******

***ALY6040 – Data Mining Applications***

***2019 Fall CPS/First Half***

***Project Kick-Off and Model Optimization Lectures***

***Kasun Samarasinghe***

***October 8, 2019***

***Team Participants:***

*Nitika Bhatia*

*Pooja Shirke*

*Sheith Sadeddin*

*Sreyluch Seang*

*Zixin Meng*

# **Introduction**

The objective for this week is to perform optimization, tree-based modeling and prediction algorithms on the Ames Housing dataset. The dataset is randomly sampled into training (75%) and test (25%) data. We used generalized linear regression and random foresting to evaluate the relation between sale price and other dependent factors. Upon testing the correlation, we observed a high multicollinearity between dependent variables. For eliminating redundancy in our regression equation, we used the Lasso and Ridge optimization techniques to deal with the variables with high Variable Inflation Factor (more than 10). K-Fold Cross Validation is also used to optimize the random forest model. Using the regression hyper tuning parameters like RMSE, MAE, AIC and BIC, we have selected the best fit model to predict the final sale price.

# **Analysis – Modelling and Optimization**

## **Linear Regression**

The generalized linear regression perfomed on the Ames housing dataset which had an R-squared of 0.9648 suggesting a possibility of overfitting and multicollinearity. To perform the optimization techniques on our models, we first divided the dataset into training and test data using the following code:

#############################################Sampling  
##############################################  
set.seed(101)  
sample <- sample.int(n = nrow(main\_train), size = floor(.75\*nrow(main\_train)), replace = F)  
main\_train <- main\_train[sample, ]  
test <- main\_train[-sample, ]

**Optimization: Dropping variables based on VIF**

We performed VIF and multicollinearity test on the training dataset using imcdiag() function in “mctest” package using the following code:

####################Checking Multicol#################  
X <- main\_train[,c(1:115)]  
Y <- main\_train$logSalePrice  
imcdiag(X,Y)

The results from imcdiag() function provided the VIF , TOL and CVIF for all the numerical factors. MSSubclass, YearBuilt, GrLivArea, Pool Area, Pool Quality, Garage Area and Garage quality were some of the dependent variable that had VIF higher than or close to 10 leading to multicollinearity. Thus, we hyper tuned our glm model by removing different combinations of these variables to check the best fit model. The tuning parameter values used to determine the best fit model are given in Table 1:

*Code used for determining tuning factors:*

lm1 <- c(A,1-A, BIC(mymodel), AIC(mymodel), rsq(mymodel),RMSE(MPred,test$logSalePrice),MAE )  
lm2 <- c(B,1-B, BIC(mymodel1), AIC(mymodel1), rsq(mymodel1),RMSE(M1Pred,test$logSalePrice), MAE1)

**Table 1: Performed Models and parameters used for Regression tuning**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Training Error** | **AIC** | **BIC** | **RSQ** | **RMSE** | **MAE** |
| LM1 | 10.615% | -1594.52 | -1069.67 | 92.913% | 9.985% | 7.160% |
| LM2 | 11.305% | -1468.64 | -973.78 | 91.962% | 10.754% | 7.931% |

In Table 1, the tuning parameters for two of the best fit regression models have been provided. For our final selection, we chose the model with lower AIC, BIC, RMSE and MAE while higher RSQ, that is, LM1.

**Optimization: Regularization using Lasso and Ridge**

We used Lasso and Ridge regression as other optimization techniques to see how well these regularized models tackle the problem of multicollinearity and overfitting. The following code has been used to predict the values and calculate their tuning parameters. The performance of Lasso and Ridge regression is given in Table 2.

##################LASSO  
newx <- model.matrix(logSalePrice~.,data = test)  
lassopredict <- predict(fit, newx = newx)  
RMSELasso <- RMSE(lassopredict,test$logSalePrice)  
  
#####################Ridge  
ridgepredict <- predict(fitR, newx = newx)  
RMSERidge <- RMSE(ridgepredict,test$logSalePrice)  
  
###Performance of Lasso  
Lasso <- c(C,1-C, BICL, AICcL,RMSELasso, MAE(lassopredict,test$logSalePrice))  
Ridge <- c(D,(1-D), BICR, AICcR, RMSERidge, MAE(ridgepredict,test$logSalePrice))

**Table 2: Performance of Lasso and Ridge Regression**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Training Error** | **AIC** | **BIC** | **RMSE** | **MAE** |
| Lasso | 10.89% | -6.815 | 342.792 | 10.49% | 7.54% |
| Ridge | 10.81% | 93.456 | 636.531 | 10.30% | 7.38% |

Ridge Regression technique is observed to perform better than Lasso on this dataset. However, if we compare the parameters of Ridge with the LM1, then the glm model has lower AIC, BIC, RMSE and MAE. This means that LM1 is the best fit regression model to predict the sale price with 7.1% mean absolute error, suggesting a 92.9% accuracy.

**Optimization: K-Fold Cross Validation**

We have used the K-fold cross validation method to determine the average prediction error based on k = 10, which means that the data will be divided into 10 subsets of training and test data and provides us the average prediction errors from all those subsets. Using those parameters as threshold for our linear model, we will identify whether LM1 is the rightly fitted or not. The following code has been used for cross validation:

# Define train control for k fold cross validation  
train\_control <- trainControl(method="cv", number=10)  
# Fit Linear Regression Model  
model <- train(logSalePrice~., data=main\_train, trControl=train\_control, method="lm")

# Summarise Results  
print(model)

The resampling results from our 10 fold cross validation provided that the average RMSE for all the models are 0.123, MAE is 0.087 and RSQ is 0.905. LM1 has an RSQ of 0.929, MAE of 0.0716 and RMSE of 0.099. Based on these parameters, LM1 model is better than resampled model through 10-fold cross validation.

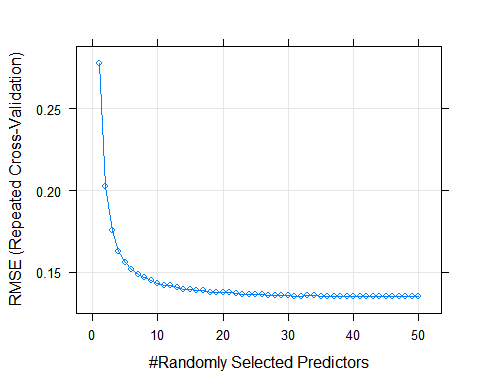
## **Random Foresting**

We used Random Foresting algorithm since our data has a lot of categorical variables. We have used the k-fold cross validation for RF Models as well to determine the best model that will give us the lowest prediction error. The following code has been used for rf cross validation:

#Cross Validation  
control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")  
set.seed(101)  
tunegrid <- expand.grid(.mtry=c(1:50))  
rf\_sale <- train(logSalePrice~., data=main\_train, method="rf", tuneGrid=tunegrid, trControl=control)  
print(rf\_sale)

The results from our model suggested that selecting 48 random predictors for our model will provide us with the best fit model. The figure 1 provides the change in RMSE with change in number of predictors (mtry).

**Figure 1: RMSE vs MTRY**



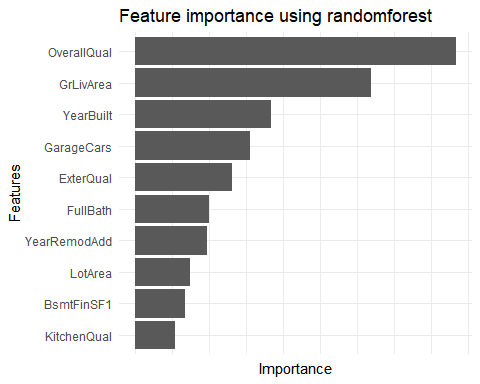
Based on the figure and calculations derived from the code, we see that mtry = 48 gives the best fit model and thus, form RF model based on the results using following code:

#Setting the model  
RFmodel <- randomForest(main\_train$logSalePrice ~. , data = main\_train, method = "anova",   
 ntree = 100,  
 mtry = 48,  
 replace = F,  
 nodesize = 1,  
 importance = T)

We also derived the importance of each feature in RF Model using the following code:

importance\_rfmodel <- data.frame(variable= row.names(imp), importance= round( imp[,"IncNodePurity"],2))  
  
importance\_rfmodel %>% top\_n(importance,n=10) %>%   
 ggplot(aes(reorder(variable,importance),importance)) + geom\_bar(stat= "identity") +   
 xlab("Features") + theme\_minimal(base\_family = "Ubuntu Condensed") + coord\_flip() +  
 theme(axis.text.x = element\_blank()) + ylab("Importance")+  
 ggtitle("Feature importance using randomforest")

**Figure 2: Important Feature in Random Foresting Model**



We predict the values using this RFModel and calculate RMSE and MAE using the following code:

#Predicting the values  
RFmodelpred <- predict(RFmodel, test, type="response",  
 norm.votes=TRUE, predict.all=FALSE, proximity=FALSE, nodes=FALSE)  
  
#Comparison with regression coeffs  
RFRMSE <- RMSE(RFmodelpred,test$logSalePrice)  
RFMAE <- MAE(RFmodelpred,test$logSalePrice)

The RMSE and MAE derived from the model is 0.0491 and 0.0339.

# **Model Comparison – GLM, Lasso, Ridge and Random Forest**

A comparison of the five most significant variables for each of the model is given in Table 4.

**Table 4: Comparison of five significant variable for the models**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **LM1** | **LM2** | **Lasso** | **Ridge** | **Random Forest** |
| Overall Quality | Overall Quality | Electrical Mix | Central Air | Overall Quality |
| Overall Condition | Overall Condition | Heating - Grav | Kitchen Abv Gr | Gr Liv Area |
| Lot Area | BsmtUnfSF | Neighbourhood - Stonebr | Foundation - Wood | Year Built |
| Gr Living Area | Total Rooms Abv Grd | Building Type - Townhouse | Building Type - Townhouse | Garage Cars |
| BsmtFinSF1 | BsmtFinSF1 | Neighbourhood - Crawfor | Neighbourhood - Stonebr | Full Bath |

After running all the models and predicting them on the test dataset we have summaries the tuning parameters, accuracy and time run of all the models in Table 3.

**Table 3: Model Comparison Parameters**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Training Error** | **RMSE** | **MAE** | **Accuracy** | **Time Run (in Seconds)** |
| **LM1** | 10.615% | 9.985% | 7.160% | 92.840% | 0.08 |
| **LM2** | 11.305% | 10.754% | 7.931% | 92.069% | 0.13 |
| **Lasso** | 10.893% | 10.493% | 7.540% | 92.460% | 6.34 |
| **Ridge** | 10.810% | 10.298% | 7.382% | 92.618% | 5.48 |
| **Random Forest** | 3.487% | 4.915% | 3.391% | 96.609% | 7.71 |

In table 3, we see that Random Forest model has the highest run time, however it provides with the lowest training error, RMSE and MAE of 3.48%, 4.91% and 3.39% respectively. Thus, **Random Foresting is the best fit model for house sale predictions.**

## **Pros and Cons Analysis**

**Linear Regression**

Linear regression is an easy and intuitive method to use and understand. Even when it doesn’t fit the data exactly, we can use it to find the nature of the relationship between the two variables. The model run time was actually not even one second. It is really fast and swift in building relationships. However, the major drawback is the it cannot incorporate categorical variables, we had to change a lot of variables into numeric class to run the model. Also, it is very sensitive to the anomalies in the data.

**Lasso Regression**

Lasso regression is better than the usual methods of automatic variable selection such as forward, backward and stepwise all of which can be shown to give wrong results.

However, in our case, it did not improve the output and showed higher AIC, RMSE and MAE than glm model.

**Ridge Regression**

Ridge regression adds just enough bias to make the estimates reasonably reliable approximations to true population values. However, it does not perform feature selection. Ridge regression shrinks the coefficients towards zero, but it will not set any of them exactly to zero. Also, ridge regression took more time to run than Linear Regression

**Random Forest**

The predictive performance can compete with the best supervised learning algorithms. This is also proved by our model analysis, as Random forest modelling gave us the highest accuracy of 96%. However, training a large number of deep trees can have high computational costs (but can be parallelized) and use a lot of memory. Predictions are slower, which created challenges for applications that require low latency. This model had the highest running time than all the others.

# **References**

1. What are the advantages and disadvantages of linear regression? (n.d.). Retrieved from: https://www.quora.com/What-are-the-advantages-and-disadvantages-of-linear-regression
2. Ridge Regression and Its Application. (2018, November 5). Retrieved from: https://stepupanalytics.com/ridge-regression-and-its-application/
3. Hands-On Machine Learning for Algorithmic Trading. (n.d.). Retrieved from: https://learning.oreilly.com/library/view/hands-on-machine-learning/9781789346411/e17de38e-

421e-4577-afc3-efdd4e02a468.xhtml