Project 3 - Non-Linear Programming

Kolton Fowler kjf937, Jacob Pammer jwp2843, Sam Laplatney sjl2953, Yashwini Kalva yk8348

Problem Statement

The following pages outline an experiment conducted to assess the efficacy of using MIQP for regression variable selection. Using an example dataset, regression models were fit to training data using k-fold cross validation for both MIQP and basic Lasso regression. Each model was then evaluated with test data and the out of sample accuracy results were compared. Based on our out-of-sample accuracies of each method, we have determined that MIQP can be a viable option for variable selection for datasets similar to that which we have used to conduct this experiment.

Dataset Description

The dataset used consists of 300 observations with 50 independent variables and 1 dependent variable which we will be attempting to predict. The independent variables are numeric values ranging from -4.18 to 3.93. The independent variable is a numeric value ranging from -13.63 to 13.55. The dataset was split into training and test sets of size 250 and 50, respectively. The image below shows descriptive statistics of the first 10 independent variables in the training set (the remaining 40 independent variables have similar stats).

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
count	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000
mean	-0.087282	0.026473	-0.163096	0.041009	0.020960	-0.001726	0.044174	-0.007413	-0.041655	0.055776
std	1.083987	0.908522	1.050009	1.067472	0.964491	0.990949	0.945099	1.073749	1.280636	0.958147
min	-3.392061	-2.700156	-2.978052	-3.951204	-2.670434	-2.467016	-2.092597	-3.709112	-3.134870	-2.370430
25%	-0.849062	-0.572142	-0.810256	-0.647595	-0.667149	-0.631013	-0.634113	-0.728466	-0.941972	-0.660088
50%	-0.048234	0.074477	-0.150321	0.029772	0.068520	0.000236	0.107919	-0.021933	-0.141181	0.061222
75%	0.703440	0.618942	0.531564	0.825568	0.716679	0.696246	0.736878	0.823935	0.895871	0.694496
max	2.428520	2.503543	2.891464	2.941689	2.495949	2.586696	2.515698	3.000847	3.754544	2.554471

The image below shows descriptive statistics for the target variable in the training set.

```
count    250.000000
mean       1.276232
std        4.443005
min      -13.629531
25%       -2.015682
50%        1.478406
75%        4.618874
max       13.548518
Name: y, dtype: float64
```

MIQP Model

Quadratic programs take the following form:

$$x^T Q x + c^T x$$

s.t.

$$Ax \leq b$$
$$x \geq 0$$

For the purposes of variable selection, a quadratic program will take the form:

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

In order to execute variable selection using quadratic programming, the data must be manipulated into the form shown above. Our decision variables in the program will be the 51 beta values (50 for each independent variable coefficient and one for the intercept) and 50 binary variables (z) that determine whether or not each independent variable is selected for the model. The following steps were taken to put the data in the correct form:

1. Pandas dataframe of training data was converted to numpy array
2. Column of value 1 for every observation added to represent value for B0
3. Dependent variable data converted to a column vector
4. Create Q Matrix
    a. Q is a 101X101 matrix where the top left quadrant is equal to the data matrix transposed dot product with itself; this form allows gurobi to assign values to the z variables
5. Create objective vector of length 101 with for 51 beta parameters and 50 binary variables (z)
6. Create sense vector of length 101 (50 '>' for the zs and 51 '<' for the betas)

The code to formulate the problem as an MIQP is shown below. This function initializes our matrices/vectors described in steps that are to be used in Gurobi. After the matrices are complete, we set up our gurobi model to minimize the variables selected and also return AVG MSE when applying our test set.

```
# Define a function to perform k-fold cross validation. In this case we are doing a 10-fold cross validation.
# Since there are 50 Independent variables, k = 5,10,15,...45,50.
# In this case Mixed Integer Quadratic programming has to run for 100 times.
# We need to check which K has the lowest validation error

def Mixed_Int_Quad_Program(k,train_x,test_x, train_y,test_y):
    #print("K-Value: ", k)
    M=20
    C=0
    m=train_x.shape[1]-1
    n=train_x.shape[0]

    X=train_x.to_numpy()

    #Objective Functions

    Q=np.zeros((2*m+1,2*m+1)) #Initializing Objective function with 0's
    Q[:m+1,:m+1] = X.T @ X    # Defining objective function

    c=np.zeros((2*m+1))
    c[:m+1]=-2*(train_y.to_numpy()).T@X

    # Initializing constant matrix

    equation_count = (2 * m) + 1
    decision_variable_count = (2 * m) + 1
    A=np.zeros((equation_count,decision_variable_count))
    b=np.zeros((equation_count,1))

    # Preparing constant matrices
    # z_i are binary variables used to decide if the feature is important or not

    # 1 * beta_i - M * z_i < 0
    for p in range(m):
        A[p , 1 + p] = 1
        A[p , 1 + p + m] = M

    # -1 * beta_i - M * z_i < 0
    for p in range(m):
        A[p + m , 1 + p] = 1
        A[p + m , 1 + p + m] = -M

    for p in range(m + 1 ,2 * m + 1):
        A[2 * m , p] = 1
    b[2 * m] = k
```

```
# Preparing the sense matrix
sense = ['>'] * m + ['<'] * (m + 1)
vtype = ['C'] * (m + 1) + ['B'] * m

# As gurobi assumes all decision variables as as positive
# To allow coeffiients to take negative values, set lb value to -M
lb = [-M] * (m + 1) + [0] * m


# Gurobi Model
model = gp.Model()
modelX = model.addMVar(2 * m + 1, lb=lb, ub=None, vtype=vtype)
modelConstr = model.addMConstr(A, modelX, sense, b)
model.setMObjective(Q,c,C, sense = gp.GRB.MINIMIZE)
model.Params.OutputFlag = 0
# setting a timer for 1 hour
model.Params.timeLimit = 3600
model.optimize()

coefficients = modelX.x[:m+1]

for b in coefficients:
    if b == M or b == -M:
        print("Coefficients are Bounded")

return mean_squared_error(test_y, test_x @ modelX.x[:m+1])
```

To evaluate models with different numbers of parameters, K-fold Cross validation with K=10 was executed for k = 5, 10, 15, 20, 25, 30, 35, 40, 45, and 50 variables. We then took the average of the MSEs for each value of k and compared the results to determine our best MIQP variable selection model. The overall results, best MSE, and optimal number of variables is shown below.

MIQP Results

- Optimal number of variables: 10
- Average MSE: 2.3365439645525248

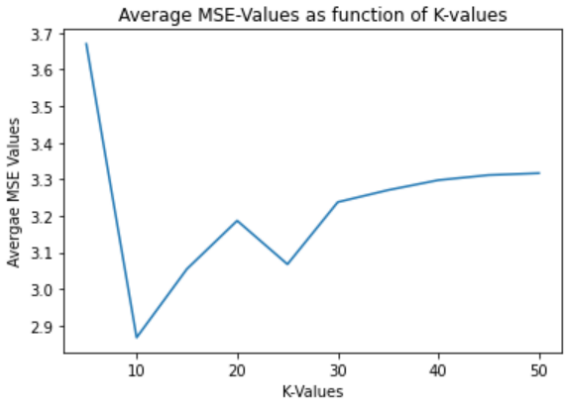
```
print("MSE Values for each K value across 10 folds : ",mse_df)
```

```
MSE Values for each K value across 10 folds :            1         2         3
4         5         6         7   \
5   3.989653  3.668456  3.628249  3.016326  4.137824  2.950241  4.005491
10  3.087377  2.691431  2.598744  3.104283  3.654779  2.722702  3.085196
15  3.051939  2.586483  2.492001  3.239033  3.799988  3.157608  3.628723
20  3.069909  2.612931  2.940455  3.249115  3.825879    3.4352   3.85728
25  2.700598  2.542442  2.663355  3.168037  3.815658  3.746373  3.664893
30  2.853712  2.933952  3.189938  3.187329  3.801332  3.597326  4.006988
35  3.230019  3.045027   3.13641   3.39008   3.46865  3.617229  4.021677
40  3.278053   2.95303  3.157792   3.43824  3.583962  3.593547   4.10417
45  3.221199  3.006661    3.2932  3.385424  3.570978  3.611326  4.152091
50  3.167247  3.000676  3.314224  3.365449  3.571293  3.609335  4.220439

          8         9         10
5   3.002222  5.135943  3.162297
10  2.764482   3.13973  1.825048
15  2.997251  3.673036  1.921194
20  2.794895  3.676022  2.406255
25  2.867033  3.555527  1.953304
30  2.974393  3.538591  2.291469
35  3.047205  3.375591  2.371587
40  3.063719  3.420736  2.382521
45  3.108155  3.430237  2.334278
50  3.142765  3.440257  2.333992
```

```
mean_mse = mse_df.mean(axis=1)
print("Best K with minimum MSE is : ",mean_mse[mean_mse == mean_mse.min()])
```

```
Best K with minimum MSE is :  10    2.867377
dtype: float64
```



Lasso Regression

Simple lasso regression was then executed on the same data in scikit learn, using the same cross validation method described above. The lasso objective function takes the form:

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

The code for lasso regression and the corresponding results are shown below.

Lasso Results

- Optimal number of non-zero variables: 18
- MSE on testing data: 2.3467369853530413
- The MIPQ solution seems to select less variables and still returns a better MSE.

Lasso Regression

```
cross_val = KFold(n_splits=10, shuffle=True, random_state=10)

lasso_regression = LassoCV(cv = cross_val).fit(X_train,y_train)
print('Non-zero Coefficients of Lasso Regression ',   sum(lasso_regression.coef_ != 0))

y_pred = lasso_regression.predict(X_train)
print("MSE on Train Data is :", mean_squared_error(y_pred,y_train))

y_pred = lasso_regression.predict(X_test)
print("MSE on Test Data is :", mean_squared_error(y_pred,y_test))
```

```
Non-zero Coefficients of Lasso Regression  18
MSE on Train Data is : 2.3583500089229474
MSE on Test Data is : 2.3467369853530413
```

Observations/ Recommendations

Based on the results of the experiment, MIQP appears to be a better option for variable selection than lasso regression in this particular circumstance based on the lower test MSE (2.35 for Lasso and 2.33 for MIQP) and the smaller number of variables selected (18 for Lasso, 10 for MIQP). Therefore, the MIQP model was more accurate and less complex. Based on these results alone, I cannot recommend to exclusively use MIQP in future projects involving variable selection as the efficacy of each method largely depends on the dataset being used. However, moving forward we should always consider MIQP alongside lasso regression when conducting projects that involve a large number of independent variables.