RM 294 – Optimization I
Project 3: Non-Linear Programming

Manvi Goyal (mg65952), Shreyansh Agrawal (sa55742), Rianna Patel (rnp599), Nevin Arimilli (na24887)

Purpose:
Variable selection for regression has always been a challenge in predictive analysis. Large numbers of variables can often lead to overfitting of the data and computational power. Hence, variable selection comes into play so that model performance is improved by removing irrelevant features, thereby, reducing overall complexity of the model and generalizing the model for unseen data. It is important to select a set of variables that reduces multicollinearity and noise to provide the best fit for the model so accurate predictions can be made.

Overview:
As explained above, variable selection is a very important part of predictive analytic models.
There are several methods that have been introduced and are nowadays used to select the best possible combination of features.Through this project, we will be exploring two methods for selecting the variables, LASSO regression and Mixed-Integer Quadratic Program. LASSO regression regularizes model parameters by adding a "shrinkage" component and hence, reducing some of the regression coefficients to zero. The model is then fit after this features selection phase using the left features. This method is very commonly used. Another way to perform variable selection is through optimizing the cost function through mixed integer quadratic programs (MIQP).

In this report, we will be analyzing features selection through both of these methods and recommend which method should be adopted by our firm for the variable selection in predictive analytics. We will also highlight the reason behind selecting a particular method and advantages and disadvantages of both these methods.

Method 1: Direct Variable Selection Method – MIQP

Overview:
Mixed Integer Quadratic Programming (MIQP) has the objective function that consists of two parts: Quadratic form and Linear form. It has a non-linear objective function. We select the best features by minimizing the ordinary least squares i.e. squared difference between predicted values and the true values.

In our case, we have m independent variables X and a dependent variable Y. For the purpose of feature selection, we incorporate binary variables, zj. The value of zj helps in forcing the corresponding β to be zero if zj is zero, using the big-M method. We also introduce a hyperparameter k which helps us to specify the maximum number of features to be selected and the value of k is chosen using the cross-validation. We have used a 10-fold cross-validation approach.

We know that the general form of the Quadratic Programming is:

Objective Function and Constraints:

$$\min_{x} x^T Q\, x + c^T x$$
$$\text{s.t. } A\, x \leq b,\ x \geq 0$$

Below are values that will be used for implementing the MIQP:

***Total Number of Decision Variables*:** 2m+1

1. m+1 beta values (intercept + coefficients) - $\beta_o, \beta_1, \ldots, \beta m$
2. m binary variables - $z_1, z_2, \ldots, z$

Objective Function:

$$\min_{\beta, z} \beta^T (X^T X)\, \beta + (-2\, y^T X)\, \beta.$$

On comparing with Quadratic form, we know:

$$Q = X^T X \text{ and } c = -2 y^T X$$

***Total number of Constraints*:** 2m+1

1. $\sum_{i=1}^{m} z_i \leq k$ --> 1 constraint
2. $-z_i M \leq \beta_i$ *where* $i = 1,2,\ldots,m$ --> m constraints
3. $\beta_i \geq z_i M$ *where* $i = 1,2,\ldots,m$ --> m constraints

Method:

10-fold cross-validation was performed on the training data in order to find the optimal value of k. The possible values of k are 5,10,15,20,25,30,35,40,45,50. The optimal value of k was decided on the basis of cross-validation error i.e. the average error for a given k considering all 10 folds.

Firstly, we splitted the training data in 10 unique subsets randomly by defining the cross-validation function ***(Fig5)***. For each k and each fold, the MIQP model was optimized and the error (mse) and beta values are stored in a dataframe ***(Fig7)***. For each k, we can get the average MSE across the 10 folds and the k corresponding to the minimum cross validation error was selected (k=10 was chosen)

Results:

Below are values of Total MSE observed for different values of k after we run the MIQP model:

	Total SSE
5	914.11453
10	703.96601
15	779.45883
20	793.42285
25	782.62708
30	813.98226
35	817.32599
40	828.47247
45	836.88594
50	840.45402

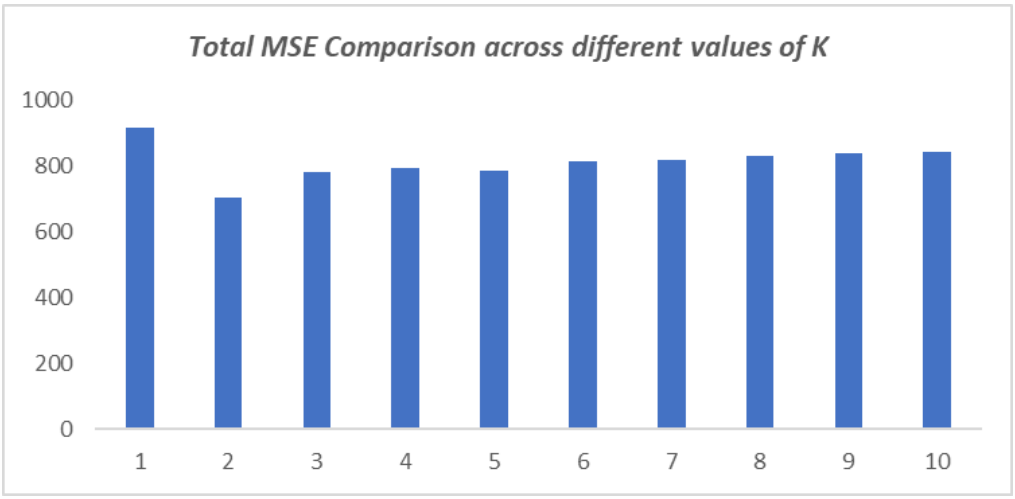

Fig1. MSE value for different k

As observed from above graphs, we get the lowest value of Total MSE for **k=10**. Therefore, the optimal value of k for the MIQP model is 10. Hence, we will now use the value of k as 10, use the entire training data to fit the MIQP model ***(Fig8)***. This model was used to predict the target variable on the test set and evaluate the MSE. **MSE on Test = 2.3365, MSE on Train = 2.3919**.

```
The test error is: 2.33654
```

Method 2: Indirect Variable Selection Method – LASSO Regression

Overview:

LASSO regression is an indirect method for selecting the features. It uses the *"Shrinkage" or "Penalty"* term to reduce the features by forcing them to be zero. Hence, it introduces a penalty term along with the ordinary least squares to select the best possible combination of features. It is also known as the L1 regularization method. With the increase in value of the regularization parameter (λ), more and more features are shrinked towards zero, hence, selecting the right value of λ *(penalty term)* is very important. Lasso does not penalize the intercept.

Objective Function:

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

Method:

We first standardized the data ***(Fig10)*** using StandardScaler so that all the features have a zero mean and standard deviation of 1. Then we fit the LASSO regression using the 10-fold cross-validation. Scikit-learn reduces a lot of complexity when compared with the cross-validation process carried out for Gurobi Optimization. With a single line of code, we get the best value of λ (lambda) and best number of features. Using this value of lambda, we fit the LASSO model on the complete training data and compute the MSE on the testing data ***(Fig9)***.

Results:

Using the 10-fold cross-validation, the best value of lambda is 0.084. We get 18 as the total number of best features. Using the model, we evaluated the model performance on test and train data using MSE. Below is a snapshot of the same.

```
Optimal Lambda is : 0.08471942409934509
Total non-zero features using Lasso: 18
The value of MSE on the test data is: 2.35667
The value of MSE on the train data is: 2.38644
```

Comparing MIQP and LASSO Regression:

1) Firstly, we compared the beta values and intercept values computed from both the models. As observed from the graph, the value of coefficients are comparable. The best number of features by LASSO is 18 whereas for MIQP it is 10. Moreover, we can see that the values of coefficient for the 8 additional features computed through LASSO have very low coefficient values. MIQP does a better job in selecting the features which explain the most variance in Y and disregards the features with low coefficient values.

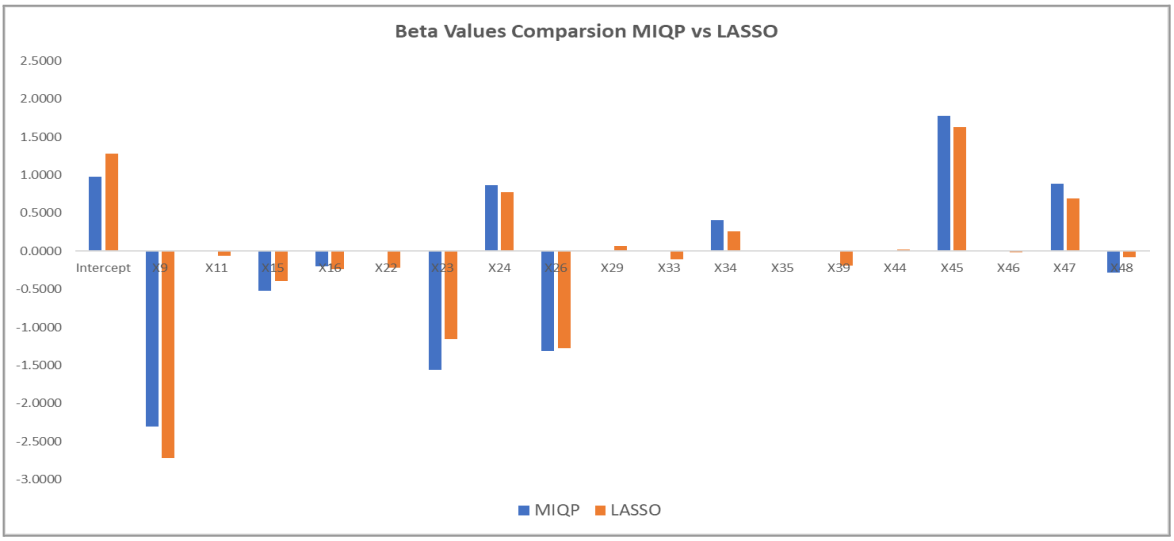

Fig2. Beta Values Comparison MIQP vs LASSO

2) Below is the graph representing the test set predictions using both the methods. We can see that predictions from both the models are comparable and are tracking well with the target variable.

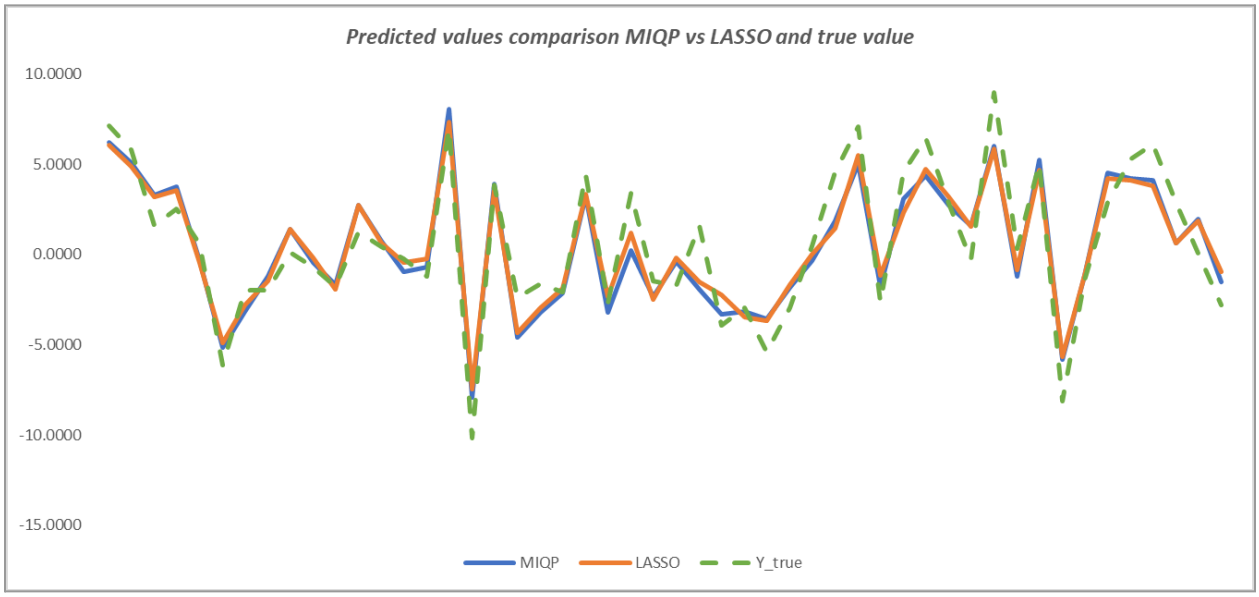

Fig.3 Predicted values comparison MIQP vs LASSO and true value

3) On comparing the test and training MSE, it can be seen that the training MSE is comparable for both the models. However, the test MSE for MIQP is slightly lower than the LASSO regression model. It is a less complex model and is able to generalize well with the unseen data. Hence, for this particular dataset, MIQP performed better.

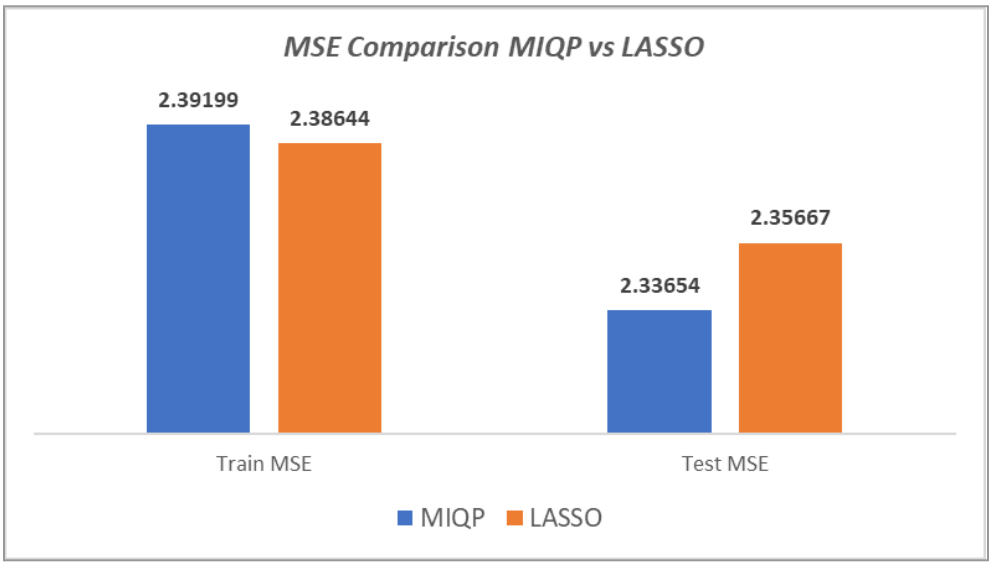

Fig4. Test and Train MSE Comparison MIQP vs LASSO

Recommendations and Conclusions:

From the above comparison, we see that LASSO selects 18 features to achieve a test MSE of 2.35 and MIQP gives slightly better performance with a test MSE of 2.33 and with lower number of features. MIQP is computationally expensive and gives better results but LASSO is quick and gives comparable results. So, both these techniques have some advantages and disadvantages which are outlined in detail below.

MIQP:

1) Advantages
    a) MIQP is a more explicit and direct way of selecting features of the model which gives a bit more control of the results
    b) Selected features do not have very low value of coefficients and hence reduces the variability in the model and improves the interpretability and generalization
    c) It gives less complex model i.e. select lesser number of features
2) Disadvantages
    a) It requires a lot of computational power and time especially when the datasets or features are very large
    b) Output depends on how much time the code is run, so that introduces variability in the results.
    c) It is not an automated system (like scikit-learn), we need to write the code manually which is time consuming and can pose difficulty to non-coders.
    d) Functionality not present in MLOPs platform

LASSO:

1) Advantages
    a) It has automated system, one-line of code gives us the optimal value of lambda and beta number of features required
    b) It helps in avoiding overfitting by shrinking coefficients to zero
    c) It is computationally very efficient and is integrated as a feature in all the MLOPs platforms.
2) Disadvantages
    a) It does not work well when the number of features are very high as compared to the number of data points ($n << p$)
    b) If the features are correlated, it may randomly select any one of them
    c) Best performance can result in selecting higher number of features which can make the model slightly more complex and less generalized
    d) Need to standardize the feature prior to introducing L1 penalty.

On the basis of our analysis along with the advantages and disadvantages listed for each method, we do not recommend the company to completely rely primarily on one method. ***The method selected should depend on the use case as our business requirement and technical capabilities.*** We should take into account multiple factors while selecting the model.

LASSO can be preferred when:

a) Limited computational power and turnaround time is less
b) When you need to run the model on a frequent basis (say you get updated data weekly), Lasso will save both time and computational resource massively
c) It's wrapped up neatly in a package which is easy to implement. Employees with limited coding background can also implement it
d) It’s an open source package, unlike Gurobi which needs to be purchased to perform MIQP

MIQP can be preferred when:

a) You have abundant computational power and the time is not a constraint
b) You need to select a pre-specified number of features
c) You want to explain most of the variance in target variable based on minimum pre specified features when cost of procuring data for additional features is high

In a nutshell, if a model does not need to be run frequently and computing power is not an issue, MIQP can have an edge over LASSO when model performance and generalization is considered. However, its computationally expensive and complex to code. With the various tradeoffs between computational efficiency,preferred error margins, and complexity to code and deploy, we should take the time to discuss these limitations to determine the best feature selection method to suit the business purpose.

Appendix:

```python
# outputs the n folds for cross validation
def gen_test_val_split(X_train, n_folds):

    n_elements = int(X_train.shape[0]/n_folds)
    cv_list = []

    a = np.arange(0,X_train.shape[0])

    for i in range(0,n_folds):
        cv = np.random.choice(a,size=n_elements,replace=False)
        cv_list.append(cv)
        a = a[~np.isin(a,cv)]

    return cv_list
```

Fig5. Defining code for cross-validation for MIQP

```python
def optimize_miqp(X,y,k,TIME_LIMIT,M):

    # number of rows
    n = X.shape[0]
    # number of variables
    m = X.shape[1]-1

    # Quadratic part Q
    obj_quad= np.zeros((2*m+1, 2*m+1))
    obj_quad[0:m+1,0:m+1] = X.T @ X

    # Linear Part C
    obj_lin = np.zeros(2*m+1)
    obj_lin[:(m+1)] = -2*y.T @ X

    # Defining the constraints

    # Creating the A matrix
    A = np.zeros((2*m+1, 2*m+1))
    sense = ['']*A.shape[0]
    b = np.zeros(2*m+1)

    # big M constraint: b_j <= Mz_j
    np.fill_diagonal(A[:m, 1:m+1], 1)
    np.fill_diagonal(A[:m, m+1:2*m+1], -M)

    # big M constraint: -Mz_j <= b_j
    np.fill_diagonal(A[m:-1, 1:m+1], 1)
    np.fill_diagonal(A[m:-1, m+1:2*m+1], M)

    # Sum of the number of independent betas should be equal to k
    A[-1, m+1:] = 1
    sense = np.array(['<']*m + ['>']*m + ['<'])
    lb = np.array([np.NINF]+[-M]*m+[np.NINF]*m)

    b = np.concatenate((np.zeros(2*m), [k]))

    opt_model = gp.Model()
    opt_model_x = opt_model.addMVar(len(obj_quad),vtype=['C']*(m+1)+['B']*m, lb=lb)
    opt_model_con = opt_model.addMConstrs(A, opt_model_x, sense, b)
    opt_model.setMObjective(obj_quad, obj_lin, 0, sense=gp.GRB.MINIMIZE)
    opt_model.Params.OutputFlag = 0
    opt_model.setParam('TimeLimit', TIME_LIMIT)
    opt_model.optimize()

    return opt_model
```

Fig6. Defining the equations for MIQP in Gurobi

```
# running 10-fold cross validation for different values of k

X = np.array(train_df.iloc[0:,1:])
y = np.array(train_df['y'])

cv_index = gen_test_val_split(X_train, n_folds)
k_list=np.linspace(5,50,10)
a = np.arange(0,train_df.shape[1])

# list to store mse and beta values for each iteration
mse_list = []
beta_list = []

# dataframe to store the outputs of all th eietrations
col_list = ['k','n_fold']
col_list.extend(train_df.columns[1:])
col_list.extend(['mse'])
df_history = pd.DataFrame(columns=col_list)

counter = 0
for k in k_list:

    for i in range(0,n_folds):

        X_val = X[cv_index[i]]
        y_val = y[cv_index[i]]
        X_train = X[a[~np.isin(a,cv_index[i])]]
        y_train = y[a[~np.isin(a,cv_index[i])]]

        model = optimize_miqp(X=X_train,y=y_train,k=k)

        m=X_train.shape[1]-1
        beta = np.array(model.X)[0:m+1]
        beta_list.append(beta)

        y_pred = X_val@beta
        mse = mean_squared_error(y_pred, y_val)
        mse_list.append(mse)

        row_arr = [k,i]
        row_arr.extend(beta)
        row_arr.extend([mse])

        # storing to df
        df_history.loc[counter] = row_arr

        counter = counter+1

df_history.to_csv("miqp_outputs.csv")
```

Fig7. Running MIQP for all possible values of k

```
# now retraining the model using the complete training data with k=10
final_model = optimize_miqp(X=X_train,y=Y_train,k=10,TIME_LIMIT=6000,M=100)

m=X_train.shape[1]-1

# selcting the value of coefficients
beta = np.array(final_model.X)[0:m+1]

# predicting on test data
y_pred = X_test@beta

#mse = np.power(y_pred-Y_test,2).sum()
print("The test error is: "+ str(round(mean_squared_error(Y_test, y_pred),5)))
```

```
C:\Users\shrey\AppData\Local\Temp\ipykernel_28712\112114350.py:2: DeprecationWarning: Deprecated, use Model.addMConstr() instead
  final_model = optimize_miqp(X=X_train,y=Y_train,k=10,TIME_LIMIT=6000,M=100)

The test error is: 2.33654
```

Fig8. Solving MIQP for best value of k i.e. k=10 on entire train data

```
# fitting the lasso model to get the optimal lambda
lasso_regression_cv = linear_model.LassoCV(cv=10).fit(X_train, Y_train)
print('Optimal Lambda is :', lasso_regression_cv.alpha_)

# calculating number of features
print('Total non-zero features using Lasso:', (lasso_regression_cv.coef_ != 0).sum())

#predicting value of y on the test data
y_pred=lasso_regression_cv.predict(X_test)

#calculating MSE on test data
print('The value of MSE on the test data is:', round(mean_squared_error(Y_test, y_pred),5))

#predicting value of y on the train data
y_pred=lasso_regression_cv.predict(X_train)

#calculating MSE on test data
print('The value of MSE on the train data is:', round(mean_squared_error(Y_train, y_pred),5))
```

Fig9. Using Scikit-learn for LASSO regression

```
# standardizing the dataset
from sklearn.preprocessing import StandardScaler

# fitting on x train
std_trans = StandardScaler()
std_trans.fit(X_train)

#transforming on X_train and X_test
X_train_std = std_trans.transform(X_train)
X_test_std = std_trans.transform(X_test)
```

Fig10. Standardizing the features using Scikit-learn preprocessing