Optimization I
Project 3: Non-Linear Programming

Submitted to:

Dr. Daniel Mitchell, Professor
Optimization I, R M 294
05240

Prepared by:
Krittika Deshwal (kd29275)
Mayank Gupta (mg66426)
Rahull Borana (rb47374)
Sanyam Jain (sj33448)

Master of Science Business Analytics Program
The University of Texas at Austin
Austin, Texas
Fall 2023

Table of Contents

Introduction and Scope of the Problem 3

Method 1: Direct Variable Selection Method – MIQP 4

Overview: 4

Methodology: 5

Results: 7

Method 2: Indirect Feature Selection via LASSO Regression 9

Overview: 9

Methodology: 9

Results: 11

Comparison (MIQP using Gurobi vs Lasso) 12

1. Comparing the errors across both methods: 12
2. Comparison of the Beta Values: 13
3. Predicted v/s Actual Output: 14

Recommendations 15

Conclusion 16

Introduction and Scope of the Problem

This project compares variable selection in regression using MIQP (Mixed Integer Quadratic Programming) and LASSO methods. MIQP enforces variable inclusion through optimization, while LASSO minimizes the sum of squared residuals with an added penalty which may shrink coefficients, possibly even to zero. The objective is to assess if LASSO's shrinkage technique outperforms MIQP's direct selection method. Cross-validation helps optimize the respective hyperparameters (k for MIQP and λ for LASSO).

MIQP problems are typically very demanding computationally. Here we attempt to solve a MIQP problem using an advanced optimization software Gurobi in Python.

To implement Lasso regression, we use our standard scikit-learn which features the algorithm for Lasso in Python.

Comparing the two approaches:

MIQP (Mixed Integer Quadratic Programming):

- *Parameter Selection*: Direct variable selection via optimization and binary variables.
- *Model Complexity Control*: Determines variable inclusion, impacting model complexity.
- *Interpretability*: May result in more complex models with a larger number of variables.

LASSO (Least Absolute Shrinkage and Selection Operator):

- *Parameter Selection*: Penalizes coefficients, forcing some to zero for feature selection.
- *Model Complexity Control*: Encourages fewer predictors with non-zero coefficients for simplicity.
- *Interpretability*: Favors sparsity, leading to more interpretable models with fewer variables.

Method 1: Direct Variable Selection Method – MIQP

Overview:

The objective function, decision variables and constraints to formulate the MIQP model are as follows:

Objective Function:

MIQP employs a nonlinear objective function to optimize feature selection by minimizing ordinary least squares i.e., sum of squared difference between predicted and actual value for each data point.

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

Here β_o is the intercept and β_1 - β_m are the coefficient of the independent variables x_1-x_m in the dataset.

Decision Variables:

There are 'm' independent variables (X) and one dependent variable (Y) in the data. We consider (2m+1) decision variables in total-

1. m+1 beta values (including intercept and coefficients) - β_o, β_1, ..., β_m

2. m binary variables - z_1, z_2, ..., z_m

Binary variables help in selecting 'm' independent variables for the equation as they decide the beta coefficients to be zero or non-zero using the big-M method.

Constraints:

There are 2m constraints related to β and z values:

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

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

A hyperparameter 'k' is used to determine the maximum number of features to select.

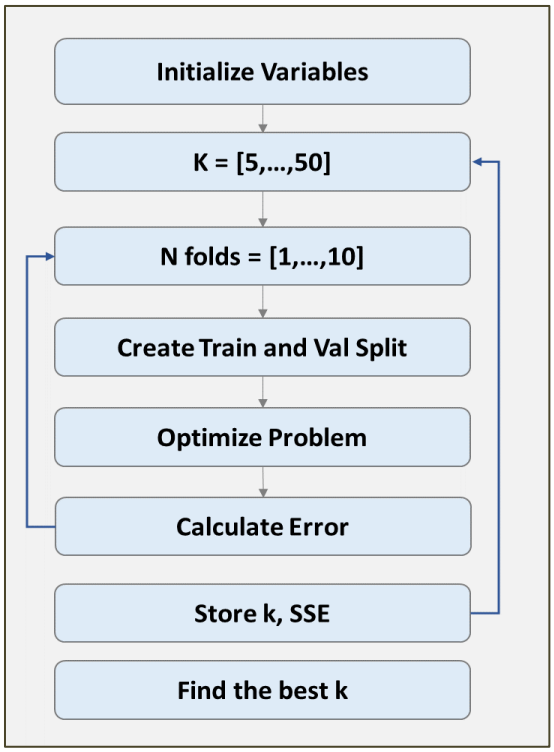
First 'm' constraints are the big-M constraints which force beta variables to be zero or non-zero. An important point to note is that we don't impose the constraint on the intercept variable (β_o)

to become zero. The second set of constraints are for the binary variables ‘z’ where we want only ‘k’ z variables to be non-zero.

Methodology:

To determine the best value of k, we conducted 10-fold cross-validation on the training dataset. The optimal k was chosen based on the lowest cross-validation error, which is the mean error across all 10 folds for each k value.

The process flow is illustrated in the flowchart below.



We applied the MIQP method on k values of 5, 10, 15, 20, 25, 30, 35, 40, 45, and 50.

```
#list of k values to try
k_options = np.arange(5,51,5)
print(k_options)
```

The data was divided into 10 distinct subsets of training and testing data for each fold.

```
#splitting training df into 10 folds for performing cross-validation
split_ind = KFold(n_splits=num_k_folds).split(X_train)

k_folds_dict = {}
for i, ind in enumerate(split_ind):
    #temp dictionary
    index_dict = {}
    # The training indices are in the first item of the tuple and the testing indices are in the second.
    index_dict['train_indices'] = ind[0]
    index_dict['test_indices'] = ind[1]
    k_folds_dict[i] = index_dict
```

For each value of k, the data, Gurobi model and decision variables are initialized.

```
    for k in k_options:
        print(f"for k = {k}")
        for fold, indices_dict in k_folds_dict.items():
            # Select data for the current fold
            X_fold, y_fold = X_train[indices_dict['train_indices']], y_train[indices_dict['train_indices']]

            # Create a Gurobi model
            model = gp.Model()

            # variables
            beta = model.addMVar(X_train.shape[1] + 1, name="beta", lb=-np.inf)
            z = model.addMVar(X_train.shape[1], vtype=GRB.BINARY, name="z")
```

Objective Function and constraints were added and subsequently the MIQP model was optimized.

```
    # Objective function
    residuals = gp.quicksum((y_fold[j] - (beta[0] + gp.quicksum(beta[i + 1] * X_fold[j, i] for i in range(X_train.shape[1]))))*\
        (y_fold[j] - (beta[0] + gp.quicksum(beta[i + 1] * X_fold[j, i] for i in range(X_train.shape[1])))) for j in range(X_fold.shape[0]))
    model.setObjective(residuals, GRB.MINIMIZE)

    # Add constraints
    M = 100 # Set a large value for M
    M2 = -M

    bigMconspos = model.addConstrs((beta[i+1] <= M*z[i]) for i in range(50))
    bigMconsneg = model.addConstrs((beta[i+1] >= M2*z[i]) for i in range(50))

    z_con = model.addConstr(gp.quicksum(z[j] for j in range(X_train.shape[1])) == k)

    # Optimize the model
    model.Params.OutputFlag = 0
    model.Params.TimeLimit = grb_runtime
    model.optimize()
```

The resulting sum of squared error (SSE) was recorded in a dataframe for all values of k for each fold. The dataframe snippet is attached in the results section.

```
#calculate residuals on validation set
X_fold_val, y_fold_val = X_train[indices_dict['test_indices']], y_train[indices_dict['test_indices']]
pred = predictions(X_fold_val,beta.x)
error = sse(y_fold_val,pred)
print(f"fold {fold} MSE on val set is = {error}")

# Store the results in the DataFrame
grb_df.at[fold, k] = error

#Save the results to csv file
grb_df.to_csv('results.csv',index=False)
```

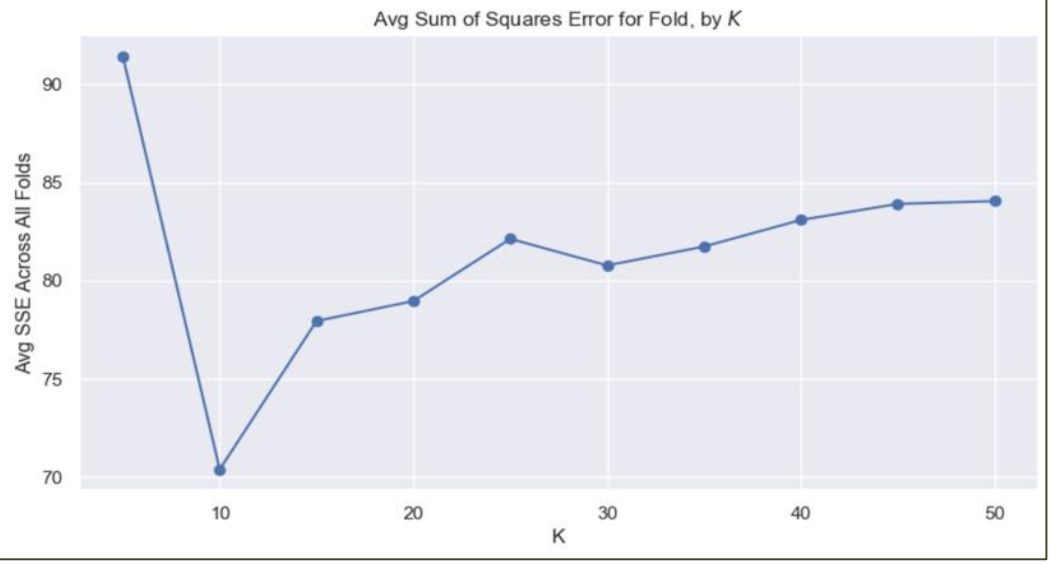
Results:

The values of Sum of Squared errors for every value of k for each fold is given below.

	5	10	15	20	25	30	35	40	45	50
0	121.08483	83.22601	84.98515	82.43595	92.91026	95.66110	97.65884	99.99508	96.91679	97.57031
1	76.27096	68.72295	85.44507	91.70347	102.49541	98.26731	96.42208	96.91586	100.55214	101.13772
2	76.40846	62.50184	70.91066	83.65107	88.56933	73.46238	78.72707	80.83133	79.92967	79.96562
3	72.64028	62.72359	71.70770	71.71293	70.69936	72.59391	76.85514	77.86652	78.59641	78.33711
4	118.95958	82.00438	83.84794	80.40048	80.64813	86.74073	90.89855	87.67083	86.81163	86.93229
5	82.97925	61.88108	65.54801	59.53692	75.45312	65.59325	61.14406	66.13959	67.22787	67.36372
6	76.18434	65.08857	69.81931	70.62222	69.18039	70.47346	67.89549	71.89055	75.20079	75.06871
7	60.29282	48.66334	59.89032	54.19826	56.43211	55.66685	54.96695	60.79733	62.11971	63.55508
8	133.75785	93.39369	99.85986	105.95280	98.82562	105.08918	104.69604	103.73865	104.83067	104.08194
9	95.53616	75.76057	87.45220	89.38065	86.02599	84.18368	88.06178	84.99322	86.84317	86.44153

By averaging the SSE for each k across all 10 folds, we identified the k value that yielded the minimum cross-validation error. In this case, **k=10** was selected as the optimal value for the MIQP model as illustrated below.

	Avg SSE
5	91.41145
10	70.39660
15	77.94662
20	78.95948
25	82.12397
30	80.77318
35	81.73260
40	83.08390
45	83.90288
50	84.04540



The model was applied to predict the target variable in the test dataset, and its performance was assessed using the sum of squared error (SSE). The results are as follows:

```
Optimal K Value for MIQP : 10
SSE on Train Data for MIQP : 597.99
SSE on Test Data for MIQP : 116.82
```

Method 2: Indirect Feature Selection via LASSO Regression

Overview:

LASSO regression represents an indirect approach to feature selection. This method incorporates a "Shrinkage" or "Penalty" component, which effectively reduces the number of features by shrinking some coefficients to zero.

This approach adds a penalty term to the ordinary least squares method, aiding in the identification of the most effective feature combination. This technique is also referred to as L1 regularization. As the regularization parameter (λ) increases, more features are compressed towards zero, making the correct choice of λ (the penalty term) crucial. Notably, LASSO does not apply this penalty to the intercept.

Objective Function:

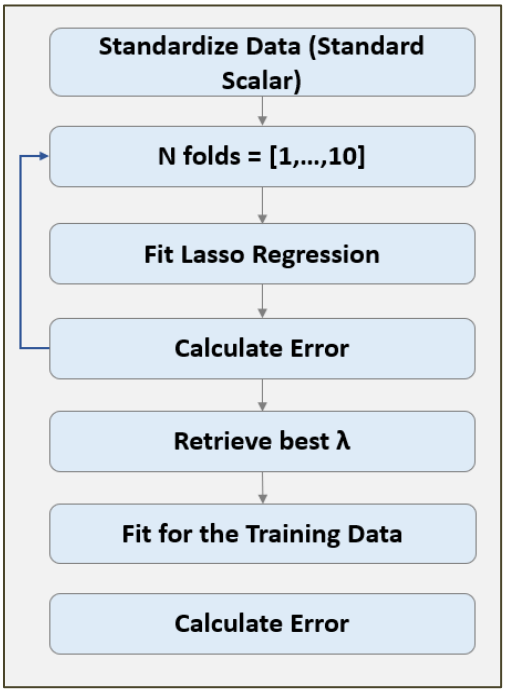
$$\min_{\beta} \sum_{i=1}^{n} (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_m x_{im} - y_i)^2 + \lambda \sum_{j=1}^{m} |\beta_j|$$

Here we have m independent variables 'X', a dependent variable 'y' and 'n' number of observations.

Methodology:

The data science pipeline using Lasso regression with cross-validation involves standardizing the data, performing cross-validation with multiple folds, fitting Lasso regression models, calculating error metrics, retrieving the best regularization parameter, and evaluating the performance of the tuned model. This pipeline provides a robust approach to building and evaluating Lasso regression models for predictive tasks.

The pipeline is presented in the following flow diagram:



Python Implementation:

The LassoCV method efficiently handles the cross-validation process, providing a straightforward way to find the optimal regularization parameter for LASSO regression.

Using LassoCv to get the best lambda and training the data with this value:

```
lassocv = LassoCV(alphas = None, cv = 10)
lassocv.fit(X_train_std, y_train)
```

Predict on hold out set and calculate the metrics:

```
lasso_predictions_test = lassocv.predict(X_test_std)
lasso_sse_test = sse(y_test,lasso_predictions_test)
lasso_mse_test = mean_squared_error(y_test,lasso_predictions_test)
lasso_r_squared_test = r2_score(y_test,lasso_predictions_test)
```

Results:

The Lasso regression, guided by a lambda of 0.0847 from cross-validation, displayed promising test set performance with an SSE around 117.833, while the training set SSE was approximately 596.611. This suggests potential for improvement in generalization, yet the model effectively highlighted 18 essential features out of the available variables.

```
Best Value of lambda: 0.08471942409934505
SSE on train data for Lasso: 596.61112
SSE on test data for Lasso: 117.83332
Total non-zero features using Lasso: 18
```

Comparison (MIQP using Gurobi vs Lasso)

1. Comparing the errors across both methods:

Based on the error values below, both methods seem to perform similarly. MIQP method has a slightly lower SSE compared to the LASSO method. This suggests that, on the test data, the MIQP method is performing slightly better. It is a less complex model and can generalize well with unseen data. Hence, for this dataset, MIQP performed better.

Train results

	SSE	MSE	R_Squared
Gurobi_Method_Metrics	597.99632	2.39199	0.87834
Lasso_Metrics	596.61112	2.38644	0.87862

Test results

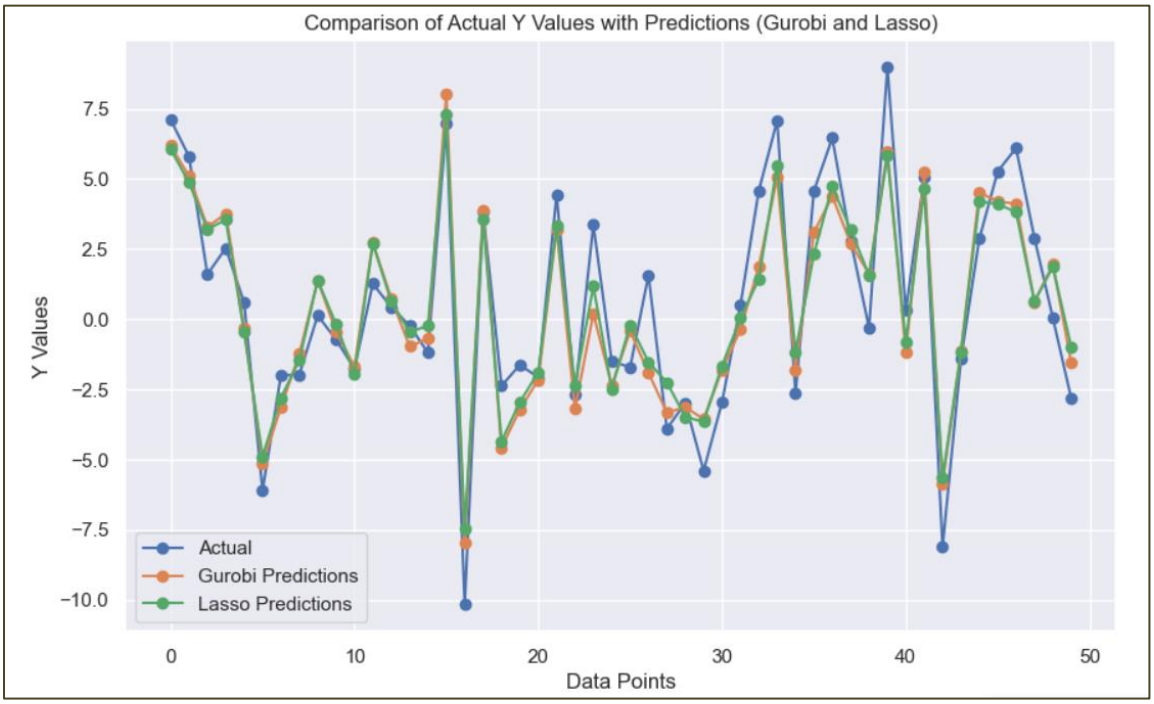
	SSE	MSE	R_Squared
Gurobi_Method_Metrics	116.82720	2.33654	0.85867
Lasso_Metrics	117.83332	2.35667	0.85745

2. Comparison of the Beta Values:

LASSO identifies the optimal number of ***features as 18***, whereas ***MIQP determines it to be 10***. Additionally, it is observed that the coefficients associated with the 8 extra features identified by LASSO are notably low. MIQP excels in the selection of features that contribute significantly to explaining the variance in Y, effectively excluding features with low coefficient values.

3. Predicted v/s Actual Output:

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



Recommendations

In the comparison presented above, LASSO opts for 18 features, resulting in a test Mean Squared Error (MSE) of 2.35. On the other hand, MIQP exhibits slightly superior performance, achieving a test MSE of 2.33 with a lower count of features. While MIQP is computationally intensive, delivering improved results, LASSO stands out for its speed and provides results that are comparable. Hence, both techniques possess their own set of advantages and disadvantages, which are elaborated as follows:

	MIQP	Lasso
Advantages	a) MIQP offers a more explicit and direct method for selecting model features, providing greater control over the outcomes.	a) LASSO provides an automated system; a single line of code yields optimal values for lambda and the required number of features.
	b) The selected features in MIQP have coefficients that are not very low, reducing variability in the model, enhancing interpretability, and improving generalization.	b) It aids in preventing overfitting by shrinking coefficients to zero.
	c) MIQP results in a less complex model by selecting a smaller number of features.	c) Computationally efficient, LASSO is integrated into all major MLOPs platforms.
Disadvantages	a) It demands substantial computational power and time, particularly with large datasets or numerous features.	a) LASSO's performance diminishes when the number of features is substantially higher than the number of data points (n << p).
	b) The output is influenced by the runtime, introducing variability in the results.	b) In the presence of correlated features, LASSO may randomly select any one of them.
	c) It lacks automation (unlike scikit-learn), requiring manual code writing, which is time-consuming and can pose challenges for non-coders.	c) Optimal performance might result in selecting a higher number of features, making the model slightly more complex and less generalized.
	d) The functionality is absent in MLOPs platforms.	d) Feature standardization is necessary before applying L1 penalty.

Based on our analysis and considering the pros and cons outlined for each method, we advise against exclusively relying on one method for the company's needs. The choice of method should be contingent upon the specific use case, aligning with our business requirements and technical capabilities. It is essential to consider multiple factors when deciding on the most suitable model.

We recommend consider using LASSO under the following conditions:

a) Limited computational power and a quick turnaround time are crucial.
b) Frequent model runs are required, especially with regularly updated data (e.g., weekly updates), as LASSO significantly saves both time and computational resources.
c) Easy implementation is essential, as LASSO is neatly packaged and can be utilized by individuals with limited coding experience.
d) Open-source accessibility is preferred, distinguishing it from Gurobi, which requires a purchase for MIQP tasks.

We recommend consider using LASSO under the following conditions:

a) Abundant computational power is available, and time constraints are not a primary concern.
b) The goal is to select a predetermined number of features.
c) Explaining a substantial portion of the variance in the target variable is necessary with a minimal set of pre-specified features, particularly when acquiring additional data for extra features incurs high costs.

Conclusion

In summary, when a model does not require frequent runs and computational power is not a limiting factor, MIQP may outperform LASSO in terms of model performance and generalization. However, it is crucial to note that MIQP comes with computational expenses and coding complexities. Given the tradeoffs involving computational efficiency, desired error margins, and coding and deployment complexity, a thorough discussion is necessary to carefully weigh these limitations and select the most suitable feature selection method aligned with the business objectives.