

Results:



The above graph shows the variation of the 50 parameters as a function of the penalty parameter, lambda. One interesting thing to observe from the above graph is that majority of the coefficients are already close to zero, even at no regularization. We can see that 7 variables are having higher coefficient values which would be instrumental in determining the dependent variable y.
This demonstrates the motivation to reduce the number of variables in determining the output as it might be helpful in reducing information acquisition cost and still give comparatively better results.



```
LassoCV
LassoCV(cv=10, normalize=True, random_state=0)
```

```
# What was the ideal lambda value chosen?

print("Ideal Lambda Value: ",lassocv.alpha_)
```

```
Ideal Lambda Value:  0.005745343786445085
```

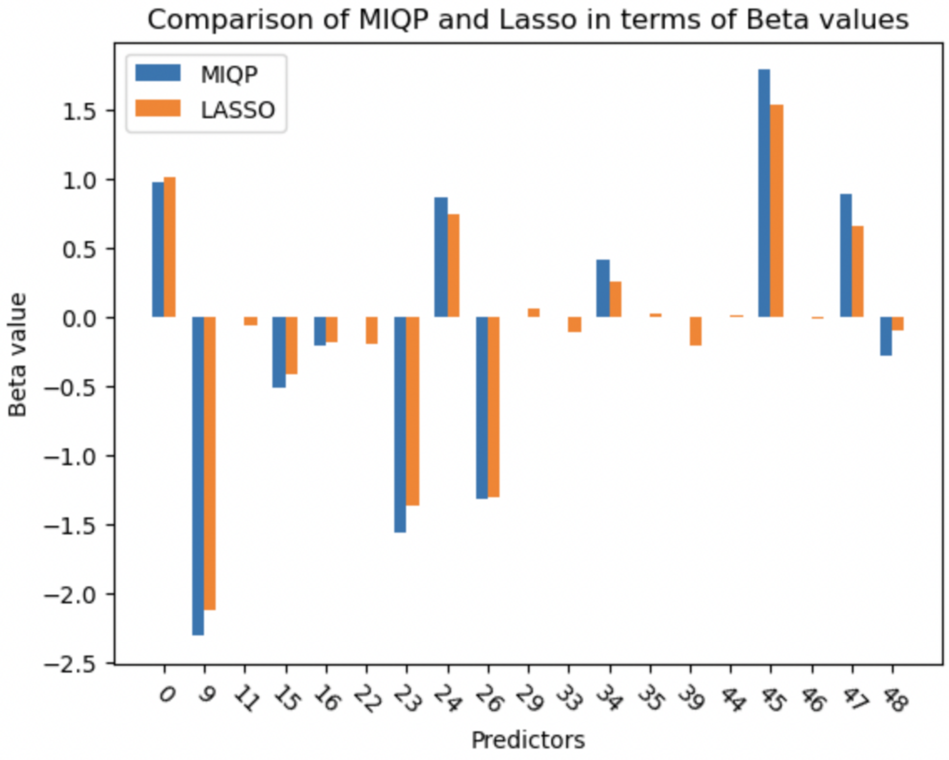
The ideal penalty parameter(Lambda) determined by running the LassoCV model from scikit learn is **0.0057.**

This value of lambda gives us a **test error** of about **2.35**, similar but slightly on the higher side of the direct selection model however, Lasso retains **18 variables** as opposed to the 10 chosen by direct selection. Lasso method chose X11, X22, X29, X33, X35, X39, X44, and X46 in addition to the 10 variables chosen by the direct method.

Note that this list does not include the intercept as it is not penalized by the value of lambda.

4. Direct Selection vs Lasso:

Comparing the beta values determined from these two methods, we get the following graph:



Visually, we could see that beta coefficients which are higher in magnitude are picked by both, the direct and indirect methods. The extra variables which are picked by Lasso are very small in magnitude and do not affect the dependent variable to a large extent.
This is in line with our observation from the Lasso regressions' graph with varying values of lambda. The ones which are important are picked by both methods, but Lasso picks some additional variables whose values are closer to zero, meaning they are not that influential in determining the output variable.

Advantages:

Direct Selection	**Lasso**
• Gives minimum MSE with lesser number of variables • Gives a slightly lower MSE than LASSO • Lower cost of acquisition of variables possible	• Quicker than the Direct Selection method. • Only one parameter(lambda) to be determined • Fast and highly Interpretable • Scikit learn packages available in python to implement Lasso

Disadvantages:

Direct Selection	**Lasso**
● Computationally heavy ● To achieve better results, a more granular value search for k is needed ● Less interpretable ● No package available in programming languages	● Higher number of variables compared to direct selection for minimum MSE. ● Performance drops slightly on the test dataset compared to the direct method ● Even though it selects more variables, MSE does not improve

5. Conclusion:

Both the techniques applied on a given dataset aim to reduce the predictor variables which are lesser influential on the output variable. However, both techniques come with their own advantages and disadvantages discussed in the previous section and must be used according to the situation in which they are to be applied.

If we have not collected all the variables for the entire population for which we want to conduct our analysis, and the cost of acquisition of these variables is high, It would be advisable to conduct the above analysis on a sample of our population and determine the minimum number of variables required which give a lower MSE. Subsequently, for all the other data points, acquire only these variables which were selected using the direct method, thus, lowering the acquisition cost of the variables.

If we have already acquired all the variables for the entire population for which we want to conduct our analysis, then it would be advisable to use the Lasso method as it gives fast and interpretable results with more or less the same error. Also, as the computation time of the direct method is comparatively higher, it might become much more time-consuming when the number of variables to select the best k variables is too high. However, we have seen that it does give a slightly better performance than the indirect method.