Variable Selection for Regression
(Nonlinear Programming)

Comparison of Mixed Integer Quadratic Programming & Lasso Regression for variable selection

Group 30
Abdullah Khan (ak46996)
Aritra Chowdhury (ac79277)
Shiyong Liu (sl52967)
Vishal Gupta (vg22846)

Introduction

In nearly every discipline of science and engineering, modern advances in technology have made it easier to collect more and more data. With prediction being the goal of so many models, it has become extremely difficult to choose an appropriate subset of features to base the model on. Furthermore, particularly when there are more predictors than observations, an omnipresent danger is that of overfitting, increasing the importance of optimal variable selection even further. This results in three key objectives behind variable selection:

- Understanding relationships among variables
- Weeding out unnecessary variables
- Building a generalizable predictive model

In this project, we compare two major methods of variable selection: the everlasting **Lasso Regression** and the more recent **Mixed Integer Quadratic Programming (MIQP)**. We do so by cross-validating our prediction errors against a validation set multiple times and then evaluating the total sum of squared errors. We finally plot a comparison of the final coefficients in a regression model and arrive at a final conclusion.

Mixed Integer Quadratic Programming

We know that given a dataset of m independent variables, X, and a dependent variable, y, the standard ordinary least squares problem is formulated as follows.

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

We first choose a value of k, which is the number of variables to be selected. For the range of k = 5, 10, 15, …, 50, we execute a 10-fold cross-validation on the training set and calculate the error as the sum of squared errors of all validation sets for each k. We then find the value of k that corresponds to the minimum error. We finally calculate the predictions for y on the test set and the final test error.

The MIQP problem is formulated as follows -

Decision Variables

1. β_j : coefficient of variable j $(m+1)$
2. z_j : selector whether variable j should be considered or not in the regression (m) - Binary

Total Decision variables - $2m+1$

Objective

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

Constraints

1. Sum of all variable present in the regression equation should be equal to *k*

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

2. Coefficient of each variable j, where j >= 1, β_j should be less than $M\, z_j$

$$\beta_j - M z_j \leq 0 \;\; for \; j = 1, 2, \ldots, m$$

3. Coefficient of each variable j, where j >= 1, β_j should be more than $-M\, z_j$

$$\beta_j + M z_j \geq 0 \;\; for \; j = 1, 2, \ldots, m$$

Total Constraints - $2m+1$

We began by creating a list for cross-validation dataset for 10-folds to evaluate the performance of different k values:

Create Cross Validation List

```
# creating function to get cross validation dataset list
def create_cross_val(X, cv=10):
    # empty list to store index
    cv_list = []

    kf = KFold(n_splits=cv)
    # getting train and validation index
    for train_idx, val_idx in kf.split(X):
        cv_list.append((train_idx, val_idx))

    return cv_list
```

This is followed by the implementation of our direct variable selection function:

```python
def create_direct_variable_selection(X, y, big_m, k=5, time_limit=10):

    m = X.shape[1] - 1

    A = np.zeros((2*m + 1, 2*m + 1))
    b = np.zeros(2*m + 1)
    sense = ['']*(A.shape[0])
    obj = np.zeros(A.shape[1])
    Q = np.zeros((A.shape[1], A.shape[1]))
    vtype = ['C']*(m + 1) + ['B']*m
    lb = np.array([-np.inf]*(m + 1) + [0]*m)

    # setting first constraint
    A[0, m+1:] = 1
    b[0] = k
    sense[0] = '<'


    # setting second & third constraint together
    for i in range(1, m+1):
        A[i, i] = 1
        A[i, i + m] = -big_m
        b[i] = 0
        sense[i] = '<'

        A[i + m, i] = 1
        A[i + m, i + m] = big_m
        b[i + m] = 0
        sense[i + m] = '>'

    # setting objective function
    obj[:m+1] = -2*y @ X
    Q[:m+1, :m+1] = X.T @ X

    # getting the optimal value
    model = create_model(A, sense, b, obj, opt=gp.GRB.MINIMIZE, vtype=vtype, Q=Q, time_limit=time_limit, lb=lb)

    return model
```

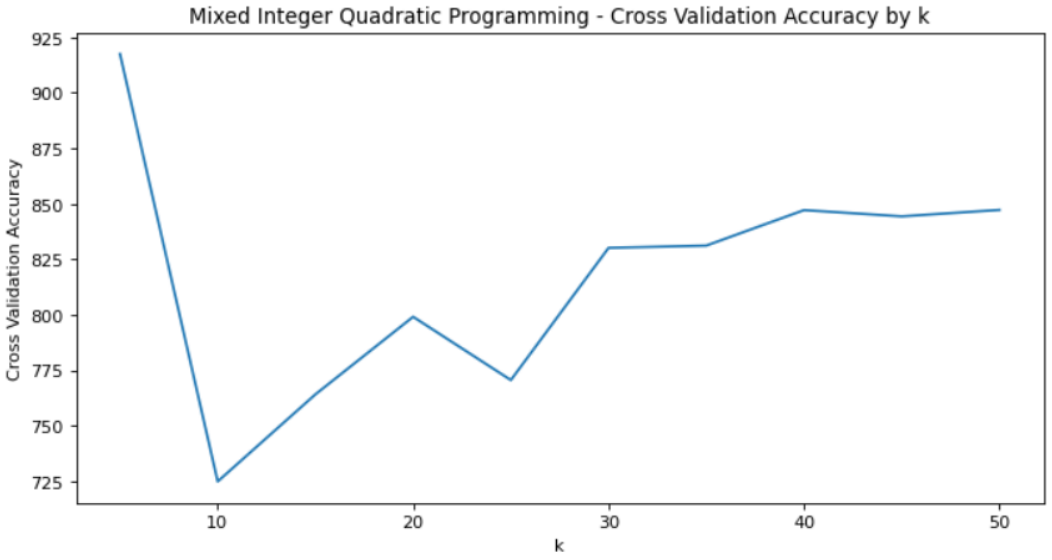
Using this function, we fit the model and perform 10-fold cross-validation to evaluate its performance.

Results

Sum square error is calculated after the MIQP model gives the number of variables with non-zero coefficient value. The error formulation & error metrics for different values of as below:

$$(X\beta - y)^T * (X\beta - y)$$

	k	cross_val_accuracy
0	5	917.479061
1	10	724.787631
2	15	764.049938
3	20	799.009127
4	25	770.482828
5	30	830.082402
6	35	831.104008
7	40	847.095515
8	45	844.283143
9	50	847.184545



The minimum cross-validation error is obtained for k = 10.

Now that we have obtained an optimal value for k through cross-validation, we assess the performance of our optimized model on our test set by using our test accuracy helper function. **We obtained a prediction error on our test set of 116.83.**

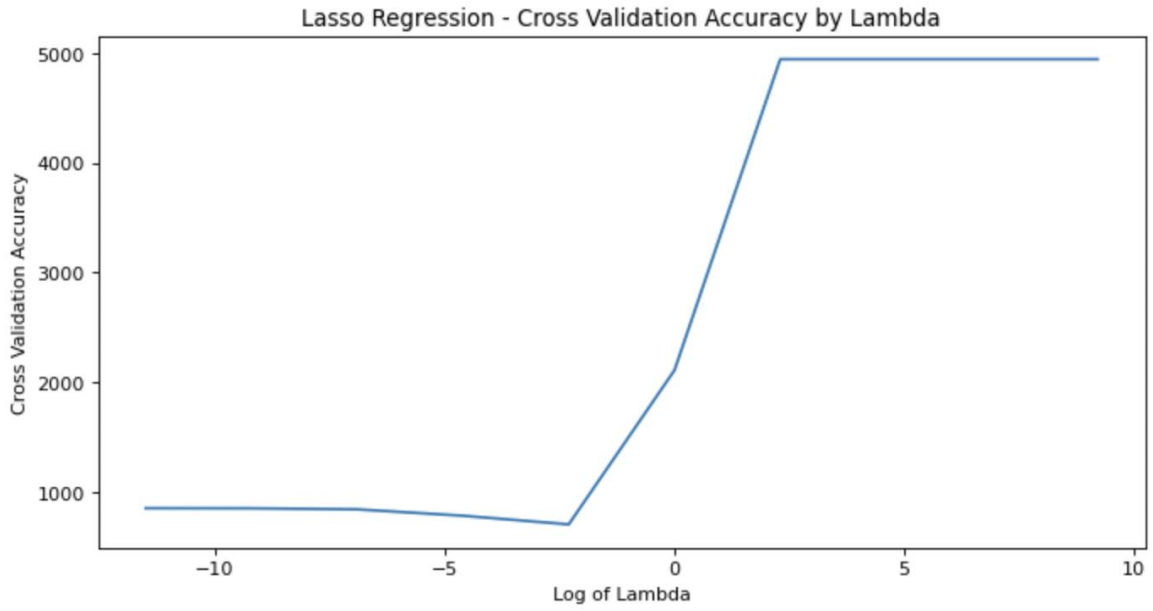
LASSO Programming

The LASSO, Least Absolute Shrinkage and Selection Operator, is a variation of the OLS regression model with regularization term that penalizes coefficients based on their absolute value. The objective function, to be minimized in LASSO, is as below -

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

To implement this process in our approach, we began by iterating through each validation set for different values of λ. In this case we have tried λ ranging from 10^{-5} to 10^{4}. Once validation set error is obtained then we pick the optimal value for λ that has the least error.

The resulting graph for log(λ) against cross-validation accuracy:



We observe a minimum cross-validation error when λ has a value of 0.1.

Now that we have a value for our hyperparameter λ, we assess the accuracy of the lasso model on our test set:

Test Accuracy

```
# fitting the model
lasso = Lasso(alpha=lambda_final)
lasso.fit(X_train[:, 1:], y_train)

# predicting and getting accuracy
y_pred = lasso.predict(X_test[:, 1:])
prediction_error = get_accuracy(y_test, y_pred)

# creating prediction dataframe
df_pred_lasso = pd.DataFrame({'y_actual': y_test, 'y_predicted': y_pred})

display(df_pred_lasso.head())

print(f'\033[1mPrediction error on test set: {prediction_error}\033[0m')
```

	y_actual	y_predicted
0	7.107949	6.012715
1	5.796272	4.909449
2	1.598651	3.149508
3	2.532953	3.524884
4	0.590685	-0.382678

For LASSO **we obtain a prediction error of 117.99**.

This is only slightly higher than the error of 116.83 that we obtained from our MIQP method of variable selection.

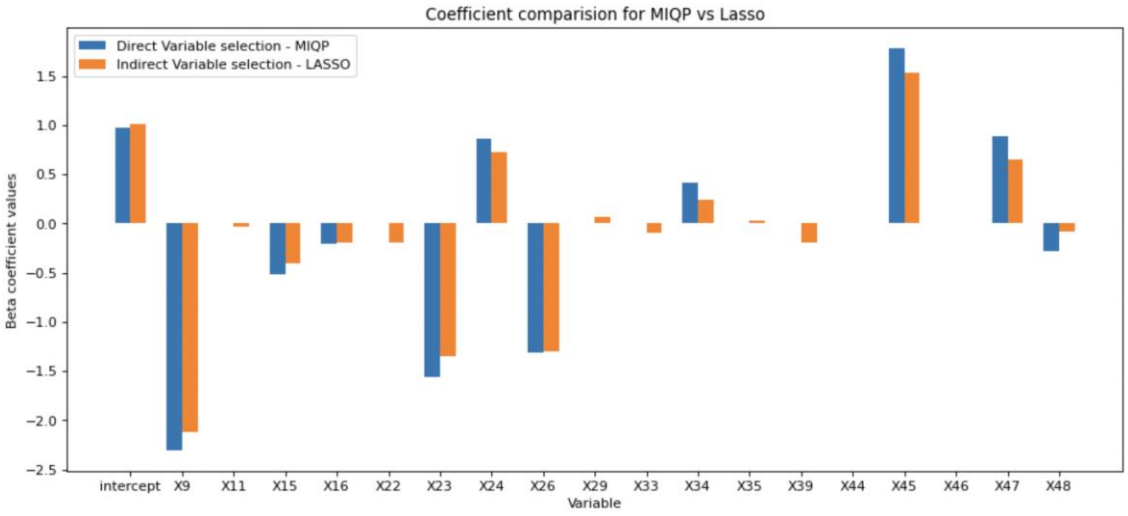
Comparison & Conclusion

To get a better grasp of how these methods differ, we compare their coefficients. To do this, we created a data frame composed of the LASSO and MIQP predictions as well as the actual observation values:

```
lasso_beta = np.insert(lasso.coef_, 0, lasso.intercept_, axis=0)
final_beta_df = pd.DataFrame(data={'variable': ['intercept'] + list(train.columns[1:]),
                                   'miqp_beta': miqp_beta, 'lasso_beta': lasso_beta})
df_pred = df_pred_miqp.merge(df_pred_lasso, on = 'y_actual', suffixes = ('_miqp', '_lasso'))
```

	y_actual	y_predicted_miqp	y_predicted_lasso
0	7.107949	6.179859	6.012715
1	5.796272	5.095243	4.909449
2	1.598651	3.285595	3.149508
3	2.532953	3.758485	3.524884
4	0.590685	-0.332975	-0.382678
5	-6.108818	-5.142737	-4.794720
6	-1.993021	-3.144544	-2.789680
7	-1.982904	-1.238063	-1.430238
8	0.134034	1.385111	1.421764
9	-0.728690	-0.441739	-0.279275
10	-1.766106	-1.695002	-1.961316

We use the coefficient values for the model with minimum error for both MIQP and LASSO to create a plot of all the non-zero coefficients in either LASSO or MIQP selection -



It is apparent from the graph that direct variable selection using mixed integer quadratic programming successfully removes more variables (by reducing their coefficients to 0) than the LASSO method does. The optimal model in **LASSO has 18 non-zero coefficients** for variables, while the **MIQP selection only has 10**. Even having a lesser number of coefficients in the linear regression equation, MIQP performs better than LASSO on the test dataset.

Both the models give very comparable results and performance, despite being very different regression equations. At the same time, we need to consider the runtime for both the methods. MIQP took about 2.5 hours to arrive at the best model though it exactly selected the number of variables that we wanted. This can really help in cases where we have a dataset with a very high number of variables and we want to keep only a handful of variables to make it more explainable to the business team. MIQP provides a lot of control compared to LASSO that the number of variables selected in the model will be very rigid. On the other hand, LASSO took less than 10 mins to execute for all the cases, making it way faster than MIQP. But the number of variables selected will always be a mystery in LASSO as you can't know or specify beforehand.

While LASSO has clear merits in its variance reduction capabilities, the advances in computational efficiency paired with the innovation of solvers mean that we no longer have to concern ourselves with the drawbacks that have previously been associated with direct variable selection. Therefore, we conclude that finding the 'best' variables to include in our regression directly is more effective than the indirect selection process that occurs via the LASSO shrinkage.