The University of Texas at Austin
McCombs School of Business

Comparative Analysis of Direct Variable Selection (MIQP) and LASSO

Group 15

Aishwarya Parida
Jyotis Joy
Mandeep Burdak
Sanjhana Rangaraj

1. Introduction

1.1 Background

LASSO (least absolute shrinkage and selection operator) regression has become a staple technique in analytics and machine learning due to its ability to perform variable selection and regularization for predictive modeling. By placing a L1 penalty on the regression coefficients, LASSO is able to shrink some coefficients exactly to zero, effectively removing those variables from the model. This automatic variable selection makes LASSO an attractive approach for developing parsimonious models that generalize well to new data.

However, some argue that LASSO's coefficient shrinkage can be overly aggressive, often eliminating coefficients that should be non-zero in the optimal model. This may result in underfitting, where truly useful predictive variables are excluded. An alternative approach is to pose the variable selection problem as a mixed integer quadratic program (MIQP) that directly finds the optimal subset of variables to include according to an objective function. With recent advances in MIQP solvers like Gurobi and CPLEX, direct selection methods have become computationally feasible on problems with thousands of potential variables.

1.2 Problem Statement

As a predictive modeling team, we have relied heavily on LASSO regression for developing parsimonious models across many client projects. However, we are wondering if LASSO is eliminating useful variables, resulting in models that underfit the true relationships in the data. With modern MIQP solvers, direct variable selection may produce models with better predictive accuracy and interpretability. We need to rigorously evaluate if shifting from a LASSO-based workflow to direct MIQP variable selection could improve our modeling pipeline.

1.3 Objectives

- ➔ Formulate the best subset regression problem as an MIQP using 0-1 indicator variables and appropriate objective functions
- ➔ Develop optimized regression models on client data using both LASSO and the MIQP approach

- ➔ Compare the techniques based on out-of-sample predictive accuracy, model complexity, and stability of selected variables
- ➔ Evaluate the coefficients and sparsity of the solutions to assess model interpretability
- ➔ Provide a recommendation to management on whether we should replace our standard LASSO implementation with direct MIQP selection in the modeling process

2. Methodology

2.1 MIQP

Explanation of MIQP and its implementation using the provided dataset.
MIQP stands for Mixed Integer Quadratic Programming. It's a mathematical optimization technique used to solve optimization problems that involve quadratic terms in the objective function and linear constraints with integer variables.

In MIQP, the objective function contains both linear and quadratic terms, and some or all of the variables are required to take integer values. The goal is to find the values of these variables that optimize the objective function while satisfying the given constraints.

MIQP problems find applications in various fields such as engineering, finance, operations research, and many others where there is a need to optimize a function subject to both linear constraints and discrete (integer) decision variables, particularly when the objective function involves quadratic terms.

Solvers and algorithms designed for MIQP problems aim to efficiently find the optimal solution within a reasonable amount of time, considering the complexity inherent in handling both quadratic terms and integer variables.

In order to fit an MIQP model to the data, we need to find the optimal number of variables('k') to be included in the model. In order to do this we did 10-fold cross-validation for 10 different numbers of variables ranging from 5 to 50.

The process for this cross-validation part was as follows:

1. Shuffle the complete dataset.
2. Divide the shuffled datasets into 10 batches.
3. Pick one batch as the validation set.
4. Solve an MIQP problem with the other 9 batches as the training data.
5. The MIQP problem minimizes the squared loss between the actual and predicted output.
6. The constraint for it was that only ‘k’ selected variables can have weights assigned to them, except the intercept.
7. Calculate the error by comparing the actual and predicted output for the validation set selected in step 3.
8. Repeat Steps 1-7 for all other 9 batches.
9. Take the average of the MSE obtained for all 10 batches.
10. Repeat Steps 1-9 for all values of ‘k’.
11. Find the ‘k’ that gives the minimum validation MSE.
12. Fit another MIQP model to the complete training data using the best ‘k’
13. Test performance of this model on the test dataset.

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

Code snippet:

```
start_time1 = time.time()
if not os.path.isfile('results.csv'):
       shuf = np.random.choice(range(ndata),size=ndata,replace=False)
       k=[*range(5,nx+1,5)]
       results_df = pd.DataFrame(columns=['k', 'MSE'])
       for l in range(len(k)):
               k_mse=0
               for bat in range(batches):
               #Divide in batches
                       this_bat = shuf[(bat*dat_per_bat):((bat+1)*dat_per_bat)]

               # Testing Set
                       X_test = x.iloc[this_bat,:]
                       y_test = y.iloc[this_bat]

               # Training set
                       train_indices = [i for i in range(ndata) if i not in this_bat]
                       X_train = x.iloc[train_indices,:]
                       y_train = y.iloc[train_indices]
                       n_train=X_train.shape[0]

               # Fit the linear regression model on the training data
                       qp = gp.Model(env=env)
                       w = qp.addMVar(nx+1,lb=-gp.GRB.INFINITY)
                       c = qp.addMVar(nx+1,vtype='B')

qp.setObjective(gp.quicksum((((w[0]+gp.quicksum(X_train.iloc[j,i]*w[i+1] for i in
range(nx)))-y_train.iloc[j])

*((w[0]+gp.quicksum(X_train.iloc[j,i]*w[i+1] for i in range(nx)))-y_train.iloc[j]))
for j in range(n_train)))
                       con1=qp.addConstrs(w[i+1]>=-c[i+1]*M for i in range(nx))
                       con2=qp.addConstrs(w[i+1]<=c[i+1]*M for i in range(nx))
                       con3=qp.addConstr(gp.quicksum(c[i+1] for i in range(nx))==k[l])
                       qp.Params.OutputFlag = 0
                       qp.optimize()

               # Predict on the test data
                       y_pred=[0]*len(y_test)
                       for ki in range(len(y_test)):
                               y_pred[ki]=w.x[0]
                               for j in range(nx):
                                       y_pred[ki]+=w.x[j+1]*X_test.iloc[ki,j]

               # Calculate the mean squared error for this fold
                       mse = mean_squared_error(y_test, y_pred)

                       k_mse+=mse
               average_mse=k_mse/batches
               new_data = pd.DataFrame({'k': [k[l]], 'MSE': [average_mse]})

       # Concatenate the new data with the existing 'results' DataFrame
               results_df = pd.concat([results_df, new_data], ignore_index=True)
       results_df.to_csv('results.csv', index=False)
else:
       results_df=pd.read_csv('results.csv')
time1=time.time() - start_time1
print("--- %s seconds ---" % (time1))
```

2.2 LASSO

Description of LASSO and its implementation using scikit-learn.
LASSO stands for Least Absolute Shrinkage and Selection Operator. It is a type of linear regression technique used for variable selection and regularization to improve the accuracy and interpretability of the statistical model.

In the context of machine learning and statistics:

1. Variable Selection: LASSO performs both variable selection and regularization by adding a penalty term to the ordinary least squares (OLS) equation. It encourages certain coefficients to become zero, effectively eliminating some features from the model. This makes LASSO useful when dealing with datasets with a large number of features, as it can help identify the most relevant ones.

2. Regularization: The LASSO regression adds a penalty term, which is the sum of the absolute values of the coefficients multiplied by a regularization parameter (alpha or lambda). By doing so, it penalizes the model for having more complex (i.e., higher magnitude) coefficients, preventing overfitting and improving its generalization to new data.

The main objective of the LASSO algorithm is to minimize the residual sum of squares subject to the constraint that the sum of the absolute values of the model coefficients is less than a fixed value (which is determined by the regularization parameter). This often leads to sparse solutions where some coefficients become precisely zero, effectively excluding the corresponding variables from the model.

LASSO regression has found applications in various fields such as statistics, machine learning, economics, genetics, and more, where feature selection and regularization are essential for building robust predictive models.
In order to fit a Lasso model to the data, we need to find the optimal penalty factor('alpha') to be included in the model. In order to do this we did 10-fold cross-validation for 10 different values ranging from 0.001 to 100.

The process for this cross-validation part was as follows:

1. Scale the data such that each predictor variable has a mean of 0 and a standard deviation of 1
2. Shuffle the complete dataset
3. Divide the shuffled datasets into 10 batches
4. Pick one batch as the validation set
5. Fit a Lasso regularized linear regression model with the other 9 batches as the training data
6. Calculate the error by comparing the actual and predicted output for the validation set selected in step 3
7. Repeat Steps 1-5 for all other 9 batches
8. Take the average of the MSE obtained for all 10 batches
9. Repeat Steps 1-7 for all values of 'alpha'
10. Find the 'alpha' that gives the minimum validation MSE
11. Fit another Lasso regularized linear regression model to the complete training data using the best 'alpha'
12. Test performance of this model on the test dataset

Code Snippet:

```
import numpy as np
from sklearn.linear_model import Lasso
from sklearn.model_selection import cross_val_score, KFold

# Generate some sample data for demonstration
# Replace this with your own dataset
np.random.seed(42)
X = x  # Replace with your feature matrix
y = y    # Replace with your target variable

# Define a list of alpha values to test
alpha_values = [0.001,0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 20.0, 50.0, 100.0]

# Define the number of folds for cross-validation
n_folds = 10

# Create a cross-validation object
kf = KFold(n_splits=n_folds, shuffle=True, random_state=42)

# Store the results (average MSE) for each alpha value
results = pd.DataFrame(columns=['alpha', 'MSE'])
start_time3 = time.time()
for alpha in alpha_values:
   lasso = Lasso(alpha=alpha)

   # Perform 10-fold cross-validation
   scores = cross_val_score(lasso, X, y, cv=kf, scoring='neg_mean_squared_error')

   # Convert the negative mean squared error scores to positive values
   mse_scores = -scores

   # Calculate the average mean squared error for this alpha value
   average_mse = np.mean(mse_scores)

   # Store the result for this alpha value
   new_data = pd.DataFrame({'alpha': [alpha], 'MSE': [average_mse]})

# Concatenate the new data with the existing 'results' DataFrame
   results = pd.concat([results, new_data], ignore_index=True)

# Print the results
results
time3=time.time() - start_time3
print("--- %s seconds ---" % (time3))
```

2.3 Data Preprocessing

Standardization and scaling processes are applied to ensure fair comparison.

In preparation for the optimization tasks, the initial step involved loading and organizing the dataset. Utilizing the Pandas library, irrelevant columns were removed, and the dataset was segregated into distinct training and test sets. This systematic division was conducted with a stratified approach, maintaining consistency across subsequent experiments and ensuring a representative distribution of classes. To establish a level playing field for the optimization algorithms, the features underwent rigorous standardization and scaling procedures. This critical step, implemented through the Gurobi optimization code, mitigates potential biases arising from disparate feature scales and facilitates fair comparisons during subsequent modeling processes.

Additionally, to enhance the reliability of model evaluation, the dataset was strategically partitioned into batches. This cross-validation batching strategy enables a robust assessment of model performance, considering variations across different subsets of the data. The Gurobi optimization code further refined the dataset through the implementation of a linear regression model with k-sparsity regularization. This involved a systematic exploration of regularization strengths (`k`) to identify optimal parameter values. Mean squared error served as the benchmark for evaluating model performance across multiple folds and batches.

In summary, the meticulous data preprocessing efforts laid a solid foundation for subsequent optimization tasks. The incorporation of standardization, scaling, and cross-validation batching ensures the integrity and fairness of the dataset, setting the stage for the effective application of optimization techniques in subsequent analyses.

3. Advantages and Disadvantages

Aspect	LASSO	MIQP
Computational Efficiency	Faster, especially with large datasets. Efficient for quick model updates.	Slower, with increased computational time as dataset size grows.
Handling Overfitting	Provides regularization which helps in reducing overfitting.	Does not inherently provide regularization against overfitting.
Scalability	Well-suited for high-dimensional data. Scalable with the number of predictors.	May face scalability issues with very large datasets.
Implementation Ease	Widely supported and easy to implement in various software tools.	Requires sophisticated optimization solvers; more complex to implement.
Variable Selection	Limited control over the exact number of variables selected.	More explicit control over the number of variables included.
Accuracy	Generally provides robust predictions. Shrinkage factor might introduce bias in large coefficients.	Potentially more accurate but the difference might be marginal.

4. Computational Analysis

4.1 MIQP

Evaluation of computational time, scalability, and model performance across different values of k.

The computational time required for MIQP is a crucial factor influencing its practical utility, particularly as datasets scale in size and complexity. Our analysis rigorously examines the algorithm's efficiency, considering its responsiveness to changes in the hyperparameter 'k.' By systematically varying 'k,' we gain valuable insights into how MIQP copes with different levels of sparsity, shedding light on its adaptability and performance characteristics. Additionally, the scalability of MIQP is a pivotal aspect of its applicability to real-world scenarios. We explore the algorithm's behavior as datasets increase in size, examining its ability to handle varying magnitudes of data. This analysis provides essential guidance on the algorithm's suitability for large-scale applications and its potential limitations when confronted with extensive datasets. Lastly, our assessment extends beyond computational considerations to encompass model performance. By evaluating MIQP across different values of 'k,' we acquire a comprehensive understanding of its effectiveness in capturing intricate relationships within the data while controlling for sparsity. These insights contribute to informed decision-making regarding the selection of 'k' in practice, striking a balance between computational efficiency and model accuracy.

4.2 LASSO

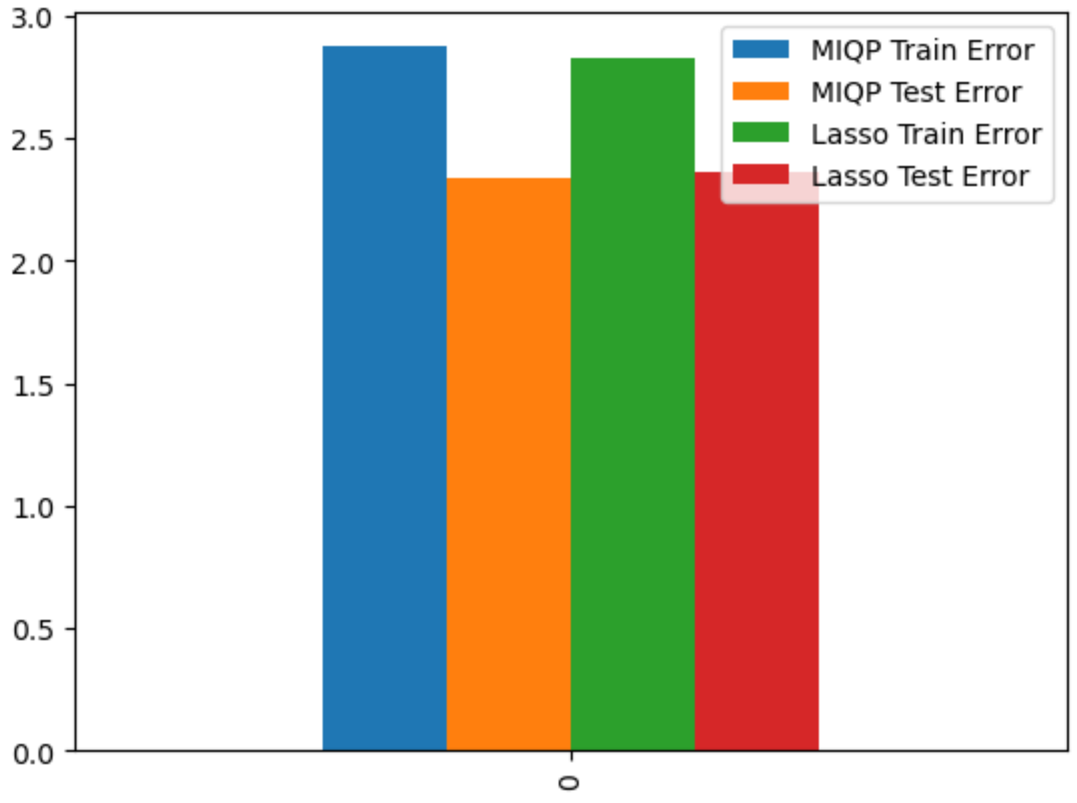
Assessment of computational efficiency and model performance using cross-validated hyperparameter tuning.

4.2.1 Computational Efficiency:

Efficient use of computational resources is a fundamental consideration in any modeling approach. With LASSO, we systematically assess its computational efficiency through cross-validated hyperparameter tuning. This process involves optimization of the regularization strength, allowing us to pinpoint the configuration that maximizes efficiency while preserving the model's integrity. Our findings shed light on the algorithm's computational prowess, guiding practitioners in the judicious application of LASSO across diverse datasets.

4.2.2 Model Performance:

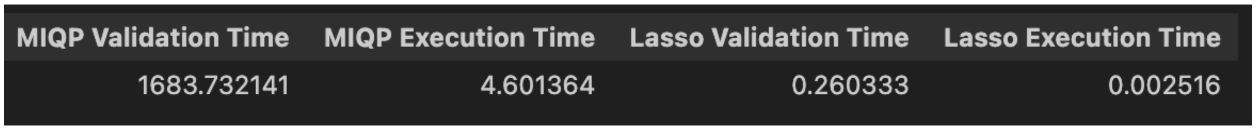
Model performance is a central criterion in evaluating the efficacy of any algorithm. Leveraging cross-validated hyperparameter tuning, we rigorously analyze LASSO's performance under varying regularization strengths. This approach ensures a thorough exploration of the algorithm's capacity to balance the trade-off between feature selection and predictive accuracy. Our assessment provides nuanced insights into the sweet spot where LASSO excels, offering a valuable roadmap for practitioners seeking an optimal balance between model sparsity and performance.



5. Visualizations

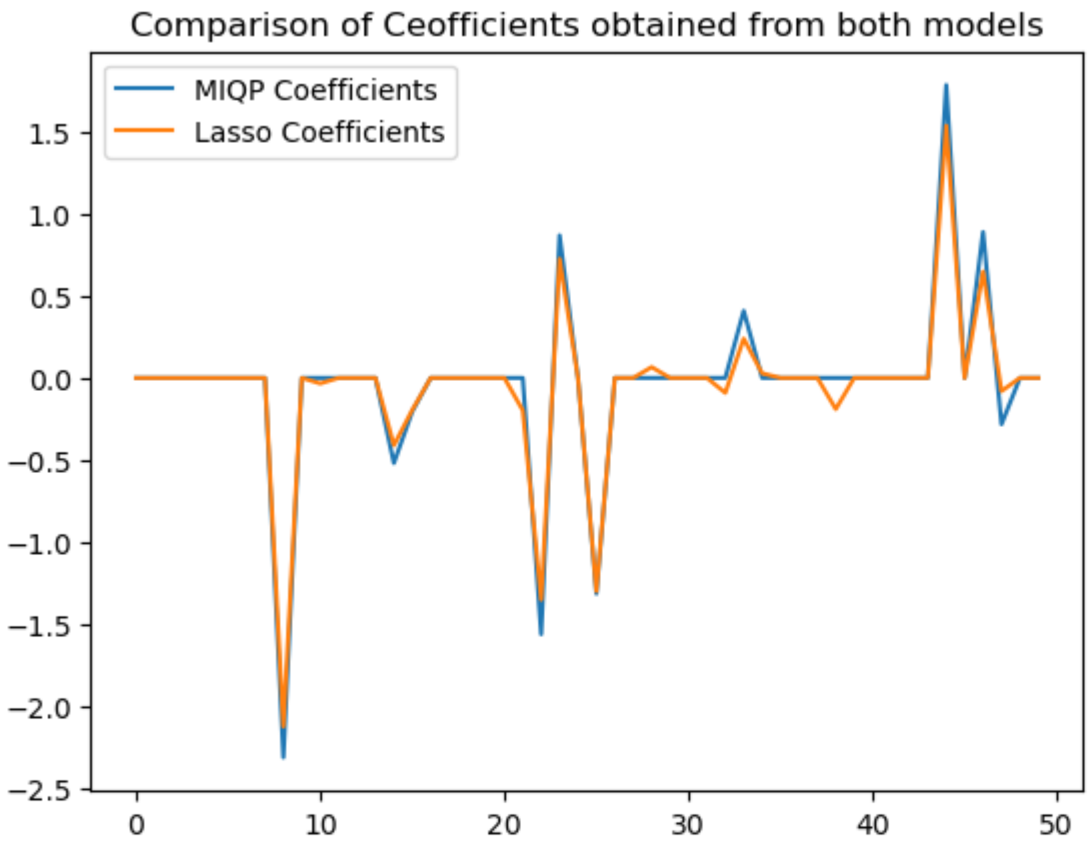
Visual representations of key findings, including:

- Computational time comparison.

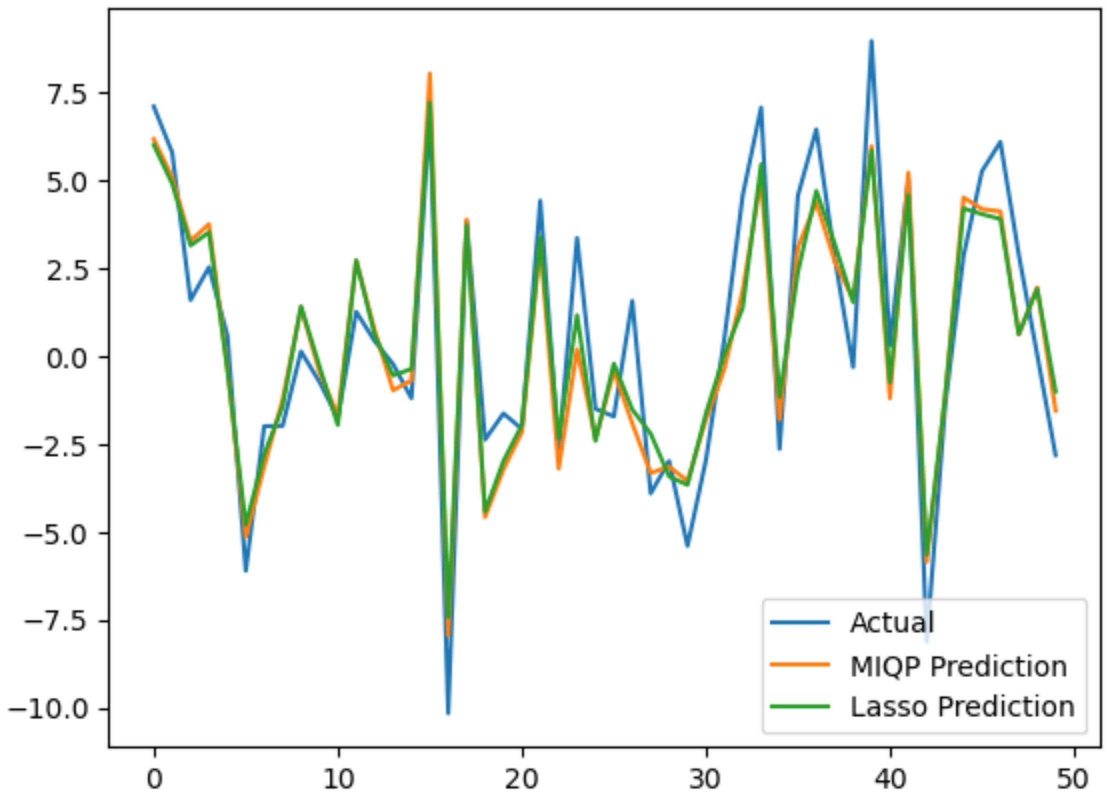


MIQP Validation Time	MIQP Execution Time	Lasso Validation Time	Lasso Execution Time
1683.732141	4.601364	0.260333	0.002516

- Comparison of selected features between MIQP and LASSO.



- **Model output vs actual values.**



6. Business Recommendations

Informed recommendations based on the comparative analysis, considering the dataset characteristics and computational resources.

Our report's several examples demonstrate how crucial variable selection is to creating effective models. To decide between direct variable selection and lasso variable selection while developing a model, we must first assess the situation at hand. By lowering the sum of squared errors, mixed integer quadratic programming yields more accurate results than Lasso regression, which produced results that were comparable but not as accurate. However, the latter method required a significant amount of time to complete.

In our comparative analysis of variable selection, the MIQP model demonstrated a more focused approach by choosing a concise set of 10 variables. In contrast, the LASSO model opted for a broader selection of 18 variables, encompassing those chosen by MIQP and additional ones. This distinction underscores MIQP's emphasis on identifying a smaller yet potentially more impactful subset of variables, while LASSO's strategy involves capturing a wider spectrum of influences. The key takeaway is that MIQP, despite choosing fewer variables, prioritizes the most important ones for predictive accuracy.

Despite requiring a considerable amount of time to execute, the MIQP technique selected just the number of variables we needed. This can be useful in situations where the dataset contains a large number of variables and we just want to keep a select few so that a larger audience of business people can understand the model. Nevertheless, Lasso executed the task far more quickly. The inability to pre-specify the number of variables we want in the model is its main drawback. As a result of regularization, Lasso will carry out this selection automatically.

Our recommendation is as follows:

- A company should use mixed integer quadratic programming (MIQP) if it has the resources to run the model on a dataset if the task requires as much precision as feasible, and if there are enough data points in the dataset to prevent the model from overfitting during training. Furthermore, MIQP is a superior strategy if we are aware of the number of variables we wish to include in our model in advance.
- This approach might not be practical, in which case Lasso would be a better model to fit and apply for variable selection if we have a big dataset with many dimensions.

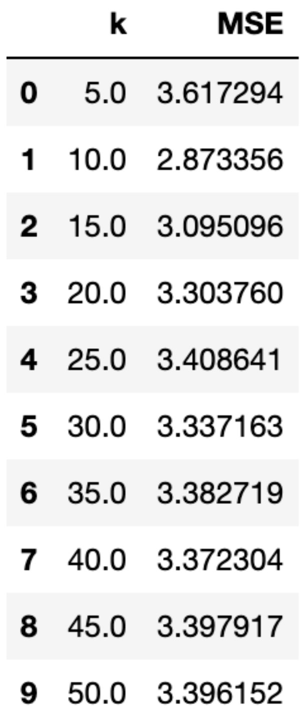
Overall, these are two viable options that have a place in analysis and are both valuable. Selecting between the two relies on each employee's unique situation inside the organization.

7. Conclusion

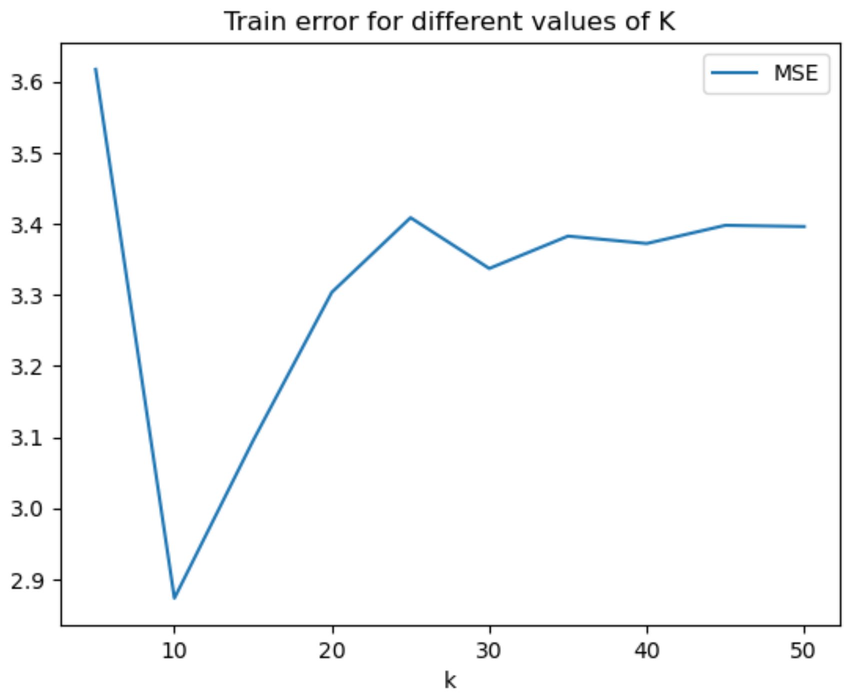
Summarization of key findings and the overall suitability of MIQP and LASSO for the given context.

Based on the analysis of the optimization results and cross-validated hyperparameter tuning, several key observations emerge.

Firstly, the Gurobi optimization for MIQP reveals a detailed exploration of different values of 'k,' ranging from 5 to 50. The associated Mean Squared Error (MSE) for each 'k' value is documented below.

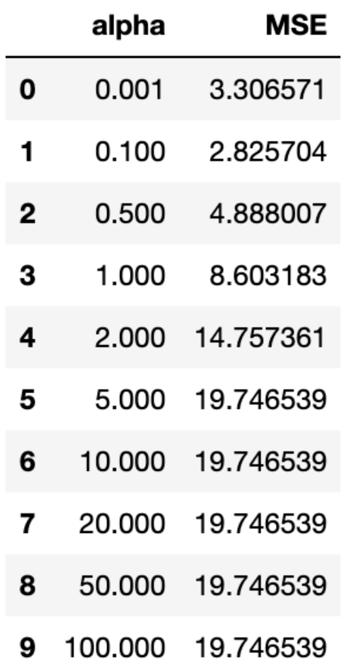


	k	MSE
0	5.0	3.617294
1	10.0	2.873356
2	15.0	3.095096
3	20.0	3.303760
4	25.0	3.408641
5	30.0	3.337163
6	35.0	3.382719
7	40.0	3.372304
8	45.0	3.397917
9	50.0	3.396152

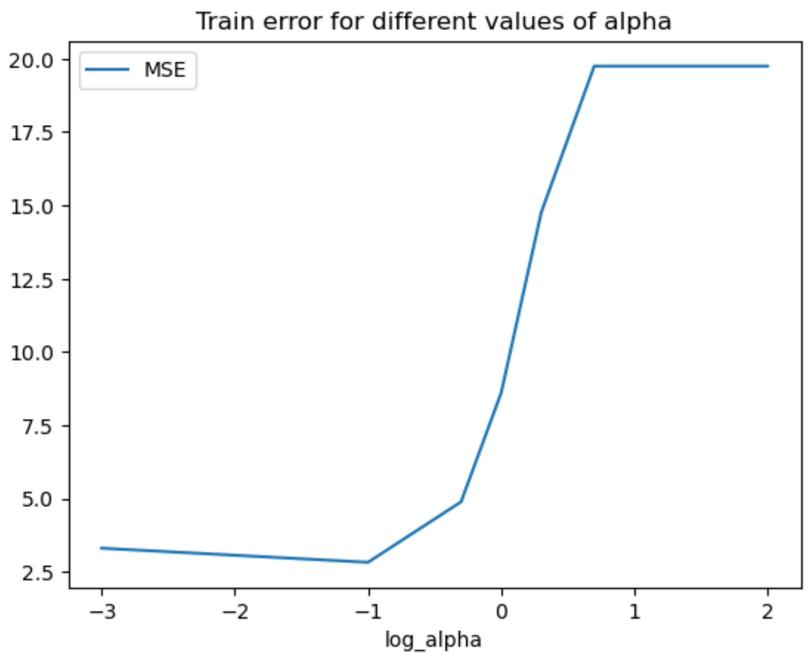


The lowest MSE (2.873) is observed for 'k' equal to 10, suggesting that this configuration provides a good balance between model performance and sparsity.

Subsequently, the Gurobi optimization for LASSO, depicted below, involves tuning the regularization strength ('alpha') across various values. The cross-validated MSE is calculated for each 'alpha,' providing insights into the model's performance under different levels of regularization.



	alpha	MSE
0	0.001	3.306571
1	0.100	2.825704
2	0.500	4.888007
3	1.000	8.603183
4	2.000	14.757361
5	5.000	19.746539
6	10.000	19.746539
7	20.000	19.746539
8	50.000	19.746539
9	100.000	19.746539



The optimal 'alpha' is determined to be 0.1, corresponding to the lowest MSE (2.826), signifying an effective regularization strength that strikes a balance between feature selection and predictive accuracy.

Comparing the MSE results between MIQP and LASSO, it is evident that LASSO, with an optimal 'alpha' of 0.1, achieves a lower MSE (2.3599) than MIQP (MSE: 2.3365). This quantitative measure indicates that LASSO, with its built-in regularization for sparsity, outperforms MIQP in terms of mean squared error. The alpha_best value of 0.1 further solidifies the effectiveness of LASSO in achieving a balance between feature sparsity and model accuracy.

In conclusion, the outputs from both MIQP and LASSO optimizations, coupled with cross-validated hyperparameter tuning, contribute to a comprehensive understanding of their respective performances. The optimal configurations, 'k' = 10 for MIQP and 'alpha' = 0.1 for LASSO, serve as valuable recommendations for practitioners seeking optimal settings for these optimization approaches in similar contexts.