Part 1 - Machine Learning on Real Data

1. Description of Data

We use the **House Sales in King County, USA** dataset for our analysis. The dataset contains house sale prices for King County, which includes Seattle for homes sold between May 2014 and May 2015.

1.1 Data Description

The various features of a the house sold are used for the anlaysis. These are - The Unqiue ID for the house (**id**), Date house was sold (**date**), Price at which it was sold (**price**), Number of bedrooms and bathrooms (**bedrooms**, **bathrooms**), the Square Footage of the home and lot (**sqft_living**,**sqft_lot**), number of floors (**floors**), if it has a Waterfront (**waterfront**), the View (**view**), the house Condition (**condition**), Rating given to the house (**grade**), square footage of house apart from basement (**sqft_above**), square footage of the basement (**sqft_basement**), built year (**yr_built**), renovated year (**yr_renovated**), zipcode (**zipcode**), latitude and longitudes (**lat**, **long**), Living room area and lot size in 2015 (**sqft_living15**, **sqft_lot15**)

1.2 Research Question

We wish to predict the **Price** of a house in King County using the other features.

1.3 Preparation of Data

1.3.1 Cleaning of Data

We check for any null values, which in our case are none. Further, We remove the **ID** and **Zip Code** features in the data as they don't help in our analysis (either unique for each row or too many unqiue values). The datatype of the **date** feature is changed so it can be interpreted as a date. Some features like **yr_renovated** and **sqft_basement** have too many values = 0, we can convert these to categorical variables where value is either 0 or 1 (all non zero values mapped to 1). Lastly, values of features like **bathroom** and **floors** are rounded off to integer values.

1.3.2 EDA

- Plotting the correlation matrix, we observe that **price** has a relatively strong positive correlation with features like **bedrooms**, **bathrooms**, **sqft_living**, **sqft_above** and **grade**. *(Figure 1)*
- Plotting the Boxplot and Histograms of the response variable, we observe that it has a lot of outliers and is rightly skewed. However, the log of the **price** is approximately normally distributed. *(Figure 2 & 3)*
- We also plot visualizations of different features with our response variable, **price**, and try to observe any patterns. Many features, like **sqft_living** seem to follow a linear pattern with the **price** *(Figure 4)*

1.3.3 Feature Engineering

- Since only two houses had **bedrooms**>10, we drop those 2 values. Some other variables, like **waterfront**, are converted to categorical variables. Further, the Year of Built of the houses is divided into 3 intervals and the feature is mapped so that each house has Built year as one of these intervals.

2. Review of Approaches Tried

2.1 Regression

2.1.1 Linear & Log Linear Regression (as the response variable is right skewed)

We perform Linear Regression and calculate the Mean Square Error. We observe from the fitted model that there are many variables that do not contribute significantly to the response variable. The MSE is 34613676333. Due to the skewness in the response variable, we try the approach of performing Linear Regression on the log of the **price** feature. Our calculated MSE for this is 36074236094.

2.1.2 Best Subset Selection

We also perform the **Best Subset Selection** *(Figure 5)* approach to find the features that we should use in our predictions. After performing the Best Subset Selection and training the model with the selected features, we get the MSE as 34646016404.

2.1.3 Lasso and Ridge Regression

We try to improve on the Linear Regression model by introducing the Lasso and Ridge penalties.

Ridge regression does not perform any feature selection, however, it does prevent the model from overfitting by restricting the coefficients. We choose the appropriate lambda value using cross validation so as to minimize the MSE. *(Figure 6).* After using Ridge Regression, our MSE is 34500445644, which is an improvement from the previous models.

We perform Lasso Regression as it performs feature selection on our data. *(Figure 7).* Using Cross Validation to select the value of lambda (using the One-Standard-Error Rule) and then training the model using the selected features, we get the MSE of 34591063624, an improvement from the previous linear regression models!

2.1.4 Principal Component Regression

We perform PCR and select the number of components using Cross Validation and the One-Standard Error Rule *(Figure 8 & 9).* After training the model using the selected components, we get MSE = 34744802347.

2.1.5 Generalized Additive Models (GAM)

We make use of Generalized Additive Models to capture any non linear relationships between the features and the response variable. We observe that all our features are significant. We fit smoothing splines to the data. Our MSE for this fit is 25632462466. A significant improvement to the linear regression models!

2.1.6 Random Forest

We use the Random Forest Regressor on the data. Since random forest is not so sensitive to the number of trees we selected, for the sake of computational efficiency, we simply choose number of trees = 200 rather than tuning the hyperparameter. But we can tune the model by changing the number of variables randomly sampled at each stage (mtry). We get an MSE of 15113885136, which is significantly lesser than the previous models.

2.1.7 Gradient Boosting

We next perform Gradient Boosting. Boosting is a sequential process in which each next model which is generated is added so as to improve a bit from the previous model. So we use multiple decision trees and keep improving the model. We tune the parameters by trying various values for maximum depth of each tree, learning rate, number of trees, fraction of training set randomly selected and other tuning parameters. The summary of the model gives the importance plot that shows the relative influence (or importance) of each feature. *(Figure 14)* We choose the model which gives the lowest MSE. This technique takes very long to execute but gives a low MSE of 15233280275.

2.1.8 XG Boosting

XGBoost is Extreme Gradient Boost. We perform XGBoost along with Cross Validation. We specify the grid space to search for the best hyperparameteres for our model, and also specify the number of folds for cross validation. We get an MSE of 14630192591. This is a low MSE compared to other models and the execution time is also less.

2.1.9 Neutral Network

Finally we try to make the predictions by making a Neural Network *(Figure 10).* We get MSE = 17482355981.974.

2.2 Classification

As for the classification methods, we first convert the dependent variable into three levels: **High**, with the price larger than 64500; **Medium**, with the price between 321725 and 645000; **Low**, with the price lower than 321725.

2.2.1 Multinomial Logistic Regression

We perform Multinomial Logistic Regression on the transformed data. After training the data 100 iterations, we get the error rate equal to 0.2137405.

2.2.2 Linear Discriminant Analysis(LDA)

We apply Linear Discriminant Analysis on the data. Error rate for this fit is 0.2324774.

2.2.3 Quadratic Discriminant Analysis (QDA)

We apply Quadratic Discriminant Analysis on the data. We encounter an error about rank deficiency if we try to train to the model on all the features. This is because some variables are collinear and one or more covariance matrices cannot be inverted to obtain the estimates in the group 'Low'. This issue can be resolved by removing strongly correlated data elements. Once this is done we get an error rate of 0.4617164.

2.2.4 K Nearest Neighbors(KNN)

We perform KNN method on the data *(Figure 11).* Use Cross Validation to get the optimal k with the value 19, and then training the model with selected k. Our error rate is 0.3687254, which is a little larger. Also, the prediction speed is slower than other algorithms.

2.2.5 Classification Tree

We next make use of Classification Tree method since it is easier to visualize *(Figure 12)* and the corresponding error rate is 0.2489012.

2.2.6 Random Forest

Then we use an improved tree method Random Forest *(Figure 13).* Error rate is 0.1503585, which does far better than the previous models.

3. Final Approach

3.1 Comparing Regression Models

Model	MSE	Elapsed Time
Linear Regression	3.46 x 10^{10}	0.04 Seconds
Log-Linear	3.60 x 10^{10}	0.2 Seconds
Best Subset Selection	3.46 x 10^{10}	1.4 Seconds
Ridge Regression	3.45 x 10^{10}	0.5 Seconds
Lasso Regression	3.45 x 10^{10}	0.6 Seconds
Principal Component Regression	3.47 x 10^{10}	2.5 Seconds
Generalized Additive Models	2.56 x 10^{10}	2.4 Seconds
Random Forest	1.51 x 10^{10}	28.5 Minutes
Gradient Boosting	1.52 x 10^{10}	1.15 Hours
XG Boosting	1.46 x 10^{10}	3.6 Minutes
Neutral Network	1.74 x 10^{10}	2.2 Minutes

We observe that the best performance for Regression is when we use Random Forest, Gradient Boosting, XG Boosting, or Neural Networks.

3.2 Comparing Classification Models

Model	Error Rate	Elapsed Time
Multinomial Logistic Regression	0.2137405	1.770615 secs
Linear Discriminant Analysis	0.2324774	0.119278 secs
Quadratic Discriminant Analysis	0.4617164	0.050426 secs
K Nearest Neighbors	0.3687254	3.712813 mins
Classification Tree	0.2489012	1.780733 secs
Random Forest	0.1503585	27.67016 secs

The best performance is when we use Random Forest for Classification.

4. Results and Conclusion

If we treat this problem as a Regression problem to calculate the price of a house using the various features, we should make use of a model that gives us a good result, but is also efficient, as with greater data points, algorithms like Gradient Boosting and Random Forests may take several hours. We therefore choose **XG Boost** as our final model since it gave us a much better result (lower MSE) than other models and a very fast execution time. Our MSE with XG Boost is 1.46 x 10^{10} with a computation time of only 3.6 Minutes.

If we treat this problem as a Classification problem, it is clear from the error rates that **Random Forest** is the best algorithm to use. It takes longer than the other algorithms, but significantly reduces the error rate. The error rate with Random Forest is 0.1503585 and the computational time is 27.67016 secs.

Part 2 : Estimation of Graphical Models using Lasso Related Approaches

This part of the project is focusing on using the methodologies of two lasso related approaches to estimate the graphical models, which will connect two variables together if their covariance is non-zero. Our aim is to apply the node-wise lasso approach and graphical lasso approach to recover to the true edge set that connects correlated nodes. We will first develop general methods to select the optimal tuning parameters λ for each approach respectively based on different criteria. Then, we conduct simulations with various settings by changing the number of observations (n), number of variables (p) and sparsity structures to observe the estimation accuracy according to classification rate and the area under the ROC curve. Finally, the results within as well as across each method will be compared and discussed.

2.1 Node-wise Lasso Approach

2.1.1 Introduction of Node-wise Lasso Approach

The main procedure of generating a graphical model is to access the covariance between each pair of variables (c_{jl}). One of the feasible solutions is to regress each X_j on the other remaining variables X_l, and then the estimations are given by the derived coefficients $\hat{\beta}_{jl}$ and $\hat{\beta}_{lj}$. We could judge whether connects these two variables based on whether the derived coefficients are zero or not. In addition, the regression without penalty will result in all non-zero coefficients, therefore, in order to obtain a sparse solution, we add the l_1 penalty, which, could force some of the coefficient estimates to be exactly zero to rule out uncorrelated pairs that should not be connected.

The lasso coefficient, in this case, is estimated using the below objective function:

$$\sum_{i=1}^{n}\left(X_j - \sum_{1\leq l\leq p, l\neq j} \widehat{\beta_{jl}}\ X_l\right)^2 + \lambda \sum_{1\leq l\leq p, l\neq j}\left|\widehat{\beta_{jl}}\right| - - - - - - - - (1)$$

where $\lambda \geq 0$ is the regularization parameter. It controls the penalty strength and the variance and bias trade-off. To make the estimate of the relationship between each pair more reliable, one optimal tuning parameter λ should be selected.

Additionally, after getting the estimated coefficients, we further consider two possible scenarios, which use the 'joint' and 'or' rules respectively. Joint rules specify that we only connect the nodes of variable j and variable l if both $\hat{\beta}_{jl}$ and $\hat{\beta}_{lj}$ are non-zero while the 'or' rule connects the nodes as long as one of them is non-zero.

2.1.2 Selection of Optimal Tuning Parameters

Considering the computational cost, we are unable to select the optimal tuning parameters for each regression, therefore we choose one optimal lambda for all the regressions so that it could lead to generally better performance for recovering edges. Here, we use two main methods to select the optimal tuning parameters. The first one is based on the prediction accuracy of the regression model, which is measured by the five-fold cross-validation mean squared error. While the second one is based on the classification accuracy, and we use the true positive rate as the criteria. Then we compare the results from those two and choose the one that performs better.

- **Criteria 1 – Mean Squared Error**

Under this criterion, we choose the optimal lambda to be the one that minimizes the sum of cross-validation error of all the lasso regressions (2).

$$\hat{\lambda} = argmin_{\lambda} \sum_{j=1}^{p} (CV\ score_{\lambda}\ \ of\ jth\ regression) ------(2)$$

To be specific, we generate a $(p \times g)$ error matrix (3), with the number of rows equal to the number of variables while the number of columns equals the number of lambda values that are being considered.

$$\text{Error matrix} = \begin{bmatrix} E_{11} & \cdots & E_{1g} \\ \vdots & \ddots & \vdots \\ E_{p1} & \cdots & E_{pg} \end{bmatrix} -------- (3)$$

where the first Subscript denotes which variable is the response variable, while the second subscript denotes which lambda is being used.

Each element in the error matrix is the 5-fold-cross-validation mean squared error for one specific regression and one specific lambda, therefore, in column m $(1 \leq m \leq g)$, we could get the 5-fold-cv error for all the possible lasso regressions with each variable is being considered as the response variable while using lambda m. If sum up all elements for each column, then the overall mean squared error for each one specific lambda could hence be derived (4).

Overall

$$MSE_m = \sum_{j=1}^{p} E_{jm} \qquad 1 \leq m \leq g\ \ ------(4)$$

In the next step, those summed values are being compared and we choose the lambda that corresponds to the smallest summed overall MSE values. It is believed that this lambda could lead to a generally more accurate estimate for β.

- **Criteria 2 - True Positive Rate**

From the other aspect, if the lambda is chosen directly based on the recovering rate of the true edge set, then the classification rate may be a reasonable measurement. However, taking the sparsity structure into consideration, our dataset is fairly imbalanced as there are much more zero values, and therefore a larger penalty is always better as it will force almost all the coefficients to zero and we could in turn get a higher classification rate. Such a result cannot be trusted. Hence, since we have only few values equal to 1, we choose the optimal lambda to be the one that maximizes the sum of the True Positive rate of all the lasso regressions (5).

$$\hat{\lambda} = argmax_{\lambda} \sum_{j=1}^{p} (True\ Positive\ rate_{\lambda}\ \ of\ jth\ regression)\ \ ------- (5)$$

To be specific, we generate a $(p \times g)$ matrix (6), with the number of rows equal to the number of variables while the number of columns equals the number of lambda values that are being considered.

$$Matrix = \begin{bmatrix} E_{11} & \cdots & E_{1g} \\ \vdots & \ddots & \vdots \\ E_{p1} & \cdots & E_{pg} \end{bmatrix}\ \ --------(6)$$

where the first Subscript denotes which variable is the response variable, while the second subscript denotes which lambda is being used.

Each element in the matrix is the true positive rate for one specific regression and one specific lambda, therefore, in column m $(1 \leq m \leq g)$, we could get the true positive rate for all the possible lasso regressions for lambda m. If sum up all elements for each column, then the overall classification accuracy for each one specific lambda could hence be derived (7).

Overall

$$True\ Positive\ Rate\ _m = \sum_{j=1}^{p} E_{jm} \qquad 1 \leq m \leq g\ ------(7)$$

In the next step, those summed values are being compared and we choose the lambda that corresponds to the largest summed values, which, hopefully, could lead to a generally more accurate estimate for the true edges.

One additional issue that is worth mentioning is the range of lambda that we used in the process. Rather than directly specify a wide range of lambda values and choose from them, we manually zoomed out and updated the range.

Finally, we compare two methods based on the classification rate. We use the best lambda from each method, and then compute the percentage of right recovering of true edges. Criteria 2 - True Positive Rate gives a more reasonable result as the classification rate exceeds 90%. In the contrast, the lambda selected from the mean squared error measurement tends to predict too many coefficients to be non-zero, which, affects the overall recovering performance.

2.1.3 Simulation for 50 Times

Our initial setting is that $n = 200$, $p = 100$ and the sparsity structure is 90%. We iterate the procedure 50 times with different datasets that are generated randomly. The results as well as the boxplot('Joint' rule - blue, 'Or' rule -red) derived are displayed below. A detailed comparison between the node-wise lasso approach and the graphical lasso approach will be discussed in section 2.3.

- **For 'joint' rule:**

1) Mean of false positive rate (FPR) is 0.0092, the standard error is 0.0014.
2) Mean of false negative rate (FNR) is 0.7745, the standard error is 0.0186.
3) Mean of the classification rate is 0.9065.

- **For 'or' rule:**

1) Mean of false positive rate (FPR) is 0.0053, the standard error is 0.0011.
2) Mean of false negative rate (FNR) is 0.8237, the standard error is 0.0187.
3) Mean of the classification rate is 0.9045.

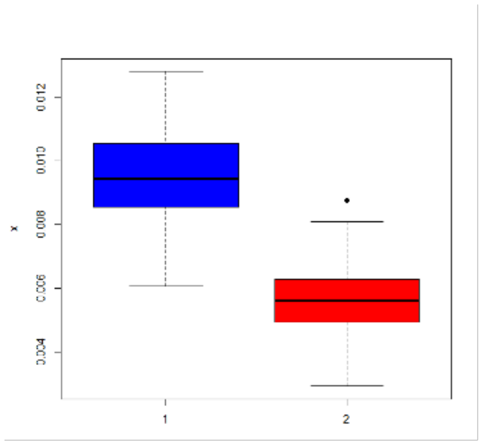


Figure 1: FPR for Node-wise Lasso

From the boxplots above, it could be observed that the false positive rate using the 'or' rule is lower than the 'joint rule', while for the false negative rate it is the opposite. Besides, the standard deviation, as well as the

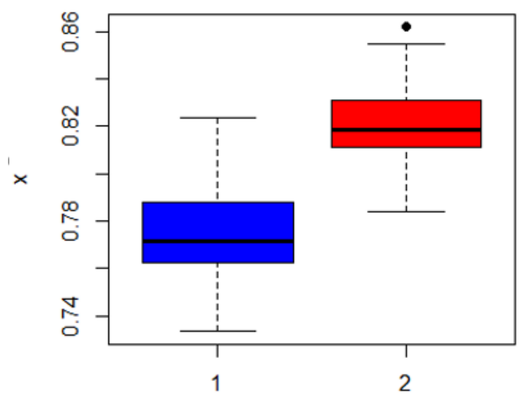


Figure 2: FNR for Node-wise Lasso

mean of the classification rate is quite similar between those two. Hence, in this situation, we conclude that the overall performance of these two rules is not varying too much, and different measurement criteria may give a different result.

2.1.4 Simulation Settings

- **Different values of n (number of observations) and p (number of variables)**

It is believed that the relatively small number of observations (n) and a relatively large number of variables (p) will result in the issue of overfitting, which affects the recovering of the true edge set and in turn, affects the estimation of graphic models. For that reason, we change the number of samples n and keep the number of variables (p) fixed to assess the outcome. In the simulation settings below ($table3$), we also fix the sparsity structure to be 90% (90% of elements in matrix B is 0).

Simulation Settings	n	p	Classification Rate (Joint)	Classification Rate (Or)
$n < p$	50	100	0.8902	0.8900
$n = p$	100	100	0.8998	0.8998
$n > p$	200	100	0.9056	0.9052
$n >> p$	1000	100	0.9131	0.9128

The results illustrate that the classification rate, as well as the area under the ROC (graph is presented and compared further in section 2.3.2) increases as the number of sample n increases, which, showing that the estimation of graphical models is getting more accurate, as well as proving the importance of the sample size.

- **Different sparsity structures**

Sparcity Structure	n	p	Classification Rate (Joint)	Classification Rate (Or)
80%	200	100	0.8334	0.8306
90%	200	100	0.9056	0.9052
95%	200	100	0.9412	0.9420

Since we selected the optimal lambda based on the criterion of the true positive rate, it is expected that the accuracy of prediction (classification rate) will rise as the sparsity level increases. The outcomes derived above demonstrates that point. However, it should be noticed that the result of the AUROC (graph is presented and compared further in section 2.3.2), which depicts the trade-off between the Ture Positive Rate and False Positive Rate is not guaranteed to increase as the sparsity level increases.

In addition, it is also noticeable that no matter how we change the n and p or change the sparsity structure, the 'joint' rule gives a better prediction than the 'or' rule in almost all cases. It could be explained that joint rules specify that the connecting of two nodes occur only when both coefficients are non-zero, which more firmly ensures the covariance between two variables is non-zero.

2.2 Graphical Lasso Approach

2.2.1 Introduction of Graphical Lasso Approach

Considering a data matrix X, a set of n observed p-dimensional realizations $x_1, x_2, \ldots, x_n$, which are generated from a multivariate normal distribution with zero mean and covariance matrix Σ (unknown). Assume its inverse $\Theta = \Sigma^{-1}$ is sparse, we use Graphical lasso approach to estimate the unknown Σ based on the n samples.

The graphical lasso approach minimizes a $l1$-regularized negative log-likelihood, which can be written as

$$min_{\Theta>0} f\left(\Theta\right) := -logdet\Theta + tr(S\Theta) + \lambda\Sigma_{j\neq l}|\theta_{jl}|$$

where

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}, S = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^T$$

is the sample covariance matrix of $x_1, x_2, \ldots, x_n$, and $\lambda \geq 0$ is the tuning parameter to control the sparsity level in Θ.

By minimizing over all positive definite matrices $\Theta \in \mathbb{R}^{p\times p}$, we obtain a sparse estimate $\hat{\Theta}$. θ_{jl} is the (j,l)-entry of the inverse covariance matrix. $\theta_{jl} = 0$ implies that the corresponding variables x_j and x_l are conditionally independent. If it is non-zero, there is an edge between node i and node j in corresponding undirected graph.

2.2.2 Selection of Optimal Tuning Parameters

λ is the tuning parameter controlling the amount of $l1$ shrinkage, and we apply Cross Validation to get the best values for tuning parameters by using R function cv.glasso. Under our initial setting, where $p = 100$, and $n = 200$, cross validation error is minimized at $\lambda = 0.1035$ as shown in figure 3.

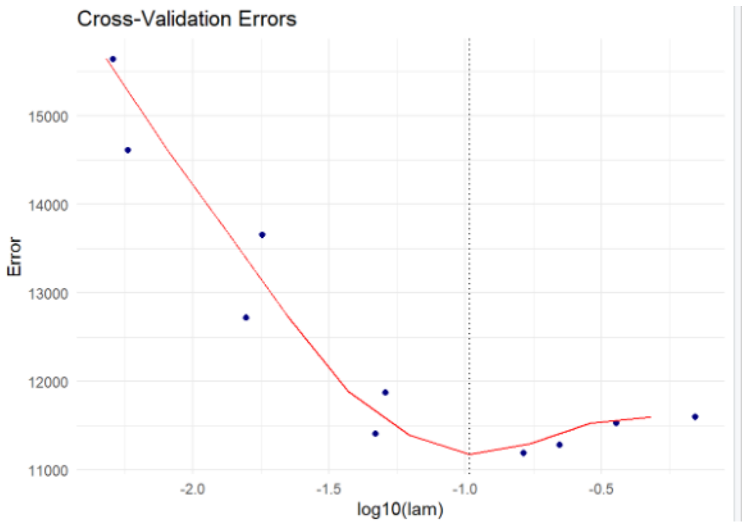


Figure 3: Cross Validation Error

Based on the selected optimal tuning parameter λ, we calculate several indicators to show the performance of Graphical Lasso Approach. Classification error rate is 0.7894. True positive rate is 0.7768 and false positive rate is 0.2090.

2.2.3 Simulation for 50 Times

We replicate the above procedure 50 times and each time we generate x randomly. Then, we obtain the mean (standard error) of FPR, FNR and classification rate of 50 simulations. Also, we draw the boxplots to show the distribution of FPR and FNR.

1) Mean of false positive rate(FPR) is 0.1737, the standard error is 0.0798.
2) Mean of false negative rate(FNR) is 0.2785, the standard error is 0.1067.
3) Mean of the classification rate is 0.8174.

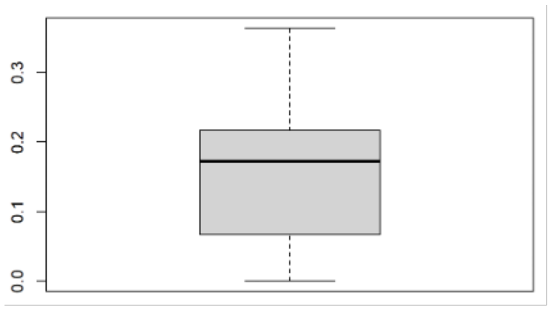


Figure 4: FPR for Graphical Lasso

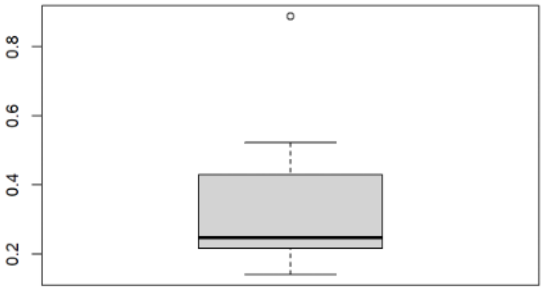


Figure 5: FNR for Graphical Lasso

2.2.4 Simulation with Various n, p and sparsity level

- **Different values of n (number of observations) and p (number of variables)**

Simulation Settings	n	p	Classification Rate
$n < p$	50	100	0.8311
$n = p$	100	100	0.9196
$n > p$	200	100	0.9398
$n >> p$	1000	100	0.9462

We use different simulation settings (different n,p) to assess the classification rate under the fixed sparsity structure 90% (90% of elements in matrix B is 0). The results again illustrate the importance of the sample size.

- **Different sparsity structures**

Sparcity Structure	n	p	Classification Rate
80%	200	100	0.8311
90%	200	100	0.8892
95%	200	100	0.9131

Similar to the node-wise lasso approach, the classification rate will increase as the sparsity level increases.

2.3 Compare the Node-wise Lasso Approach and Graphical Lasso Approach

2.3.1 Compare the Mean (standard error) of Relevant Measurements

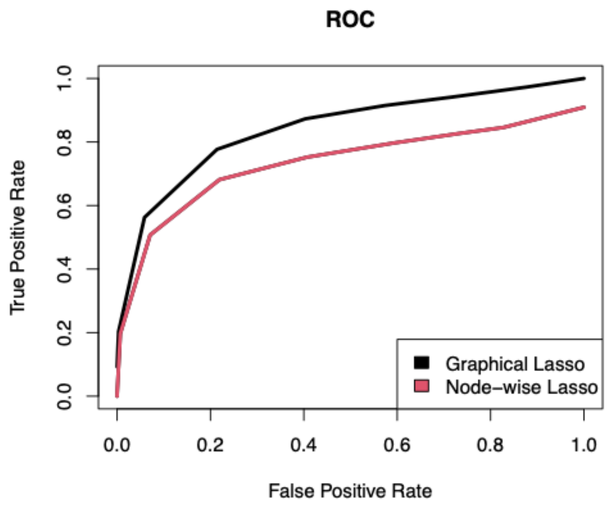


Figure 6: ROC for Node-wise Lasso and Graphical Lasso Approach

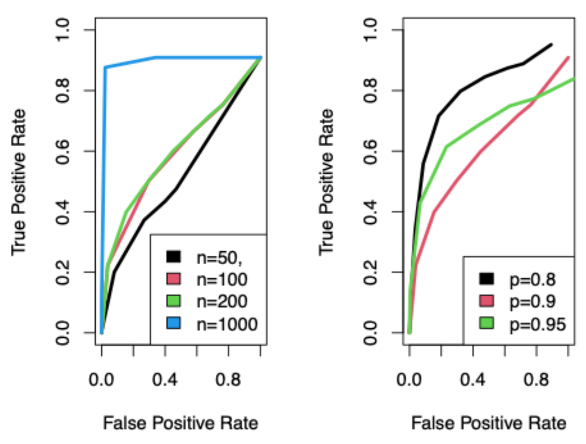
The ROC is derived by two models when using the optimal tuning parameters. The area under the Graphical Lasso ROC is 0.5463, which is large than the 0.5 under Node-wise Lasso Approach. Such an outcome proves the relative effectiveness of the trade-off between the sensitivity and the specificity of the graphical lasso approach, but it is still a concern that the value of AUROC is not large enough.

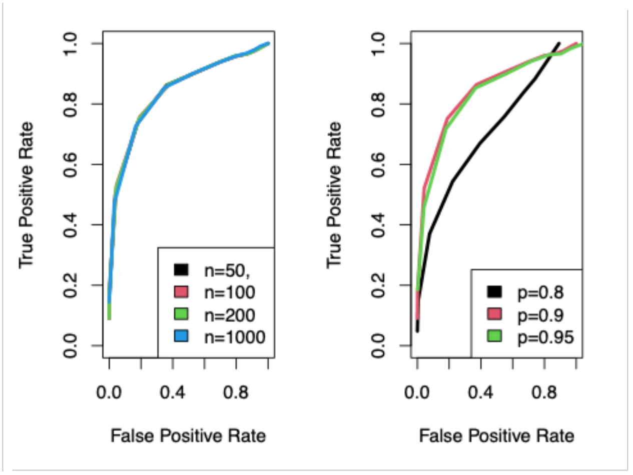
50 times	Classification Rate	Mean(FPR)	Mean(FNR)	S.D.(FPR)	S.D.(FNR)
NW Lasso (joint)	0.9065	0.0092	0.7745	0.0014	0.0186
NW Lasso (or)	0.9045	0.0053	0.8237	0.0011	0.0187
Graphical Lasso	0.8147	0.1737	0.2785	0.0798	0.1067

The table above shows that the classification rates of two Node-wise Lasso approaches are higher than Graphical Lasso Approach, and the mean of false positives and standard error of the false positives are lower, which is also an indication that two Node-wise Lasso approaches perform better with a more stable result. However, the mean of false negative rates for two node-wise lasso models are higher. It may not be good in the real world problems, such as the test of certain type of disease.

2.3.2 Various Simulation Settings (different values of n, p and sparsity structures)

Compared with the ROC of the Node-wise Lasso Approach when using different n value, the Graphical Lasso Approach's ROC seems more stable and similar within the mode. Setting n equals to 1000, which is greatly large than p (100), the area under the Node-wise Lasso's curve is much close to 1. It is coincided with the analysis that the models perform better, when large amount of data is available.





Considering the sparsity structures, the ROC shows the difference between two models. For the Node-wise Lasso, this is a pattern shows that the classification rate will higher when the sparsity level increased from 0.9 to 0.95. For Graphical Lasso, the two settings seem perform very close to each other, which is not the case in the Node-wise Lasso. However, when the sparsity level is 0.8, the two methods give the opposite results.

2.3.3 Strengths and weakness of two methods

One of the main strengths of the node-wise lasso approach is that it is easy to interpret and understand. Nevertheless, the computational efficiency of the node-wise approach is not ideal, costing a lot of time to derive results. In terms of the graphical lasso approach, it is computationally efficient and has a relatively better performance on the ROC curve, however, it does not recover the true edge set quite well.

Appendix

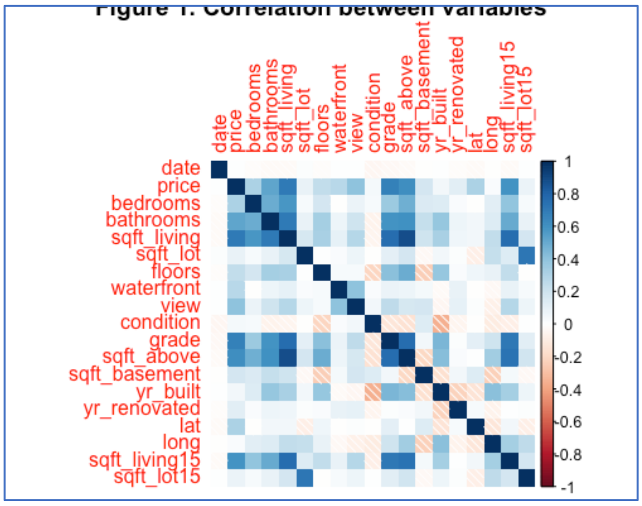


Figure 1 : Correlation Matrix

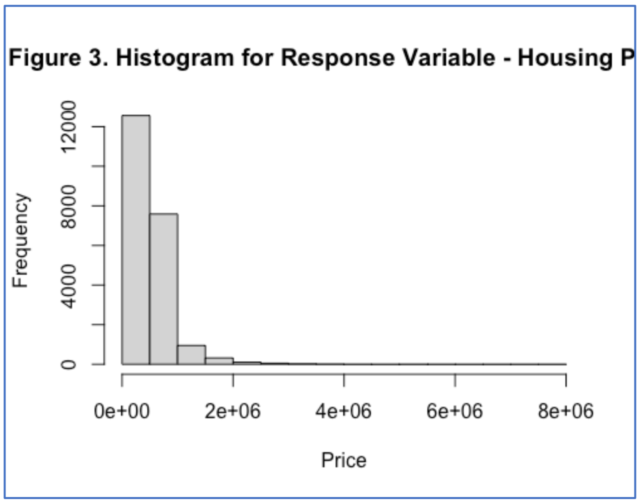


Figure 2: Histogram of Price

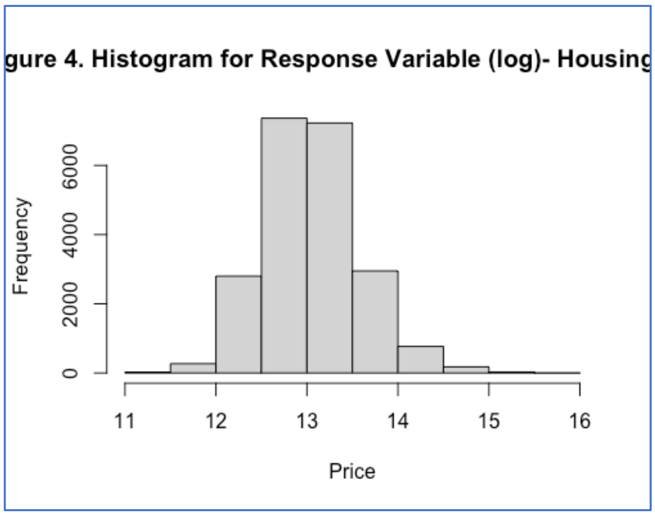


Figure 3: Histogram of Log(Price)

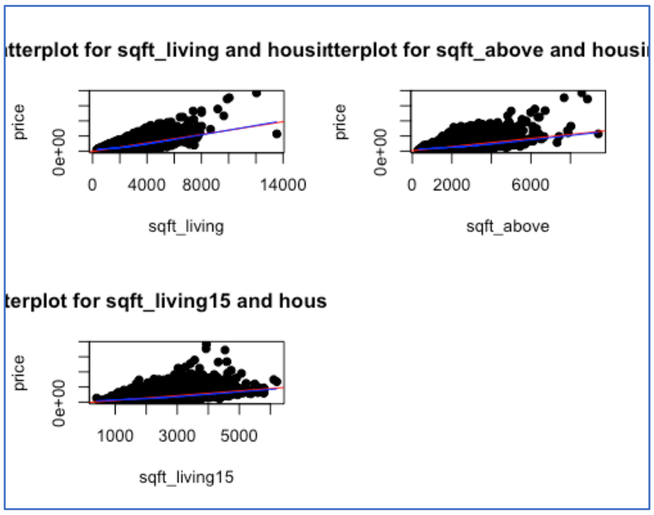


Figure 4 : Linear Relationship

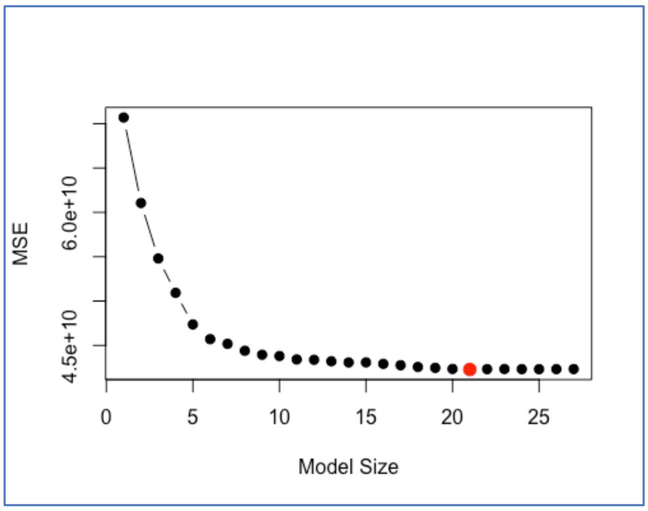


Figure 5: Best Subset Selection

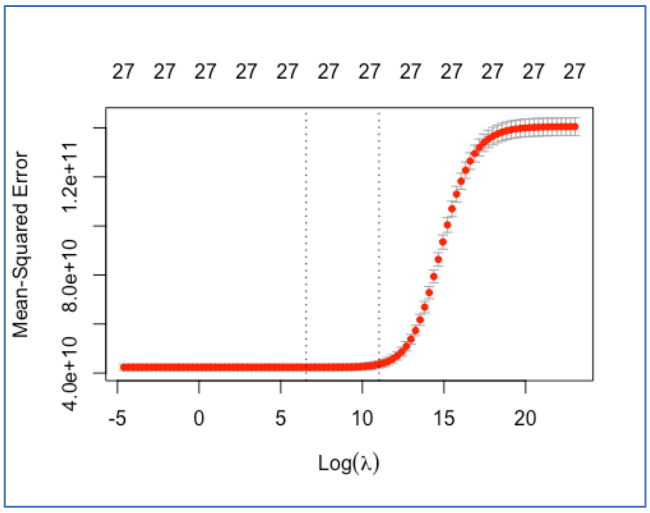


Figure 6: Ridge Regression – Log(λ) & MSE

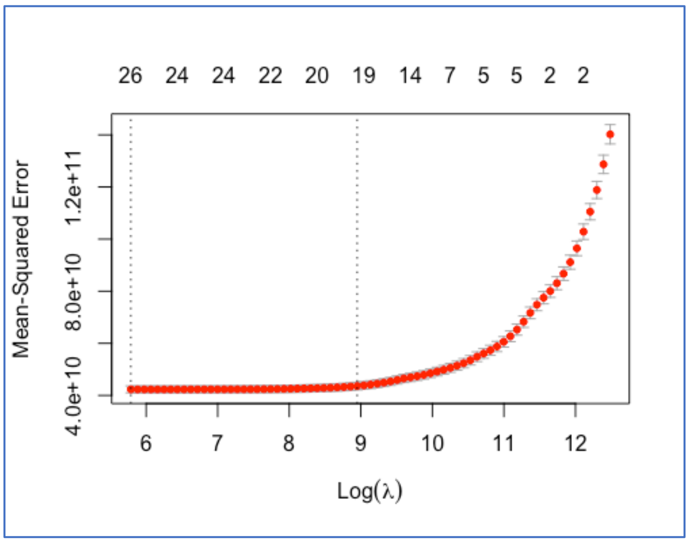


Figure 7 : Lasso Regression

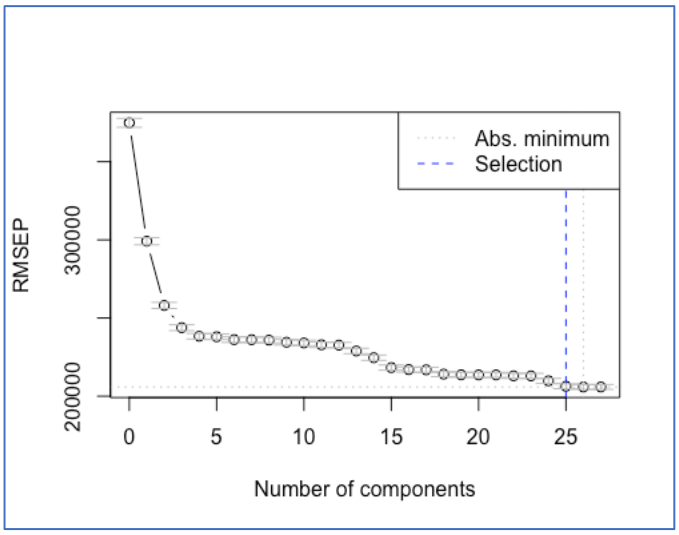


Figure 8: PCR selecting the components

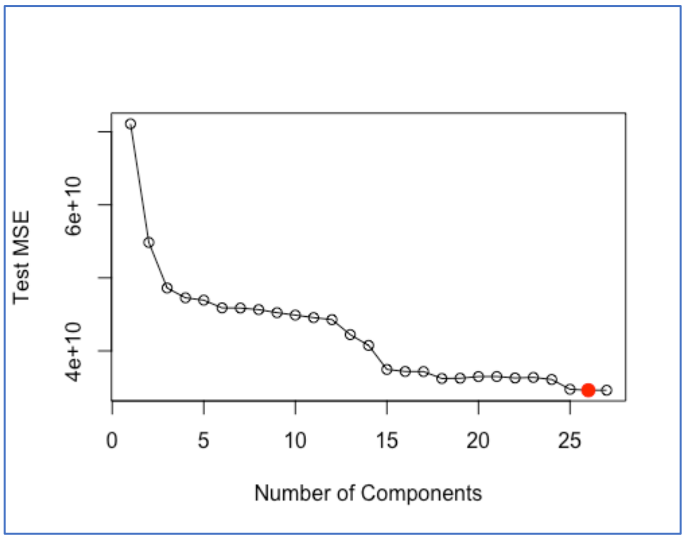


Figure 9: No. of Components in PCR

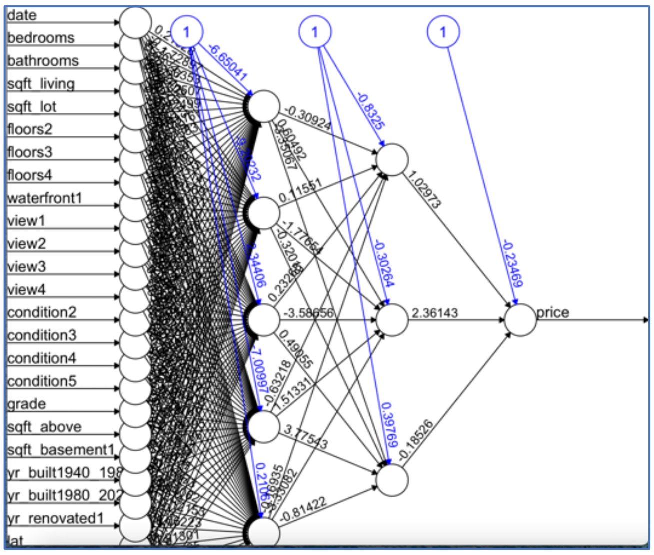


Figure 10: Neural Network

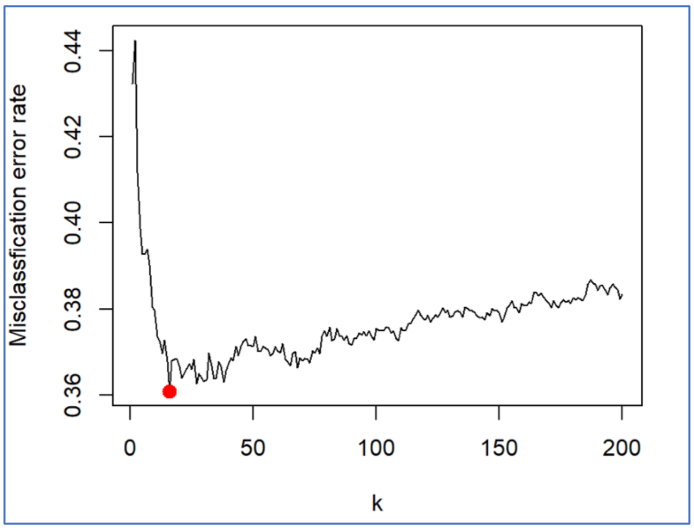


Figure 11: K Nearest Neighbor

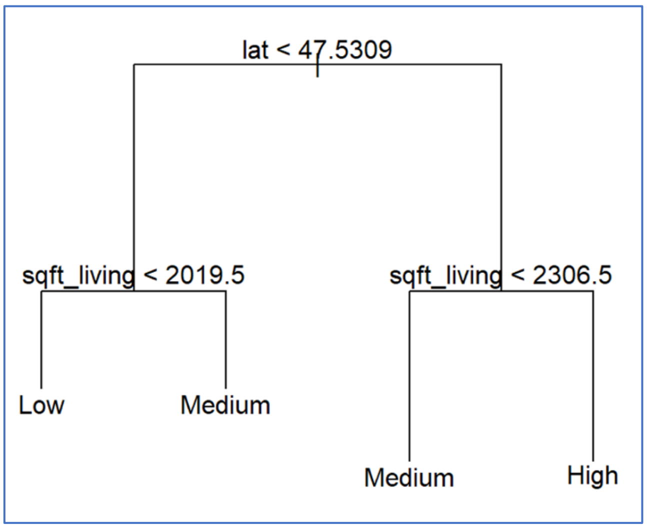


Figure 12: Classification Tree

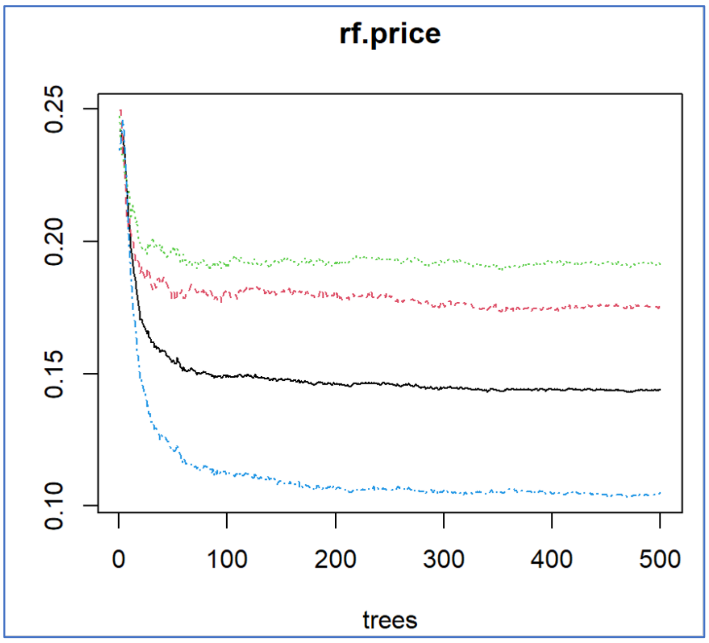


Figure 13: Random Forest

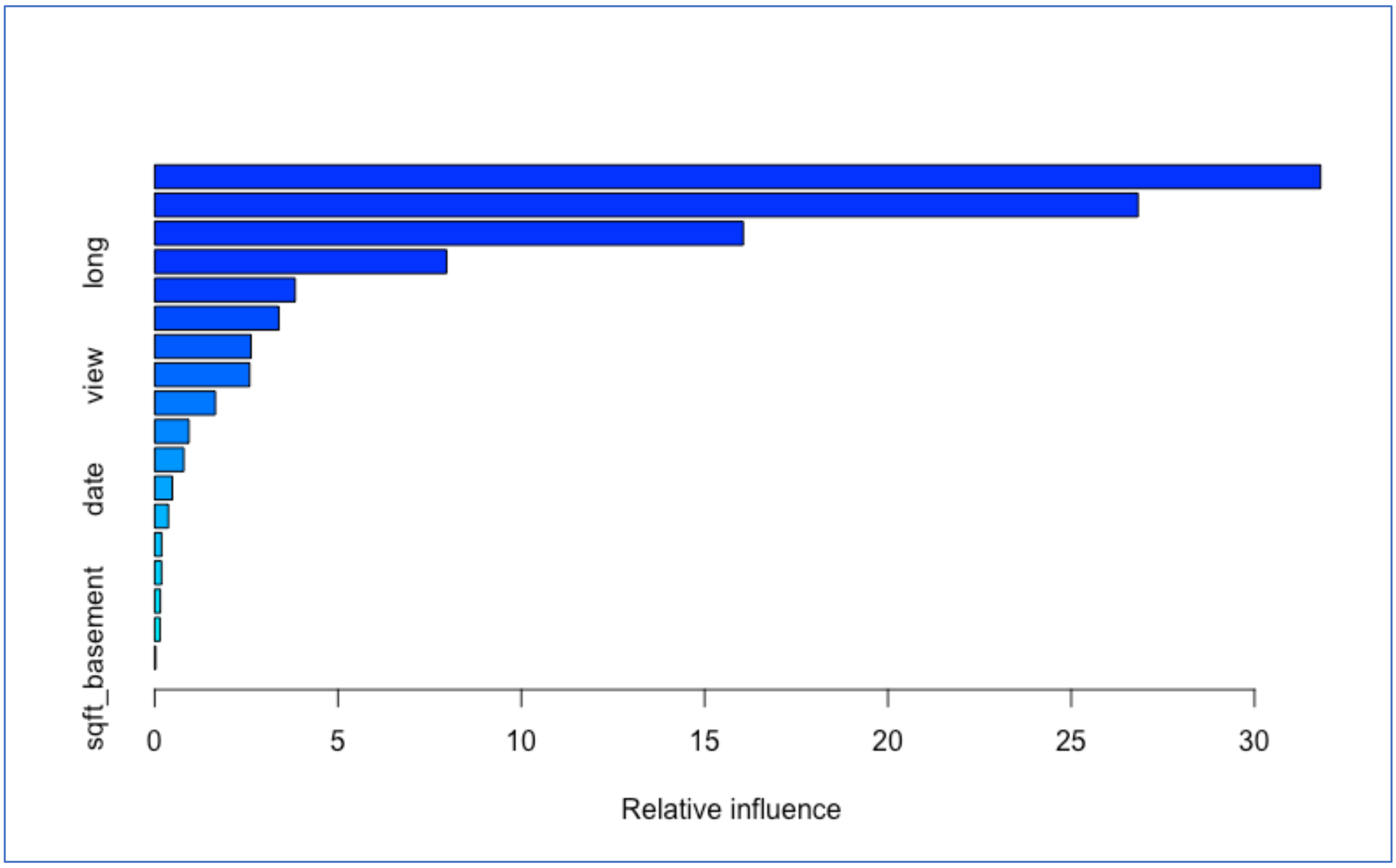


Figure 14: Gradient Boost Importance Plot