Project 3 – Nonlinear Programming

Group 16: Michael Bohnet – mrb4383, Yanqi Liang – yl29382, Matthew Tran – mct2345, Shengxiang Wu– sw38274

Overview

Our objective is to compare and contrast the Lasso method of variable selection to the direct variable selection implemented using a solver. In the following report, we will identify the advantages and disadvantages of both methods and come up with appropriate recommendations for variable selections in the future.

Mixed-Integer Quadratic Program Implementation

To formulate the variable selection problem into a MIQP problem, we posed some constraints and adjustments to the original ordinary least squares problem as follow:

$$
\begin{gathered}
\min_{\beta,z} \sum_{i=1}^{n} (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_m x_{im} - y_i)^2 \\
s.t. -Mz_j \le \beta_j \le Mz_j \quad for\ j = 1, 2, 3, \ldots, m \\
\sum_{j=1}^{m} z_j \le k \\
z_j \ \ are\ binary.
\end{gathered}
$$

The goal will be to minimize the sum of squared error using the dataset with m independent X variables and a dependent y variable. A binary variable, zj, is included to force βj to be zero if zj is zero, using the big M method where M is equal to 1000. The intercept will not be subject to the binary constraint. The variable k indicates the maximum number of variables to be included in the selection process.

Next, we reformulated the objective function as follow:

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

Here, β is the column vector containing m+1 β values. X is an n x (m+1) matrix where the first column is made up of 1s (for β_0) and the rest of the columns are β for the independent variables.

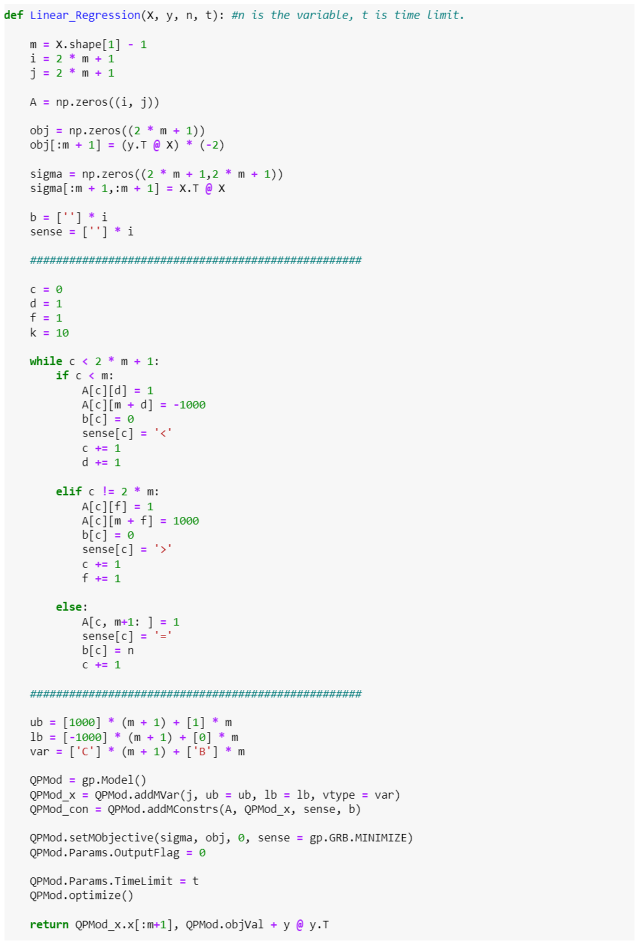
Y is the n x 1 column vector for the dependent variable. With these matrices and vectors specified, we can perform the following matrix calculations:

$$(X\,\beta - y)$$

This results in an n x 1 vector which contains n values of the errors shown in the first unformulated objective function. To take the sum of squared with this vector, we propose the following multiplication:

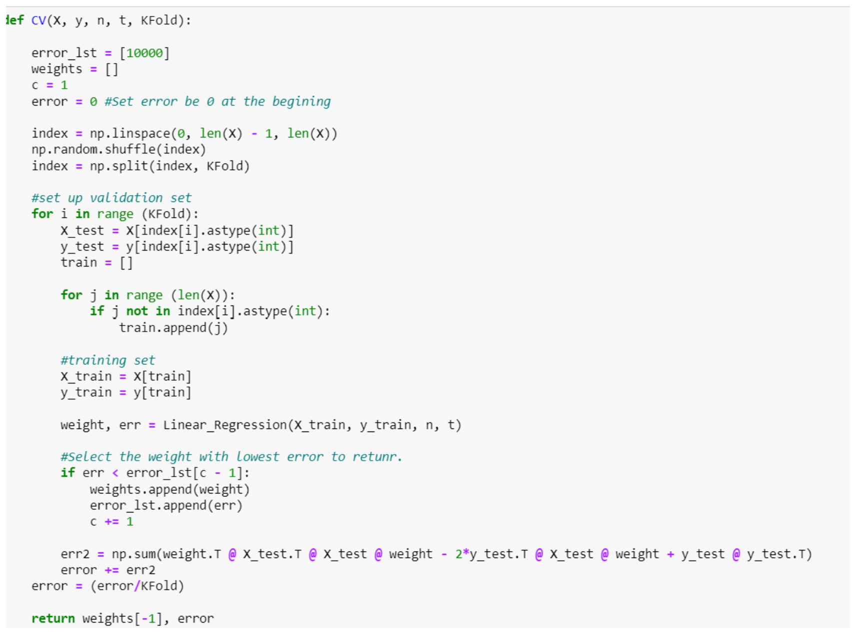
$$(X\,\beta - y)^T * (X\,\beta - y)$$

With some linear algebra, this will be posted as the above reformulated objective function. Going forward, we set up the objective function, constraint matrix, the right hand side vector and all relevant elements with the code below:



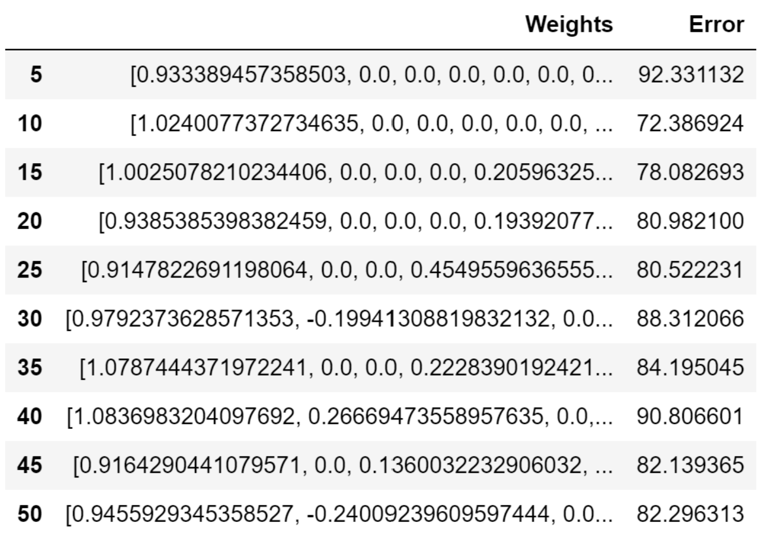
```python
def Linear_Regression(X, y, n, t): #n is the variable, t is time limit.

    m = X.shape[1] - 1
    i = 2 * m + 1
    j = 2 * m + 1

    A = np.zeros((i, j))

    obj = np.zeros((2 * m + 1))
    obj[:m + 1] = (y.T @ X) * (-2)

    sigma = np.zeros((2 * m + 1,2 * m + 1))
    sigma[:m + 1,:m + 1] = X.T @ X

    b = [''] * i
    sense = [''] * i

    ###################################################

    c = 0
    d = 1
    f = 1
    k = 10

    while c < 2 * m + 1:
        if c < m:
            A[c][d] = 1
            A[c][m + d] = -1000
            b[c] = 0
            sense[c] = '<'
            c += 1
            d += 1

        elif c != 2 * m:
            A[c][f] = 1
            A[c][m + f] = 1000
            b[c] = 0
            sense[c] = '>'
            c += 1
            f += 1

        else:
            A[c, m+1: ] = 1
            sense[c] = '='
            b[c] = n
            c += 1

    ###################################################

    ub = [1000] * (m + 1) + [1] * m
    lb = [-1000] * (m + 1) + [0] * m
    var = ['C'] * (m + 1) + ['B'] * m

    QPMod = gp.Model()
    QPMod_x = QPMod.addMVar(j, ub = ub, lb = lb, vtype = var)
    QPMod_con = QPMod.addMConstrs(A, QPMod_x, sense, b)

    QPMod.setMObjective(sigma, obj, 0, sense = gp.GRB.MINIMIZE)
    QPMod.Params.OutputFlag = 0

    QPMod.Params.TimeLimit = t
    QPMod.optimize()

    return QPMod_x.x[:m+1], QPMod.objVal + y @ y.T
```

Next, we created the 10 cross-validation functions by randomly splitting the training data set into training and validation sets using shuffle, and then we tested all the k values by running a linear regression function with different values of k in the cross-validation function, relevant code can be found below:



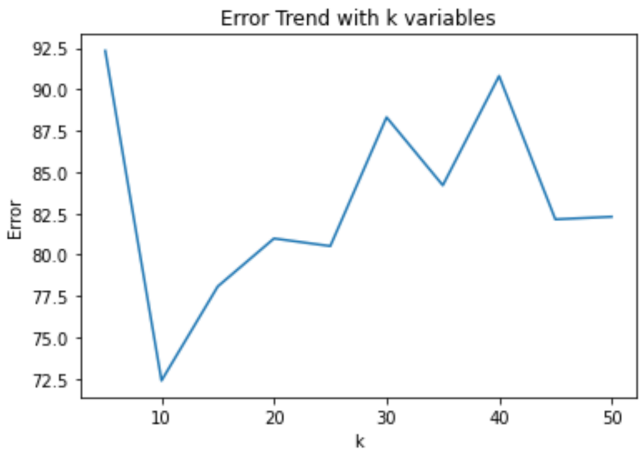
```
def CV(X, y, n, t, KFold):

    error_lst = [10000]
    weights = []
    c = 1
    error = 0 #Set error be 0 at the begining

    index = np.linspace(0, len(X) - 1, len(X))
    np.random.shuffle(index)
    index = np.split(index, KFold)

    #set up validation set
    for i in range (KFold):
        X_test = X[index[i].astype(int)]
        y_test = y[index[i].astype(int)]
        train = []

        for j in range (len(X)):
            if j not in index[i].astype(int):
                train.append(j)

        #training set
        X_train = X[train]
        y_train = y[train]

        weight, err = Linear_Regression(X_train, y_train, n, t)

        #Select the weight with lowest error to retunr.
        if err < error_lst[c - 1]:
            weights.append(weight)
            error_lst.append(err)
            c += 1

        err2 = np.sum(weight.T @ X_test.T @ X_test @ weight - 2*y_test.T @ X_test @ weight + y_test @ y_test.T)
        error += err2
    error = (error/KFold)

    return weights[-1], error
```

After hours of runtime, we are able to obtain the following results:

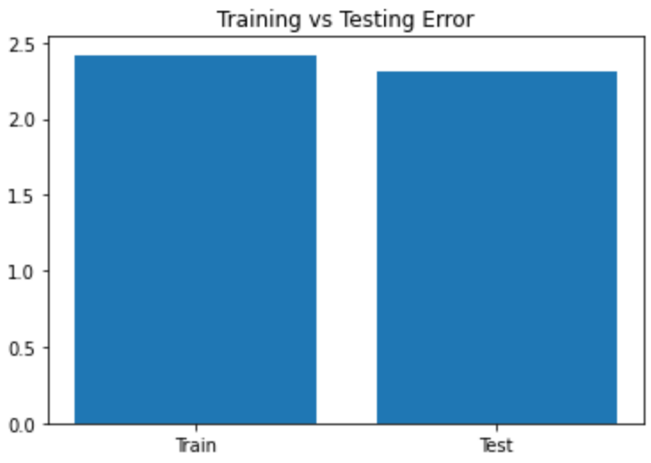


	Weights	Error
5	[0.933389457358503, 0.0, 0.0, 0.0, 0.0, 0.0, 0...	92.331132
10	[1.0240077372734635, 0.0, 0.0, 0.0, 0.0, 0.0, ...	72.386924
15	[1.0025078210234406, 0.0, 0.0, 0.0, 0.20596325...	78.082693
20	[0.9385385398382459, 0.0, 0.0, 0.0, 0.19392077...	80.982100
25	[0.9147822691198064, 0.0, 0.0, 0.4549559636555...	80.522231
30	[0.9792373628571353, -0.19941308819832132, 0.0...	88.312066
35	[1.0787444371972241, 0.0, 0.0, 0.2228390192421...	84.195045
40	[1.0836983204097692, 0.26669473558957635, 0.0,...	90.806601
45	[0.9164290441079571, 0.0, 0.1360032232906032, ...	82.139365
50	[0.9455929345358527, -0.24009239609597444, 0.0...	82.296313

To better examine the error trend among changing k variables, a plot is created below:



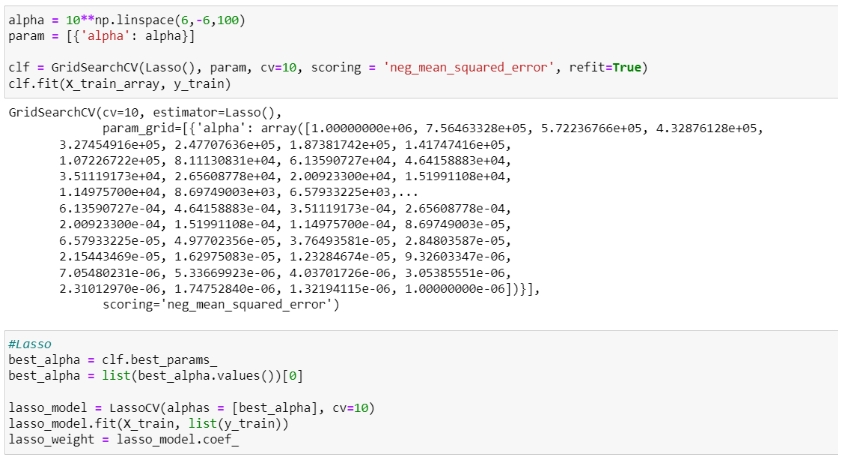
As we can see from the plots, error is the lowest when k = 10, therefore k = 10 is used when we fit the MIQP model on the entire training dataset. Next, we used the set of weights coming from the fitted model to predict y values for both the testing and training set, below is a plot summarizing our results:



With the MIQP model, we were able to achieve a training error of 2.42 and a better testing error of 2.31.

Lasso Implementation

After our own implementation of the variable selection process as an MIQP, we attempt to solve the same problem using the built-in Lasso and GridSearchCV function from sklearn. Relevant code can be found here:

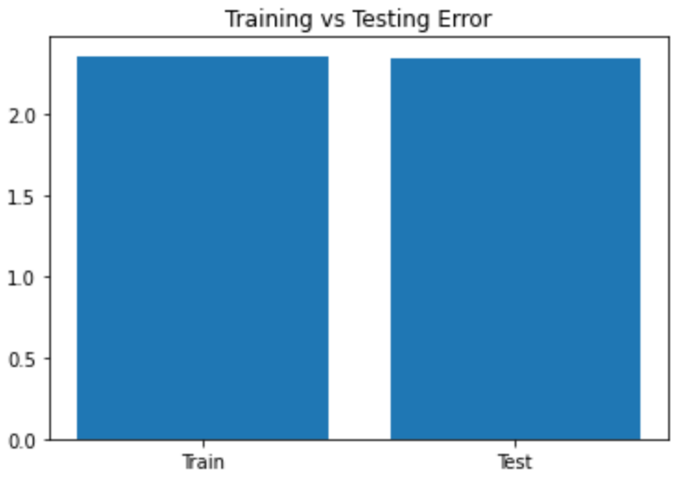


```
alpha = 10**np.linspace(6,-6,100)
param = [{'alpha': alpha}]

clf = GridSearchCV(Lasso(), param, cv=10, scoring = 'neg_mean_squared_error', refit=True)
clf.fit(X_train_array, y_train)
```

```
GridSearchCV(cv=10, estimator=Lasso(),
             param_grid=[{'alpha': array([1.00000000e+06, 7.56463328e+05, 5.72236766e+05, 4.32876128e+05,
       3.27454916e+05, 2.47707636e+05, 1.87381742e+05, 1.41747416e+05,
       1.07226722e+05, 8.11130831e+04, 6.13590727e+04, 4.64158883e+04,
       3.51119173e+04, 2.65608778e+04, 2.00923300e+04, 1.51991108e+04,
       1.14975700e+04, 8.69749003e+03, 6.57933225e+03,...
       6.13590727e-04, 4.64158883e-04, 3.51119173e-04, 2.65608778e-04,
       2.00923300e-04, 1.51991108e-04, 1.14975700e-04, 8.69749003e-05,
       6.57933225e-05, 4.97702356e-05, 3.76493581e-05, 2.84803587e-05,
       2.15443469e-05, 1.62975083e-05, 1.23284674e-05, 9.32603347e-06,
       7.05480231e-06, 5.33669923e-06, 4.03701726e-06, 3.05385551e-06,
       2.31012970e-06, 1.74752840e-06, 1.32194115e-06, 1.00000000e-06])}],
             scoring='neg_mean_squared_error')
```

```
#Lasso
best_alpha = clf.best_params_
best_alpha = list(best_alpha.values())[0]

lasso_model = LassoCV(alphas = [best_alpha], cv=10)
lasso_model.fit(X_train, list(y_train))
lasso_weight = lasso_model.coef_
```

Through the GridSearchCV function, we are able to obtain the best parameter (λ) and use it to fit a 10-fold cross validation on the entire training set. With the fitted model, we predicted y values for both the test and training set and calculated the training/testing mean squared error for a better comparison between the two approaches. Below is a comparison of the training and testing error using the Lasso function:



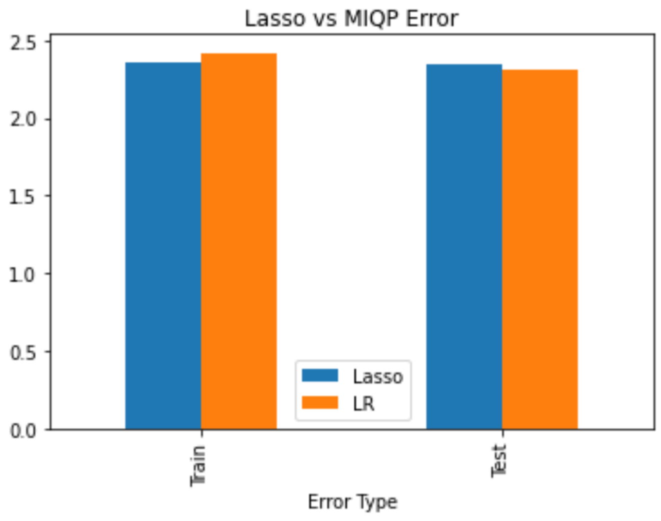
In this case, we have a training error of 2.36 and a slightly lower testing error of 2.35. In the next section, we will compare the results of both the models and offer insights/recommendations.

Insights and Recommendations

After running both models on the problem, we obtained the following training and testing errors:

	Train Error	Test Error
Lasso	2.356777	2.346139
LR	2.419120	2.309875

For a better comparison of the two models, the following plot is created:



As we can see from the table and the plot, Lasso performed better in terms of training error while MIQP performed better in testing error. In this case, it seems that the MIQP model will be a good transition for the company. However we also need to take into considerations such as computational power and time.

Both techniques operate by reducing the error based on a loss function. A major benefit of direct variable selection is that it removes unnecessary variables from the model entirely. This can increase the explainability of the model since the reduced variable number would allow for better interpretation. Additionally, it could also generate better performances especially with large datasets as the collinearity problem is not as prominent here due to removing some of the variables completely . The main drawback of using this technique more broadly is that it takes a lot of computational power and time. Lasso regression, on the other hand, is more time efficient and requires less computational power. It reduces the impact of suboptimal variables by reducing their weights (betas) to be closer to 0. While this minimizes the effect of these variables they can still affect model accuracy, which is a potential drawback.

Therefore, combining our insights and results, we believe that the firm should evaluate the choice between Lasso or direct variable selection based on the situation. If urgent results are needed or there are not enough computational resources, the firm should choose to use the Lasso method for quick and decent outputs. Otherwise, the firm should shift to a more direct variable selection process as it will yield a better outcome, especially with larger datasets.