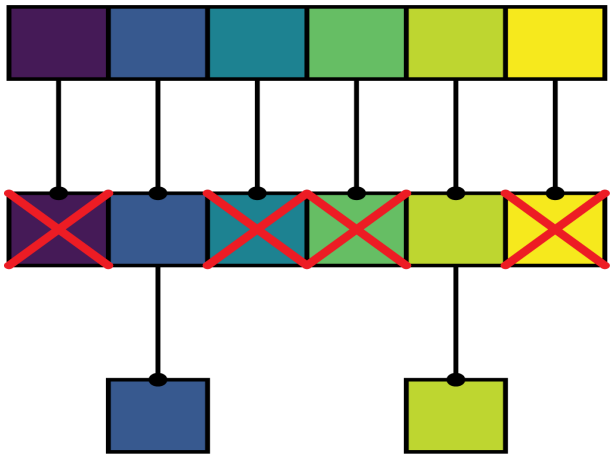
Optimization Project 3 report

Variable selection with Lasso vs Mixed Integer Quadratic Program (MIQP)



Group Members:
Anthony Moreno (am83596)
Audrey Hsien (arh4247)
Fernando Chapa (fac795)
Muskan Agarwal (ma64547)

Introduction

Predictive analytics refers to the use of historical data to discover patterns and predict future trends to drive strategic business decisions. This type of advanced analytics is often combined with statistical modeling, data mining techniques and machine learning.

One of the most common problems in predictive analytics is variable selection for regression. Due to computational difficulties, direct variable selection using optimization has long been dismissed by the statistics/analytics community. In fact, this computational issue was part of the motivation for the development of LASSO and ridge regression.

Recently, there have been tremendous advancements in optimization software, specifically the ability to solve mixed integer quadratic programs (MIQP). By comparing the results of the MIQP in Gurobi and LASSO in Scikit Learn, our project looks to observe if the additional 'shrinkage' component of LASSO is more beneficial than finding the 'best' set of variables to include in your regression.

In this report, we will pose the variable selection problem for regression as an MIQP, and compare it with LASSO. We will discuss the advantages and drawbacks of both techniques and make recommendations on which method is better with our given dataset.

Approach 1

Direct Variable Selection – MIQP Problem

Mixed integer quadratic programs (MIQP) is the problem of optimizing a quadratic function that allows for both discrete and continuous variables. It is an optimization method that minimizes the regular ordinary least squares loss function while taking into consideration constraints, such as the Big M method, which involves using binary variables (like z_j) in order to include or exclude certain variables in the calculations during loss calculation. The other important constraint is the hyperparameter k, which is the maximum number of variables to be included in the function.

The standard ordinary least squares problem can be mathematically formulated as: $min\ (\beta) \sum_{i=1}^{n} (\beta_0 + \beta_1 x_{i1} + \ldots + \beta_m x_{im} - y_i)^2$. In order to incorporate variable selection into this problem, our objective function to minimize the sum of squared errors (training error) can be mathematically formulated as: $(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_m x_{im} - y_i)^2$.

After applying linear algebra, we can rewrite the optimization problem's final objective function as below.

Objective:

$$min\ (\beta, z)\ \beta^T (X^T X)\beta + (-2y^T X)\beta$$

Where,
m = number of independent variables x
n = number of datapoints

To achieve this, the decision variables and constraints used are:

Parameters:

- k: The max number of variables that we want to include from X and a hyperparameter to be chosen using cross validation.
    - There are 50 X variables, and we chose k = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50].
- Q Matrix: Quadratic component of the objective function.
    - A (2m+1) * (2m+1) matrix where the upper left corner of the matrix is equal to $X^T X$, and all other values are zero.

- L Matrix: Linear term of the objective function.
    - Here it is a (2m+1) * 1 vector where the first (m+1) components are -$2y^T X$, and the rest are zeros.

Decision Variables:

- β_j are the beta coefficients and are continuous variables.
- z_j are the binary variables.

Constraints:

Count - 2m + 1

1. **The k constraint**: Sum of z_j needs to be equal to or less the number of variables (no_of_variables) to be selected as the number of non-zero features should not exceed k.

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

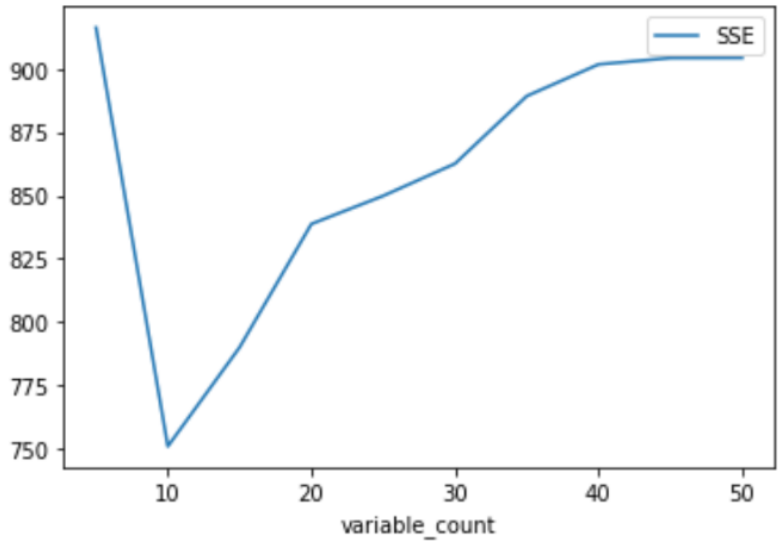
Mutual exclusivity: z_j is binary. This forces the corresponding values of β_j to be zero if z_j is zero. Otherwise, the value of β_j should be between -M and M.

2. **Big M (Upper Limit & Lower Limit)**: M needs to be large enough so that no value of β_j is equal to -M or M. Therefore, M is the constraint for the upper limit and lower limit.

$$-Mz_j \leq \beta_j \leq Mz_j \; for \; j = 1, 2, 3, \ldots, m$$

Step 1: Using 10 fold cross validation find the best k in terms of SSE. We have created a function *get_optimal_betas* that first randomizes the data for 10-fold cross validation ***(Appendix: Figure 1)*** and gives the validation SSE for each k.

The results of MIQP show that k= 10 is the best as it is giving the lowest validation error (SSE).



Step 2: When used k = 10 to find the betas using full training data in MIQP and predicting on test data, the results were as below:

variable_count	MSE
10.0	2.336544

Approach 2

Indirect Variable Selection – LASSO

LASSO regression is a type of linear regression that regularizes model parameters by shrinking the regression coefficients, reducing some to zero, and in turn, minimizing the loss function. The feature selection phase occurs after the shrinkage, where every non-zero value is selected to be used in the model.

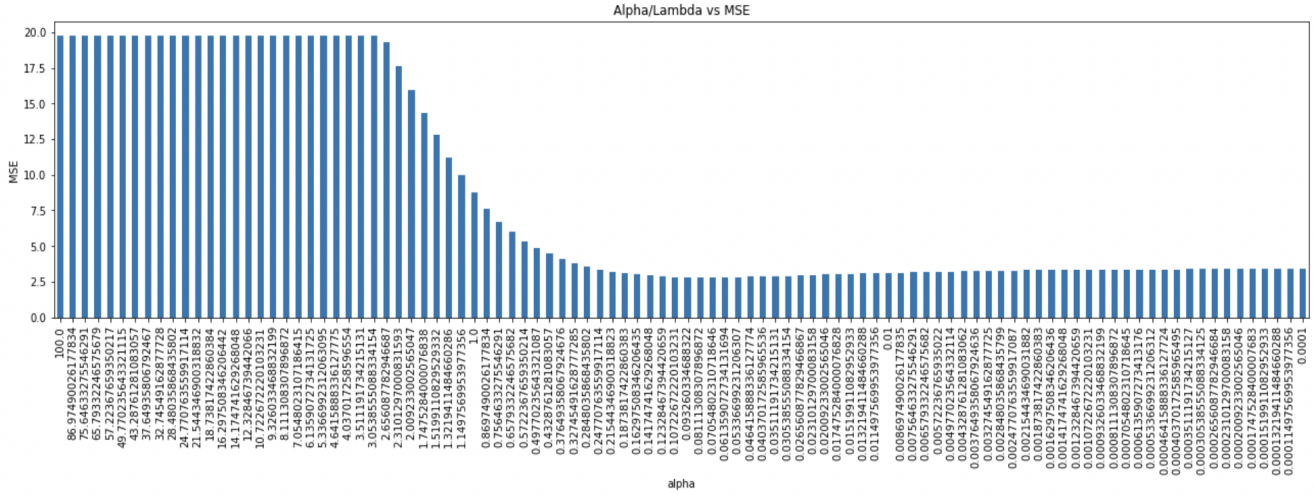
LASSO uses the hyperparameter lambda/alpha (λ) to perform this shrinkage, where the larger λ grows, the more X variables are forced to absolute zero.

Mathematically, we can represent the objective function as:

$$min\ (\beta) \sum_{i=1}^{n} (\beta_0 + \beta_1 x_{i1} + \ldots + \beta_m x_{im} - y_i)^2 \ + \ \lambda \sum_{j=1}^{m} |\beta_j|.$$

We used Scikit Learn to solve the LASSO problem.

Step 1: Using cross validation to find the best lambda/alpha in the range (100, 0.0001). The best alpha came out to be 0.081 ***(Appendix: Figure 2)***



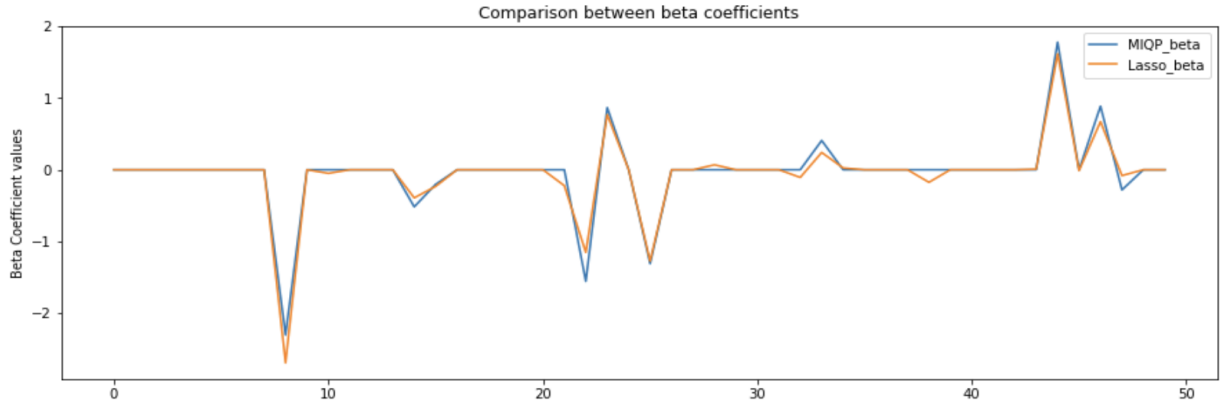
Step 2: Use the best alpha to fit the Linear Regression with Lasso regularization on the training dataset. The MSE on test data was 2.36 ***(Appendix: Figure 3)***

```
Test MSE of lasso regression is:  2.3610589089053495
```

Comparison

Beta Coefficients:

1. The MIQP selected 10 variables which were, x9, x15, x16, x23, x24, x26, x34, x45, x47, x48.
2. The LASSO also gave high beta coefficients for these variables along with some more non-zero coefficients.



Mean Squared Error (MSE) & Runtime :

Method	Tuning Parameter	MSE	Runtime
MIQP	k = 10	2.336	2 Hrs
LASSO	lambda = 0.081	2.361	2 mins

- MIQP was the better performing method yielding a lower test MSE of 2.337, while LASSO's test MSE of 2.361
- The computational time of MIQP was significantly greater at two hours, compared to some minutes for LASSO.

Conclusion

Similar to the majority of data science and predictive analytics problems, our conclusion is that neither the MIQP nor LASSO algorithm completely outperform the other in all categories.

Some advantages of the LASSO algorithm are:

1. The feature selection process is smoother and helps avoid overfitting as it allows the coefficients to gradually reduce to zero instead of completely eliminating them at once.
2. It can be implemented using built-in python libraries making it easy to use.

In contrast, some disadvantages of LASSO are:

1. If the data has a group of highly correlated features, it may arbitrarily select only one of the features from that group.

For MIQP, some of the advantages are:

1. It yields results with less number of features making it less complex and, as shown in our results, MIQP outperforms LASSO marginally due to the additional bias that LASSO's shrinkage adds.
2. It helps to find the most important features/variables for the model and is more explainable to business

But the main disadvantage of MIQP is:

1. The expense of additional computing power needed and a much greater computing time.
2. If the number of variables are high then the computing time will be very high.

Therefore, if sufficient computing power and time is available, MIQP is the better algorithm. But if quick, yet still very adequate solutions are preferred, LASSO is still a reasonable choice.

Recommendations:

- In terms of accuracy, MIQP has slightly better results than LASSO. So if the time and commuting power is not an issue then MIQP method should be selected.
- When efficiency is more important then LASSO regularization method should be selected. It is easier to implement and doesn't need understanding of optimisers.

Appendix

```
def get_optimal_betas(variables_to_select):

    index = np.random.choice(size_of_train_data, size_of_train_data, replace = False )

    # Initialize Dataframe for variable count and MSE
    df_MSE = pd.DataFrame(columns=['variable_count','MSE', 'SSE'])

    # For Loop for 10 rounds of Min MSE with different validation set for one variable count
    for count in range(no_of_valid_set):
        i = count*size_of_valid_data
        start = i
        end = i+size_of_valid_data
        valid_index = index[start:end]

        # Creating the Validation Dataset
        X_valid = train.iloc[valid_index]

        # Creating the Training Dataset
        train_index = np.concatenate((index[:start], index[end:]))
        X_train = train.iloc[train_index]
```

Figure 1: Setup of Cross-validation for MIQP problem

```
# Libraries for K-Fold cross validation
from sklearn.linear_model import Lasso
from sklearn.model_selection import cross_validate

# Creating 2 dataframes for x and y
# Train
X_train=train.drop('y',axis=1)
Y_train = train['y']

# Test
X_test=test.drop('y',axis=1)
Y_test = test['y']

# Scaling the x data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Cross validation and neg MSE

# Setting the range for alpha values to (100,0.0001)
alpha = 10**np.linspace(2,-4,100)

lasso_neg_mse_vals = []

for a in alpha:

    lasso_cv_n_mse = cross_validate(Lasso(alpha = a), X_train, Y_train, scoring='neg_mean_squared_error', cv=10,
        return_train_score = True)

    lasso_neg_mse_vals.append(lasso_cv_n_mse['test_score'].mean())

list_of_tuples_lasso = list(zip(alpha, lasso_neg_mse_vals ))

# Converting lists of tuples into pandas Dataframe
lasso_nmse_df = pd.DataFrame(list_of_tuples_lasso,
                  columns=['alpha', 'lasso_neg_mse_vals'])
```

In [120]:
```
lasso_nmse_df.set_index(['alpha'], inplace = True)
best_alpha_lasso = lasso_nmse_df[['lasso_neg_mse_vals']].idxmax()
print(best_alpha_lasso)
```

```
lasso_neg_mse_vals    0.081113
dtype: float64
```

Figure 2: Finding the best alpha for LASSO

```
# Linear Regression with Lasso regularization results

lasso_reg = Lasso(alpha=best_alpha_lasso[0])

lasso_reg.fit(X_train, Y_train)

Y_pred_train = lasso_reg.predict(X_train)
Y_pred_test = lasso_reg.predict(X_test)

from sklearn.metrics import mean_squared_error
lasso_MSE_train = mean_squared_error(Y_train, Y_pred_train)
lasso_MSE_test = mean_squared_error(Y_test, Y_pred_test)

print("Test MSE of lasso regression is: ", lasso_MSE_test)

coeff = []

coeff_alpha = list(lasso_reg.coef_)
coeff.append(coeff_alpha)

lasso_coeff = pd.DataFrame(coeff).T
lasso_coeff.rename(columns = {0:'Lasso_beta'}, inplace = True)
```

```
Test MSE of lasso regression is:  2.3610589089053495
```

Figure 3: Fitting LASSO on training and predicting on test data