Optimization Project 3

Group 2:

Tanupriya Mehra (TM33722) Junsu Kim (JK48353)

Anthony Huang (ASH3776) Prashanti Kodi (PK9759)

Problem Statement:

One of the most common problems in predictive analytics is variable selection for regression and direct variable selection using optimization has long been dismissed due to computational difficulties. In the recent past there have been tremendous advancements in optimization software, specifically the ability to solve mixed integer quadratic programs (MIQP). We will solve the variable selection problem for regression as an MIQP and compare the results found to LASSO to see if the additional 'shrinkage' component of LASSO really is more beneficial than finding the 'best' set of variables to include in our regression. We conduct these evaluations on a training data set and test data set. The report details our methodology and displays our findings and recommendations.

Direct Variable Selection - MIQP Overview:

The standard problem with independent variables X, and a dependent variable y, is formulated as:

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

To incorporate variable selection, we can include some binary variables, z_j, that force the corresponding values of β_j to be zero if z_j is zero. Using the big-M method, to include at most k variables from X, then we can pose this as:

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

With linear algebra, we can pose the optimization problem's objective function as:

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

Q1. Do a 10-fold cross validation on the training set to pick k or λ. With the optimal values of k or λ, fit your βs using the entire training set. With those βs, make a prediction of the y values on the test set, and compare your prediction of y, to the true value of y in the test set.

1. We created the X and y matrices for our training data set. We also created an objective matrix using the X matrix. A matrix XT_X was made by multiplying the X matrix with its transpose.

```
# Creating the X and y vector
y= np.array(df['y'])
X= np.array(df.drop(columns='y'))
X=np.c_[np.ones((len(df),1)),X]

X_T = X.transpose()
y_T= y.transpose()

Q = np.zeros([2*len(df.drop(columns='y').columns)+1,2*len(df.drop(columns='y').columns)+1])
Q.shape

XT_X = X_T @ X
XT_X.shape
```

```
(51, 51)
```

2. We inserted the matrix XT_X into our objective matrix to create the quadratic part of the objective. We also made the linear part of the objective by using the y transpose and X matrix.

```
# Inserting XT_X into top left corner of Q
for i in range(51):
    for j in range(51):
        Q[i,j]= XT_X[i,j]

m = len(df.drop(columns='y').columns)

linear_obj = -2*y_T @ X
obj = np.zeros((2*m+1))
obj[0: m+1] = linear_obj
```

3. The constraints are made. Since there are 50+50+1 constraints, Constraint matrix A will be 101x101.

Constraint 1 is z(1) + z(2) ... z(50) <= k

The next 50 constraints are β(j) + Mz(j) >= 0

The following 50 constraints are β(j) - Mz(j) <= 0

We define values for k and M.

4. We first initialized our constraint matrix and made a constraint for picking the number of betas. We then filled the matrix till the first 50 elements and created a constraint to set an upper limit on the betas. Next, we filled the matrix with the next 50 elements and created a constraint to set a lower limit on the betas. We finally created the b vector and specified the constraints such as the signs and the type of variable as all betas were continuous and z were binary. Lastly, we set a lower limit on the betas as gurobi has a default of 0.

```
A=np.zeros((2*m+1,2*m+1))
A[0,m+1:2*m+1] = [1]*m

# Filling A[1:50]
for i in range(1,m+1):
    A[i,i] = 1
    A[i, i+m] = M


# Filling A[51:100]
for i in range(m+1,2*m+1):
    A[i,i-m] = 1
    A[i, i] = -M

b= np.zeros(2*m+1)
b[0] =k

sense = ['<']+['>']*m + ['<']*m
vtype=['C']*(m+1) + ['B']*m
lb = -M
```

Q2. Write your own cross validation code for the MIQP model. Randomly shuffle your data and split it into 10 folds. Try k = 5, 10, 15, …, 50 in your cross validation and pick the value of k that corresponds to the smallest cross validation error.

1. We wrote a function to create 10 random folds from the dataframe for training data.

```
def cross_validation_func (cvfolds, df):
    for f in range(1,cvfolds+1):
        np.random.seed(0)
        indexes = np.random.choice(len(df),int(len(df)/cvfolds), replace= False)
    return indexes
```

2. We wrote a function for optimization that runs the k-fold cross validation and returns the SSE values for different values of k.

```
def optimization_func (df, k, cvfolds, M):
    mean_squared_error=[] # initialize list to store SSE

    for value in k:
        squared_error=[] # stores 10 error values for 10 folds at each k everytime time loop is run
        for x in range(cvfolds):
            indexes = cross_validation_func(cvfolds,df) # call cross validation function to create random indexes

            # create matrix for validation set
            validation_set = df.iloc[indexes]
            validation_set = validation_set.drop(columns='y', axis=1)
            validation_matrix = np.array(validation_set)
            validation_matrix= np.c_[np.ones((len(validation_set),1)),validation_set]

            validation_y = df['y'].iloc[indexes]
            validation_y = list(validation_y)

            # create matrix for training set
            training_set = df.drop(indexes, axis=0)
            training_set = training_set.drop(columns='y', axis=1)
            training_y = df['y'].drop(indexes, axis=0)
            training_y = list(training_y)


            m = len(training_set.columns)
```

```
# Following is the optimization section of the function
# Creating the X and y matrix
y= np.array(training_y)
X= np.array(training_set)
X= np.c_[np.ones((len(training_set),1)),training_set]

X_T = X.transpose()
y_T= y.transpose()

Q = np.zeros([(2*m+1),(2*m+1)])


XT_X = X_T @ X

# Inserting XT_X into top left corner of Q
for i in range(51):
    for j in range(51):
        Q[i,j]= XT_X[i,j] # This creates the quadratic part of the objective

# Writing the linear part of the objective
linear_obj = -2*y_T @ X
obj = np.zeros((2*m+1))
obj[0: m+1] = linear_obj
```

```
# Initialize the constraint matrix A
A=np.zeros((2*m+1,2*m+1))
A[0,m+1:2*m+1] = [1]*m  # Constraint for picking the number of betas

# Filling A[1:50]
for i in range(1,m+1):
    A[i,i] = 1
    A[i, i+m] = M  # Constraint for setting upper limit on betas


# Filling A[51:100]
for i in range(m+1,2*m+1):
    A[i,i-m] = 1
    A[i, i] = -M # Constraint for setting lower limit on betas

# Create the b vector
b= np.zeros(2*m+1)
b[0] = value

sense = ['<']+['>']*m + ['<']*m # specify the signs on the constraints
vtype=['C']*(m+1) + ['B']*m  # specify the type of variable -- all betas are continuous variables, all z are bina
lb = -M  # set lower limit on betas as gurobi automatically takes that as 0

# Set up the optimization model
olsMod = gp.Model()
olsMod_x = olsMod.addMVar(len(obj), vtype=vtype, lb=lb)
olsMod_con = olsMod.addMConstrs(A, olsMod_x, sense, b)
olsMod.setMObjective(Q,obj,0,sense=gp.GRB.MINIMIZE)

olsMod.Params.OutputFlag = 0
olsMod.Params.timelimit = time # time is specified at top of the notebook

olsMod.optimize()

betas= olsMod_x.x # store beta values in a list

# Take dot product of beta values with the x values in validation matrix to get predicted y
y_pred = np.dot(validation_matrix, betas[:51])

# Get sum of squared error
error_sq = sum((y_pred - validation_y)**2)
```

```
        # Get sum of squared error
        error_sq = sum((y_pred - validation_y)**2)

        squared_error.append(error_sq)
    mean_squared_error.append(sum(squared_error))
# return average SSE for each value of k
return mean_squared_error
```

3. We define for k values = 5,10,15,20,25,30,35,40,45,50 to run k-fold cross validation. We use these values in our optimization function.

```
k= list(range(5,51,5))
```

```
# call optimization function, specify df, k, cvfolds=10, M=200
returned_vals = optimization_func(df, k, 10, 200)
```

4. We create a dataframe to store SSE values for each value of k.

```
columns = ['k-value', 'SSE']
sse_df = pd.DataFrame(zip(k,returned_vals), columns=columns)
sse_df = sse_df.set_index('k-value')
```

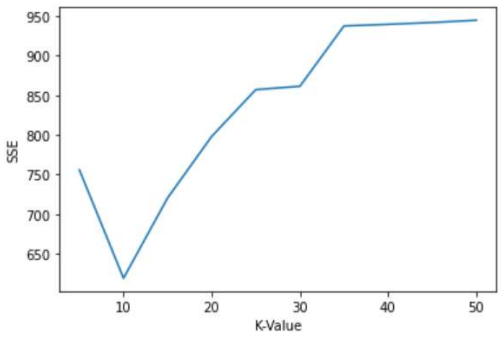
```
sse_df
```

k-value	SSE
5	755.692655
10	619.053973
15	720.317538
20	797.972161
25	856.995553
30	861.232576
35	937.382219
40	939.414678
45	941.679342
50	944.593865

5. We plot the SSE against the K value.

```
plt.plot(k, sse_df['SSE'])
plt.xlabel('K-Value')
plt.ylabel('SSE')
```

```
Text(0, 0.5, 'SSE')
```



From the graph, we see that k=10 has the lowest SSE. Therefore, the best value of k is 10.

6. We confirmed the best value of k as 10 by using idxmin().

```
best_k = sse_df['SSE'].idxmin()
```

```
best_k
```

```
10
```

Q3. Using the k with the smallest cross validation error, fit the MIQP model on the entire training set using that value of k. Use the βs you find in this MIQP to make a prediction of the y values in the test set.

1. We created the X and y matrices for our training data set and the objective matrix. We also made the linear part of the objective with the optimization problem's objective function.

```
# Creating the X and y matrix for train data
y= np.array(df['y'])
X= np.array(df.drop(columns='y'))
X=np.c_[np.ones((len(df),1)),X]

X_T = X.transpose()
y_T= y.transpose()


XT_X = X_T @ X

m = len(df.drop(columns='y').columns)


# Filling Q matrix with XT_X

Q = np.zeros([2*len(df.drop(columns='y').columns)+1,2*len(df.drop(columns='y').columns)+1])
Q[0:51,0:51]= XT_X


linear_obj = -2*y_T @ X
obj = np.zeros((2*m+1))
obj[0: m+1] = linear_obj
```

2. We initialized and filled out our constraint matrix. We also created the respective constraints.

```
A=np.zeros((2*m+1,2*m+1))
A[0,m+1:2*m+1] = [1]*m

# Filling A[1:50]
for i in range(1,m+1):
    A[i,i] = 1
    A[i, i+m] = M


# Filling A[51:100]
for i in range(m+1,2*m+1):
    A[i,i-m] = 1
    A[i, i] = -M

b= np.zeros(2*m+1)
b[0] = best_k ## INSERT HERE to try with another value of k

sense = ['<']+['>']*m + ['<']*m
vtype=['C']*(m+1) + ['B']*m
lb = -M
```

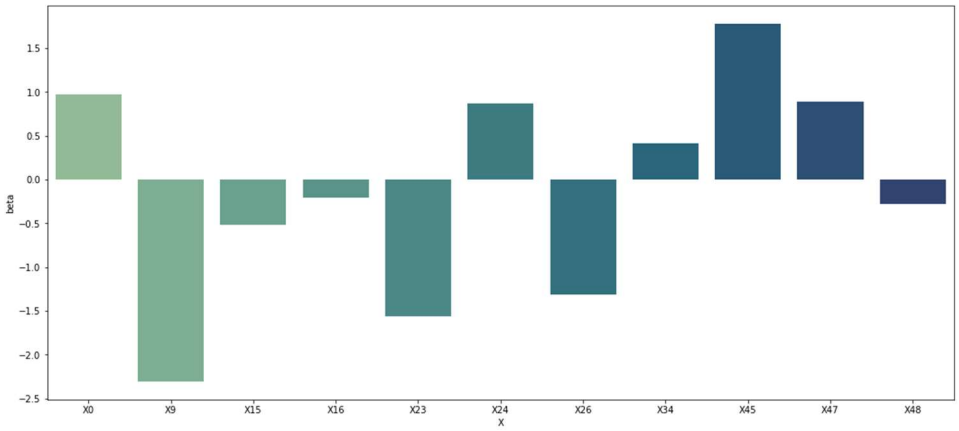
3. We solved for the optimal solution in our optimization model.

```
olsMod = gp.Model()
olsMod_x = olsMod.addMVar(len(obj), vtype=vtype, lb=lb)
olsMod_con = olsMod.addMConstrs(A, olsMod_x, sense, b)
olsMod.setMObjective(Q,obj,0,sense=gp.GRB.MINIMIZE)

olsMod.Params.OutputFlag = 0

olsMod.optimize()
```

The following graph shows the variables selected and their betas.



4. We used the betas from our solution to predict the y values in the test data.

```
betas_training = olsMod_x.x
```

```
# Use betas to predict y values on the test data
y_pred_test = np.dot(X_test_matrix, betas_training[:51])
y_pred_test
```

```
array([ 6.17985878,  5.09524299,  3.28559532,  3.75848539, -0.33297526,
       -5.14273683, -3.14454357, -1.23806288,  1.38511093, -0.44173854,
       -1.69500225,  2.73035027,  0.74744903, -0.97192232, -0.68681528,
        8.04522381, -7.94698471,  3.89063974, -4.58142919, -3.21992082,
       -2.16211454,  3.21686318, -3.19810533,  0.19740731, -2.35988844,
       -0.41999885, -1.9125216 , -3.32418587, -3.14170972, -3.55379324,
       -1.80842543, -0.37134301,  1.8670808 ,  5.04927886, -1.80005614,
        3.09427675,  4.38154309,  2.6988627 ,  1.6132886 ,  5.97584637,
       -1.1973583 ,  5.2232542 , -5.84899891, -1.14461528,  4.51802998,
        4.18774866,  4.12046008,  0.61483809,  1.95723246, -1.54904383])
```

5. We calculated for the MSE and SSE between the predicted and actual y matrix.

```
error_sq = sum((y_pred_test - list(y_test_matrix))**2)
cv_mse = mean_squared_error(y_pred_test, list(y_test_matrix))

print('The SSE on the test data is: ', round(error_sq,3))
print('The MSE on the test data is: ', round(cv_mse,3))
```

```
The SSE on the test data is:  116.827
The MSE on the test data is:  2.337
```

Q4. Do a 10-fold cross validation on the training set to pick λ. With the best value of λ, fit a LASSO model to the entire training set using that value of λ. With the βs you find in that LASSO model make a prediction of the y values in the test set.

The formula for LASSO regression is given by:

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

Where λ is a hyperparameter to be chosen using cross-validation.

1. We found out the amount of penalty with our 10-fold cross validation.

```
reg = LassoCV(cv=10, random_state=0).fit(X,y)
```

```
# keep the best hyper-parameter for the penalty, alpha
alpha = reg.alpha_
print('The amount of penalty from CV is: ', round(alpha,5) )
```

```
The amount of penalty from CV is:  0.07639
```

2. We used the best value of alpha that was previously identified into a Lasso model.

```
clf = linear_model.Lasso(alpha=alpha)
clf.fit(X,y)
```

```
Lasso(alpha=0.07638765995113514)
```

3. We used the lasso model to predict the y values in the test data. We then computed for the SSE and MSE.

```
# Predict using lasso model on the test set
y_pred_lasso = clf.predict(X_test_matrix)
```

```
# Compute the SSE and MSE
lasso_sse = sum((y_pred_lasso - list(y_test_matrix) )**2)
MSE = mean_squared_error(y_pred_lasso, list(y_test_matrix))

print('The SSE on test data using lasso is: ', round(lasso_sse,3))
print('The MSE on test data using lasso is: ', round(MSE,3))
```

```
The SSE on test data using lasso is:  117.482
The MSE on test data using lasso is:  2.35
```

4. We compared the LASSO CV with our cross validation function.

```
errors = [error_sq, lasso_sse]

index= ['CV model', 'LASSO model']
cols=['SSE']
df2= pd.DataFrame(errors, index=index, columns= cols)
df2
```

	SSE
CV model	116.827198
LASSO model	117.481738

Looking at the errors in the table above, we can say that the CV model we wrote performed better than the LASSO model.

Q5. Pretend that you are a junior consultant at an analytics consulting firm. You frequently use LASSO in your job, but your boss has heard that the computational time of direct variable selection has decreased with the advent of better solvers. Your boss wants you to figure out if the firm should shift away from using LASSO to incorporate more direct variable selection. Write this project as if this is what you're going to deliver to your boss. Describe the advantages and disadvantages of both techniques.

Lasso has gained its popularity due to its ability to induce sparsity as wells as its computational efficiency compared to those of more traditional methods such as subset selection and Ridge regression. However, Lasso has difficulty when directly controls and fine-tunes the extent of sparsity, which can be done using the direct variable selection methods like MIP, MIQP. Although using MIQP allows us to directly handle the number of non-zero features, it's, in general, costs more computational resources and not appropriate to be used in the situation where we have hundreds or thousands of variables to be considered. Along with these facts, many factors should be considered in determining which method performs better – the linear association between features (collinearity), true underlaying sparsity, and/or signal strengths. In summary, it's hard to tell to which method we need to shift without considering the complexity of the given problem. However, we can say that if our goal directs toward suggesting solution in timely and agile manner and this brings more business value for our customers, we should choose l1 constraint-based methods. If the given problem contains relatively small number of independent variables and requires sophisticated selection of sparsity, we should choose more direct methods, MIQP.

*Cite: MIP-BOOST: Efficient and Effective L0 Feature Selection for Linear Regression