RM 294 – Optimization I
Project 3 – Variable Selection Project
Group 24

Oluwaseun Ibitoye (oli63), Vivek Mehendiratta (vm24395), Karthick Ramasubramanian (kr33733), Bhavana Reddy (bc35833)

Requirements

The purpose of this report is to discuss two different methods of variable selection for predictive analytics and determine which one would be best suited for the firms' purposes. The methods to be considered are LASSO regression and direct variable selection using Mixed-Integer Quadratic Programs.

Variable Selection Methods

Direct Variable Selection Method

For the Direct Variable Selection Method, we start by looking at a dataset of m independent variable, X and a dependent variable Y. The least squares problem is formulated as below:

Equation 1: $$\min_{\beta} \sum_{i=1}^{n} (\beta_o + \beta_1 x_{i1} + \ldots\ldots\ldots + \beta\ \ x_i\ \ - y_i)^2$$

In order to incorporate variable selection into this problem we include some binary variables, z , that force the corresponding values of β to be zero if z is zero using the big M method, including at most k variables from X as shown below:

Equation 2: $$\min_{\beta,z} \sum_{i=1}^{n} (\beta_o + \beta_1 x_{i1} + \ldots\ldots\ldots + \beta\ \ x_i\ \ - y_i)^2$$

$$\text{s.t } -Mz \quad \le \beta \quad \le Mz \quad \text{for j} = 1,2,3\ldots\ldots \text{M}$$

$$\sum_{j=1}^{m} z \quad \le k,\ z \quad \text{are binary}$$

In order to use this as a quadratic programming objective, it can be rewritten using linear algebra:

Equation 3: $$\min_{\beta,z} \beta^T (X^T X)\beta + (-2y^T X)\beta$$

Where, βis an (m+1)*1 column vector that contains β_o, β_1……. β , X is an n*(m+1) matrix that has first column made up of only 1s and columns 2 to (m+1) made up of the independent variables and y is an n*1 column vector.

LASSO Regularization Method

The indirect variable selection or LASSO Regularization method is posed as

$$\min_{\beta} \sum_{i=1}^{n} (\beta_o + \beta_1 x_{i1} + \ldots\ldots\ldots + \beta\ \ x_i\ \ - y_i)^2 + \lambda \sum_{j=1}^{m} |\beta\ \ |$$

Where λ is a hyperparameter that can be chosen during cross validation. The LASSO regularization model has the advantage of 'shrinking' the β*s* closer to zero, leading to variance reduction. Also, if λ is large enough, several values of β will be forced to be equal to 0.

Results & Performance

In order to solve the MIQP problem, we first started off by defining the cross-validation function to split the training data into 10 folds (***Appendix: Figure 1***). The function was set to try values of **k** ranging from 5 to 50 since there were 50 variables to select from. We then set up the optimization problem through Gurobi (***Appendix: Figure 2***) in the format described in the Direct Variable Selection section of this report. By minimizing Equation 3, and using the cross-validation method at different values of k we found the best value of **k** to be ***10***. This means that a model with 10 variables produced the lowest cross-validation error. With this information, the MIQP model was fit on the entire training set using k=10 to determine the β's for each feature and then used to predict the y values in the test set.

To assess the performance of this method, we calculated the test MSE (Mean Squared Error) and found it to be 2.34 (***Appendix: Figure 3***).

After solving the variable selection through MIQP, we used the normal Scikit-Learn package to perform a LASSO Regression (***Appendix: Figure 4***). We first performed 10-fold cross-validation to pick the optimal value of **λ** which was found to be 0.076 and then used it to fit the LASSO model to the entire training set. Once the β's for each feature were found, we used them to predict the y values in the test set. The test MSE for this model was 3.59 and the number of features selected was 17.

A comparison chart was created to show which variables were chosen by each selection method ***Appendix: Figure 5***). For reference, both the direct variable selection method and LASSO choose variables X9, X15, X16, X23, X24, X26, X34, X45, X47, X48. In addition to those, the LASSO model also chose a further 7 variables which are X11, X22, X29, X33, X39, X44, and X46.

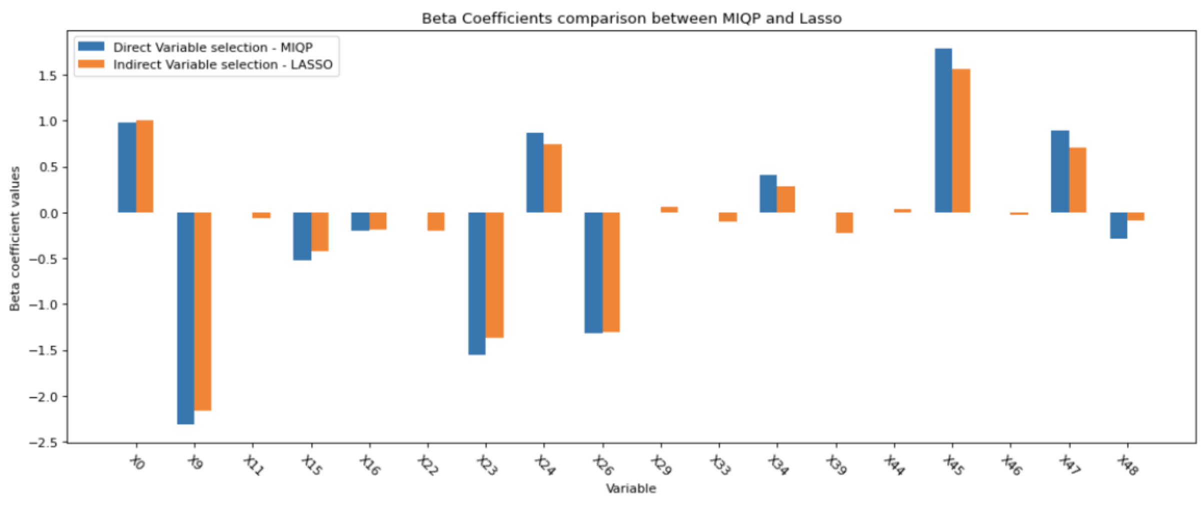


Figure 5: Beta coefficient comparison between MIQP & LASSO.

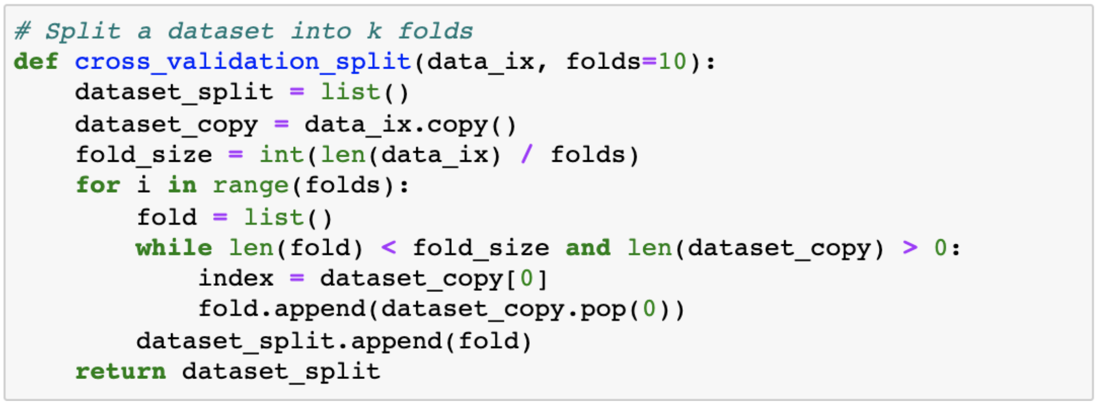
Recommendations

Following our assessments, we recommend that the firm does not completely switch from using LASSO to direct variable selection. Depending on the use case, there are pro's and con's to each method. It is important to keep this in mind when determining which one to choose. With this in mind, the firm can use either method as the business case requires.

For the direct variable selection method (MIQP model), the computational time of finding the best value of **k** is still significantly longer than the LASSO model. In situations where the number of features to be selected from are higher, the direct variable selection model can pose difficulty. However, this method has better performance which is very important in predictive analytics.

Contrastingly, LASSO regression is an in-built Python feature and can be easily accessed and used with minimal lines of code, unlike the MIQP model. The results of the LASSO model are also easier to interpret for clients and stakeholders who may not understand standard optimization techniques.

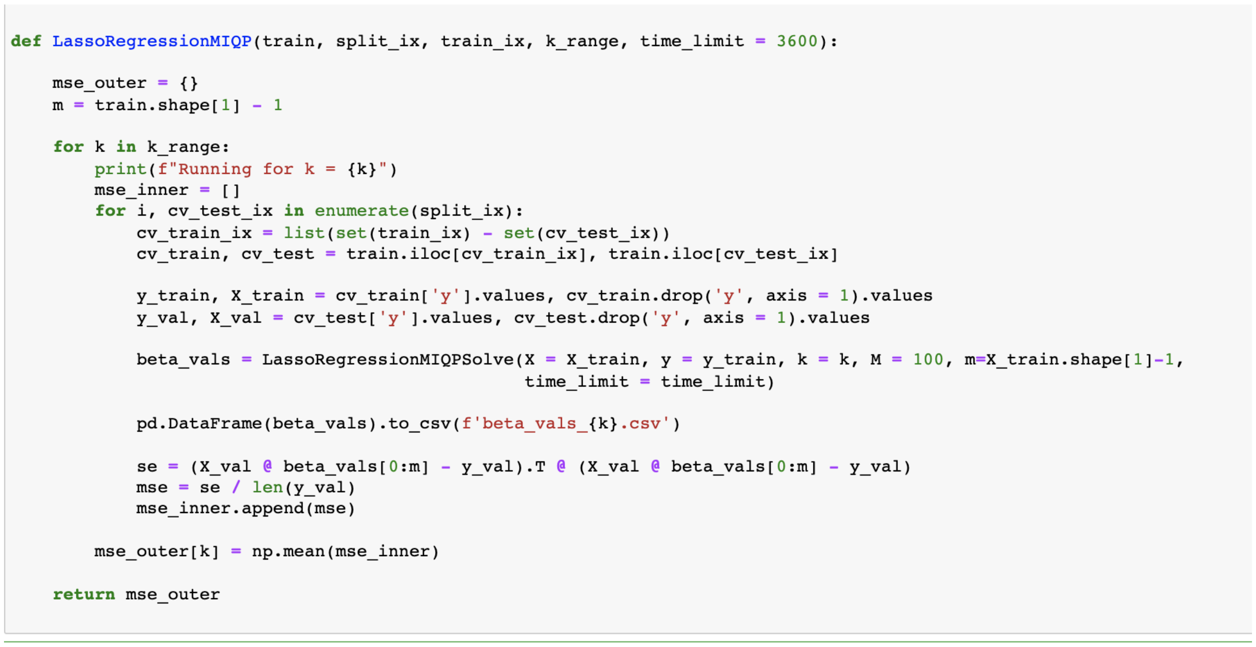
Appendix



```
# Split a dataset into k folds
def cross_validation_split(data_ix, folds=10):
    dataset_split = list()
    dataset_copy = data_ix.copy()
    fold_size = int(len(data_ix) / folds)
    for i in range(folds):
        fold = list()
        while len(fold) < fold_size and len(dataset_copy) > 0:
            index = dataset_copy[0]
            fold.append(dataset_copy.pop(0))
        dataset_split.append(fold)
    return dataset_split
```

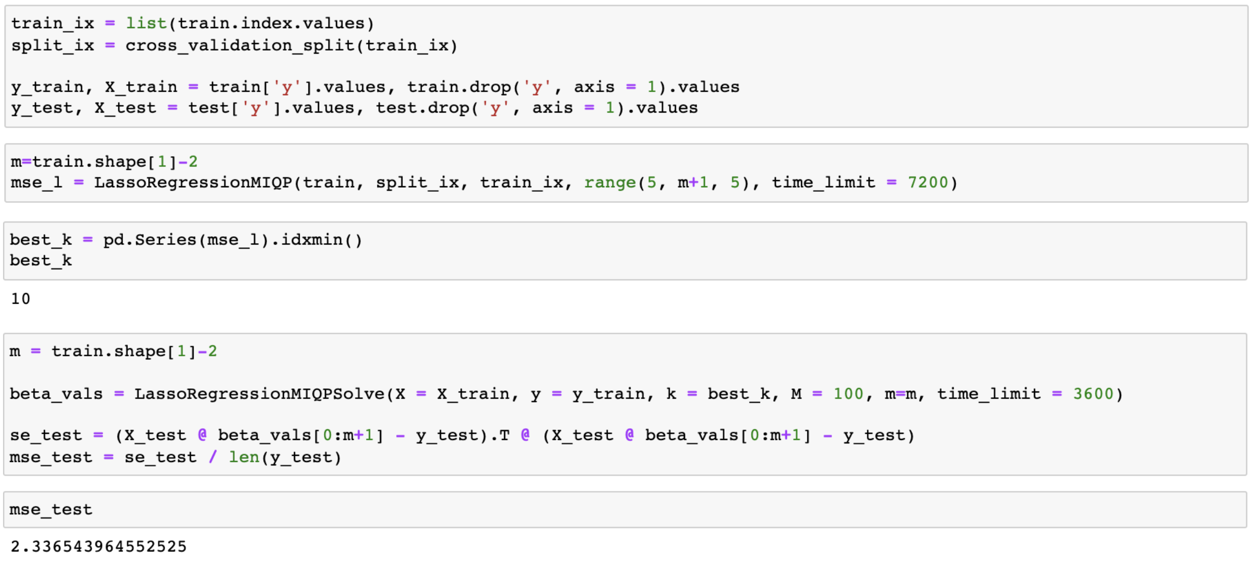
Figure 1: Setup of Cross-validation for MIQP problem.

```python
def LassoRegressionMIQPSolve(X, y, k, M, m, time_limit):

    ## Objective Definition

    # Quadratic part
    q = 2*m+1
    Q = np.zeros((q,q))
    Q[0:m+1, 0:m+1] = X.T @ X


    # Linear Part
    L = np.array(list(-2 * y.T @ X) + [0]*m)

    ## Constraint Definition
    A = np.zeros((q,q))
    sense = np.array(['']*q)
    b = np.array([0]*q)

    row = 0
    # -Mz_j <= b_j
    A[row:row+m, 1:m+1] = np.identity(m)
    A[row:row+m, m+1:q] = M*np.identity(m)
    sense[row:row+m] = ['>']*m
    b[row:row+m] = [0]*m

    row+=m
    # b_j <= Mz_j
    A[row:row+m, 1:m+1] = np.identity(m)
    A[row:row+m, m+1:q] = -1*M*np.identity(m)
    sense[row:row+m] = ['<']*m
    b[row:row+m] = [0]*m

    row+=m
    # sum(z_j) = k
    A[-1, m+1:q] = [1]*m
    sense[-1] = '<'
    b[-1] = k


    beta_vals = solveQuadratic(m_var=q, A=A, sense=sense, b=b, lb=np.array([-M]*q),
                               ub=np.array([M]*q), min_max='minimize', vtype = ['C']*(m+1)
                               + ['B']*m, time_limit=time_limit, Q=Q, L=L, C=0)

    return beta_vals
```



```python
def LassoRegressionMIQP(train, split_ix, train_ix, k_range, time_limit = 3600):

    mse_outer = {}
    m = train.shape[1] - 1

    for k in k_range:
        print(f"Running for k = {k}")
        mse_inner = []
        for i, cv_test_ix in enumerate(split_ix):
            cv_train_ix = list(set(train_ix) - set(cv_test_ix))
            cv_train, cv_test = train.iloc[cv_train_ix], train.iloc[cv_test_ix]

            y_train, X_train = cv_train['y'].values, cv_train.drop('y', axis = 1).values
            y_val, X_val = cv_test['y'].values, cv_test.drop('y', axis = 1).values

            beta_vals = LassoRegressionMIQPSolve(X = X_train, y = y_train, k = k, M = 100, m=X_train.shape[1]-1,
                                                 time_limit = time_limit)

            pd.DataFrame(beta_vals).to_csv(f'beta_vals_{k}.csv')

            se = (X_val @ beta_vals[0:m] - y_val).T @ (X_val @ beta_vals[0:m] - y_val)
            mse = se / len(y_val)
            mse_inner.append(mse)

        mse_outer[k] = np.mean(mse_inner)

    return mse_outer
```

Figure 2: Setup of equations for MIQP problem in Gurobi.



```python
train_ix = list(train.index.values)
split_ix = cross_validation_split(train_ix)

y_train, X_train = train['y'].values, train.drop('y', axis = 1).values
y_test, X_test = test['y'].values, test.drop('y', axis = 1).values
```

```python
m=train.shape[1]-2
mse_l = LassoRegressionMIQP(train, split_ix, train_ix, range(5, m+1, 5), time_limit = 7200)
```

```python
best_k = pd.Series(mse_l).idxmin()
best_k
```

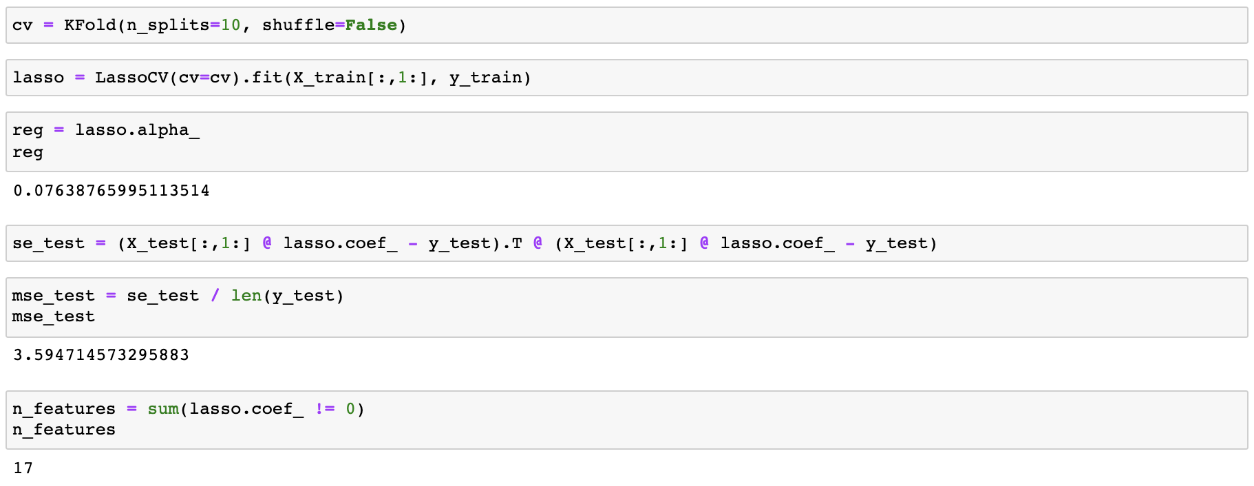
```
10
```

```python
m = train.shape[1]-2

beta_vals = LassoRegressionMIQPSolve(X = X_train, y = y_train, k = best_k, M = 100, m=m, time_limit = 3600)

se_test = (X_test @ beta_vals[0:m+1] - y_test).T @ (X_test @ beta_vals[0:m+1] - y_test)
mse_test = se_test / len(y_test)
```

```python
mse_test
```

```
2.336543964552525
```

Figure 3: Solving Regression problem using best k (k=10) from MIQP problem.



```
cv = KFold(n_splits=10, shuffle=False)
```

```
lasso = LassoCV(cv=cv).fit(X_train[:,1:], y_train)
```

```
reg = lasso.alpha_
reg
```

```
0.07638765995113514
```

```
se_test = (X_test[:,1:] @ lasso.coef_ - y_test).T @ (X_test[:,1:] @ lasso.coef_ - y_test)
```

```
mse_test = se_test / len(y_test)
mse_test
```

```
3.594714573295883
```

```
n_features = sum(lasso.coef_ != 0)
n_features
```

```
17
```

Figure 4: Using Scikit-Learn to solve LASSO regression for variable selection.