Project number: 23

Analytics: Assignment 1

Honours in Statistics, 2019

Department of Statistical Sciences, University of Cape Town

By Chongo Nkalamo & Yovna Junglee

Date: 14 March 2019

Contents

A comparative analysis of statistical learning models to forecast house prices using Ames, Iowa data

Hyperparameter tuning using h2o package in R

Appendix

A comparative analysis of statistical learning models to forecast house prices using Ames, Iowa data

March 14, 2019

Abstract

This report aims to evaluate the performance of several statistical learning methods using Ames dataset[1] to predict sale prices of houses. The methods included linear models, regression trees and ensemble techniques such as random forests, bagging and boosting. Out of bag sampling and K-fold cross validation were used for model validation while the mean square error was used as a performance metric. By comparing the cross validation, OOB and test set error, it was found that boosting was the best model to forecast the sale price of houses.

Keywords: Random Forest, Bagging, Boosting, Regression Tree, Linear Models, Mean Square Error

[1] De cock,2011

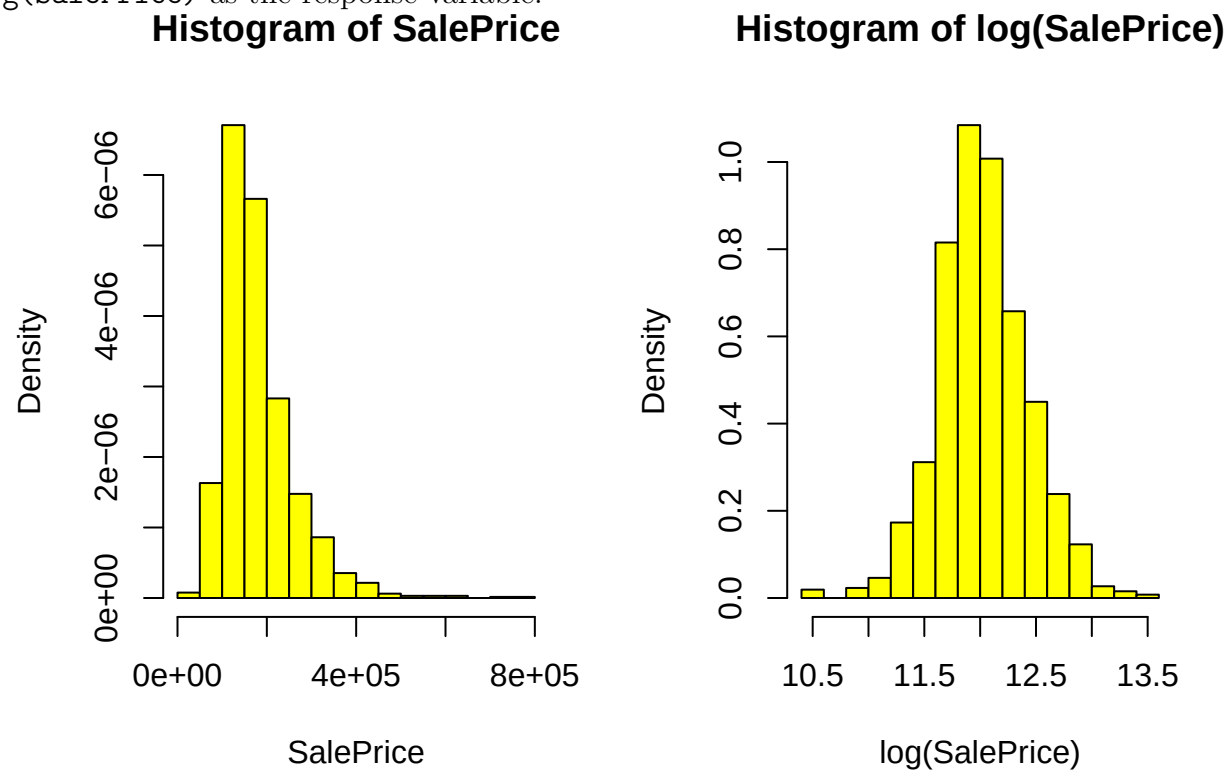
1 Introduction

1.1 Data

The data relates to the property sales and prices that occurred in Ames, Iowa between 2006 and 2010. The data contains 79 explanatory variables which describe the attributes, quality and condition of the houses sold. There are 36 continuous and 43 categorical variables.

1.1.1 Data wrangling.

As seen in the histogram plot of SalePrice below, we notice that the data is slightly skewed to the right. We consider a log transformation which would hopely give us a histogram that is approximately normal and hence allow us to model under normal distribution assumptions. Indeed we see this on the plot to the right, the data is approximately mean around log SalePrice of \$12. Therefore, all models throughout the rest of this paper consider `log(SalePrice)` as the response variable.



We also make note of a number of missing values in our dataset across various predictors. An analysis of this concluded that for majority of the predictors, these were not missing

values but instead meant that a certain attribute was not present for that particular house. For example, an NA (missing value) under Garage meant that the particular observation (house) did not actually have a garage. After all this data cleaning was done, we proceed to carry out exploratory data analysis were we attempt to only consider the most appropriate variables that best explain our response.

1.2 Exploratory Data Analysis

1.2.1 Continuous variables selection

Separation of continous variables from factor variables was done and by means of correlation plots, continous explanatory variables were analysed and the most significant were selected to contribute to the model building phase. Cohen[2] defines a small correlation as having an absolute value of approximately 0.1, a medium correlation value of having 0.3 and large correlation value being 0.5 or more. Hence, a correlation coefficient of $\rho > 0.5$ was selected as an optimal threshold.

Figure 1 shows only variables that are highly correlated ($\rho > 0.5$) to our response variable sale prices. We further make note of multicollinearity between explanatory variables. This is done because if a variable is collinear, then question as to whether the variable is redundant or not arises. Here again we choose a different threshold of $\rho > 0.8$ to assess for multicollinearity. We see that *GarageCars* and *GarageArea* have a high correlation value of , hence explain the same variation with the response. Thus only *GarageArea* was kept for further analysis as it has a higher correlation with our response. Similarly the same was noted between *TotalBsmtSF* and *1stFlrSF*, the former was chosen. *GarageArea* and *TotalBsmtSF* as well as all the other variables shown in our correlation plot were considered as significant variables.

1.2.2 Categorical Variables

Boxplot and corresponding average values of sales price across each individual factor variable was used to select the most significant factor variables. In our selection, we only

[2]Cohen, 1988

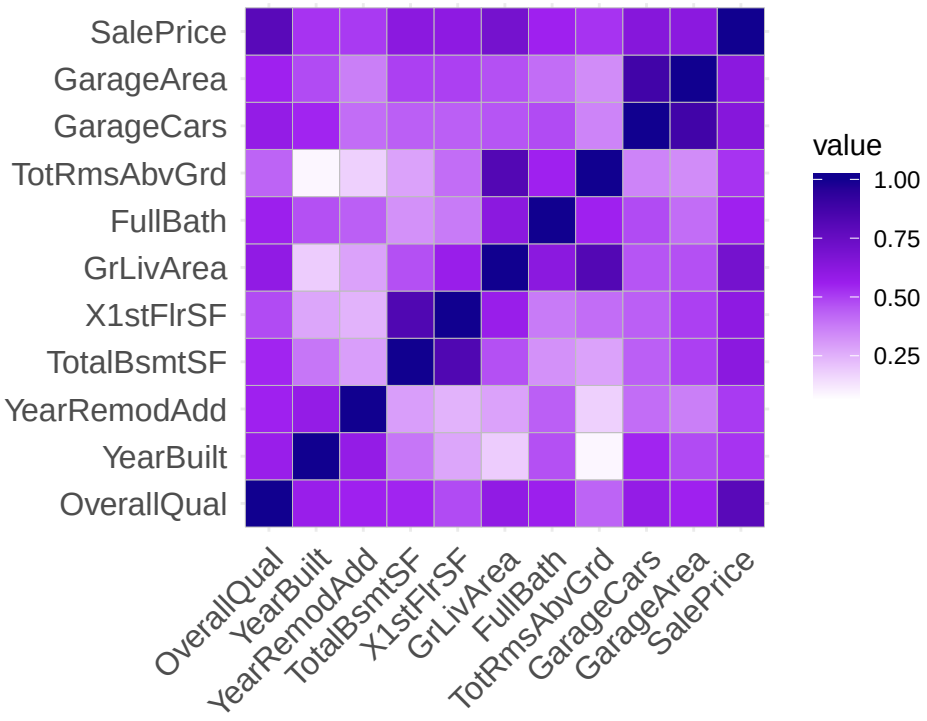


Figure 1: Correlation plot

considered a factor variable that had different mean price across each of its levels as significant. The boxplots for the factor variables can be found in the appendix.

After filtering out all the significant continous and factor explanatory variables, we were able to reduce our explanatory variables from 79 variables to 22 variables we felt explain our response well. We then proceed to build models on this new dataset.

1.2.3 Outliers

Outliers are aberrant scores that lie outside the usual range of scores we would expect for a particular variable.[3]. De Cock[4] outlines that observations with $GrLivArea \geq 4000$ should not be taken into consideration during the analysis as these are outlying. Figure 2 confirms that such observations are outliers. These observations were removed as they could affect our analysis.

[3]Miles and Shevlin, 2001

[4]De cock, 2011

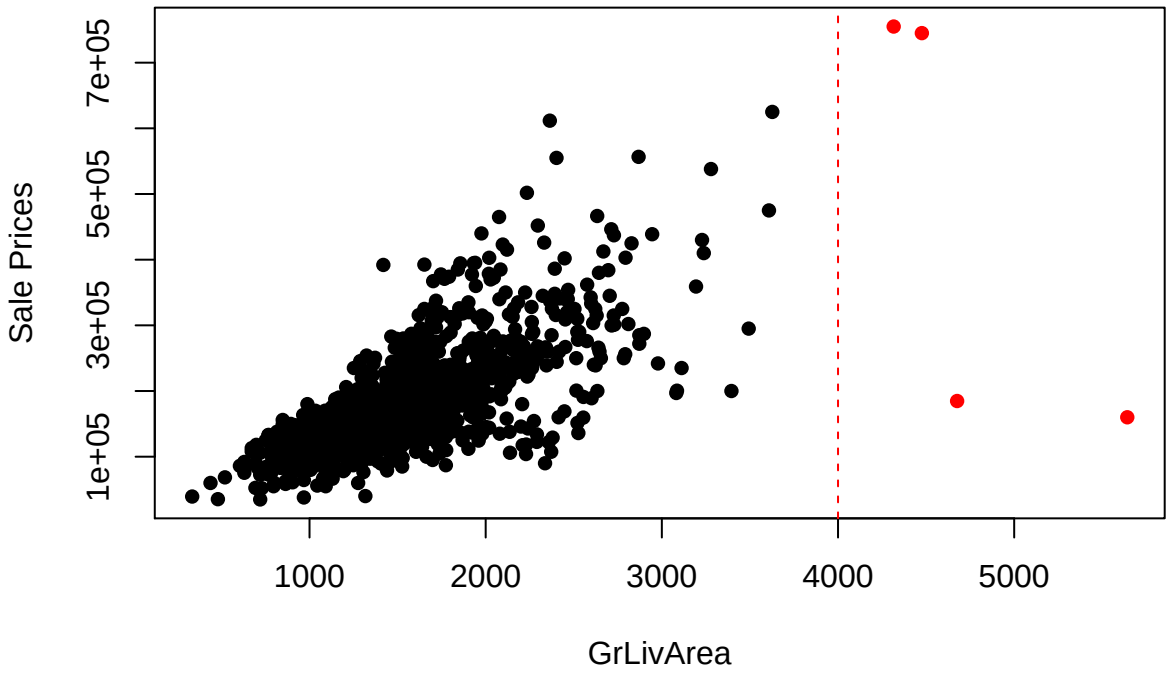


Figure 2: Observed

2 Methods

2.1 Regression Tree

This is the first tree-based method we consider. Before carrying out the model building, we split our new data into a training and test set.

To build a regression tree, a *recursive binary splitting* approach is taken as the algorithm successively splits the tree using the predictor that best minimizes the residual sum of squares.[5]. Using the `tree` function in R, we build a regression tree on our training set and the tree obtained is shown in the appendix. We get a tree that has 10 terminal nodes and variables OverallQual, GrLivArea, CentralAir, GarageFinish and TotalBsmtSF as significantly important variables. Interpretation of this shows that Overall Quality was viewed as the most significant variable that explains the sales price.

Next, we have to consider the possibility of the model only performing well on the training set only. That is, we must avoid overfitting. Therefore, we consider cost complexity pruning which enables us to select a subtree that is able to overally perform well in-

[5]James,Witten et al, 2013.(Pg 306)

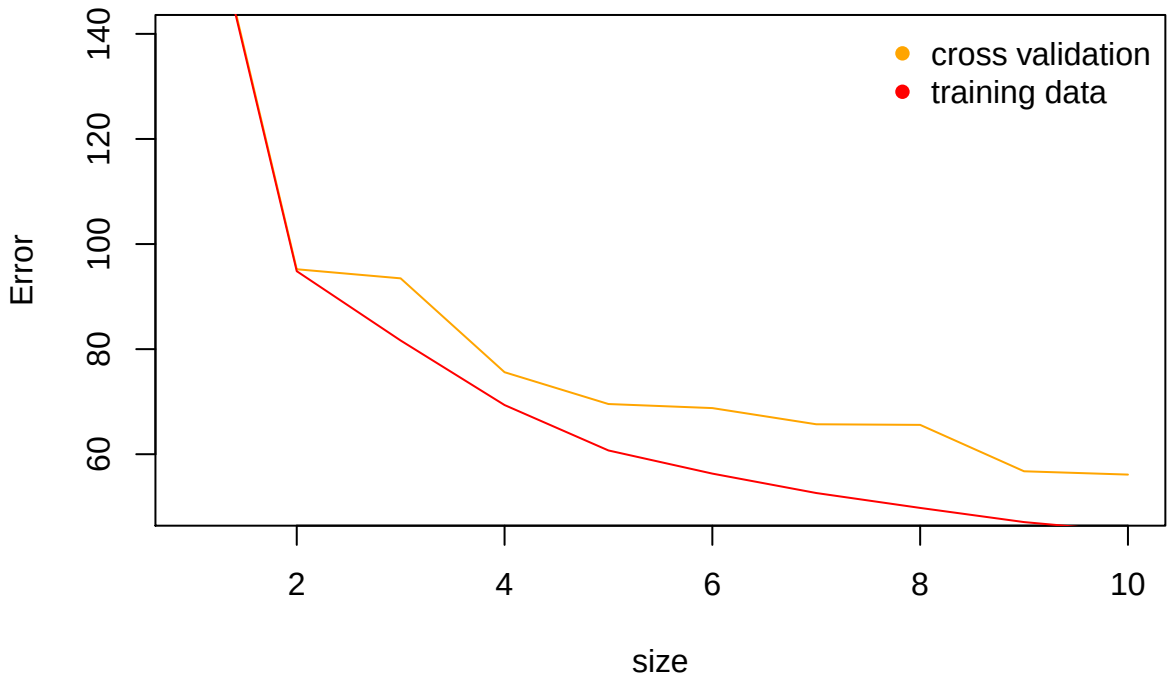


Figure 3: Regression Tree error plot

sample (training) and most importantly predict our out-of-sample data (test). An optimal tuning/cost complexity parameter that corresponds to a particular terminal node level has to be selected. Carrying out the cost complexity pruning, we obtain the following tuning parameters (alpha) and corresponding terminal nodes.

To select an optimal level of alpha/cost complexity parameter, we carry out 10-fold cross validation. Below a plot of the cross validation error (cv) vs deviance-size for the training model is shown. It is worth noting that at terminal node size 5, the model almost explains the same relationship as a model that has more terminal nodes. From this, we select the corresponding cost complexity parameter. This would be the cost complexity parameter corresponding to terminal node of size 5, this is 4.402915.

Refitting of the model was done but this time taking in to account the cost complexity value of 4.402915 which will prune the full tree to a subtree with 5 terminal nodes. Hence, we settle with this model as our final model.

Figure 4 shows our new regression tree.

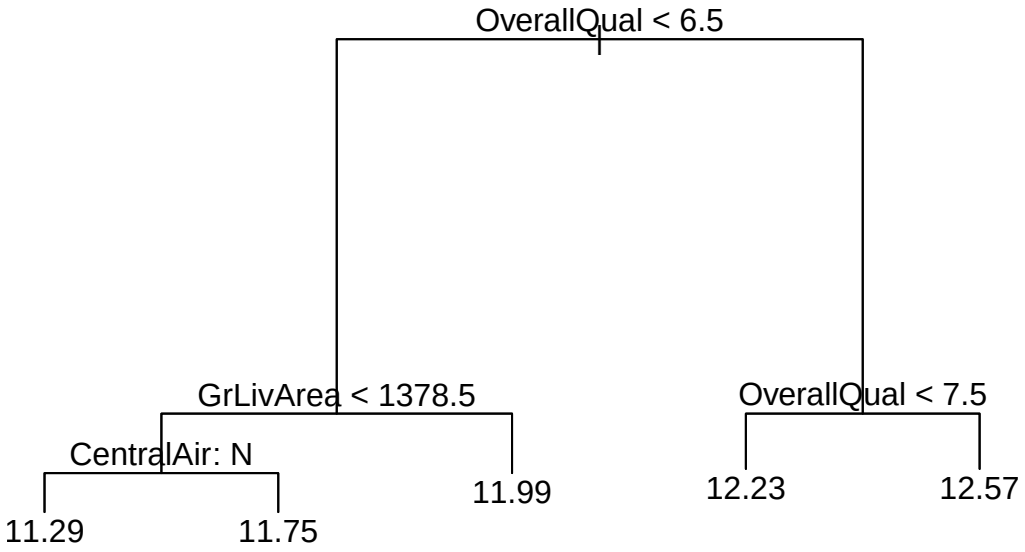


Figure 4: Final regression tree

2.2 Bagging

Bagging is a procedure that mainly helps reduce high variance by bootstrapping the training data set, thus building a tree on the bootstrap samples and averaging the prediction values.[6]. On average, one third of the sample will not be used during bootstrapping and these observations are considered *out-of-bag* (OOB). The OOB mean square error (MSE) is then used to compare various models. The building procedure as describe was carried out and Figure 5 shows the OOB mean square error when bagging ($m = p$) regression trees on the Ames dataset using the 22 predictor variables selected in the subsection exploratory data analysis. It can be seen that the MSE's are decreasing and stabilize approximately around 175 trees.

2.3 Random Forest

Bagging provides a way to reduce sampling variability. However, if some of the trees obtained are still highly correlated, this improvement will not be substantial.

Random Forest is a procedure that allows us to overcome this problem of high correlation between trees by selecting a subset of predictor variables, $m < p$, before growing each tree,

[6]James,Witten et al, 2013.(Pg 316)

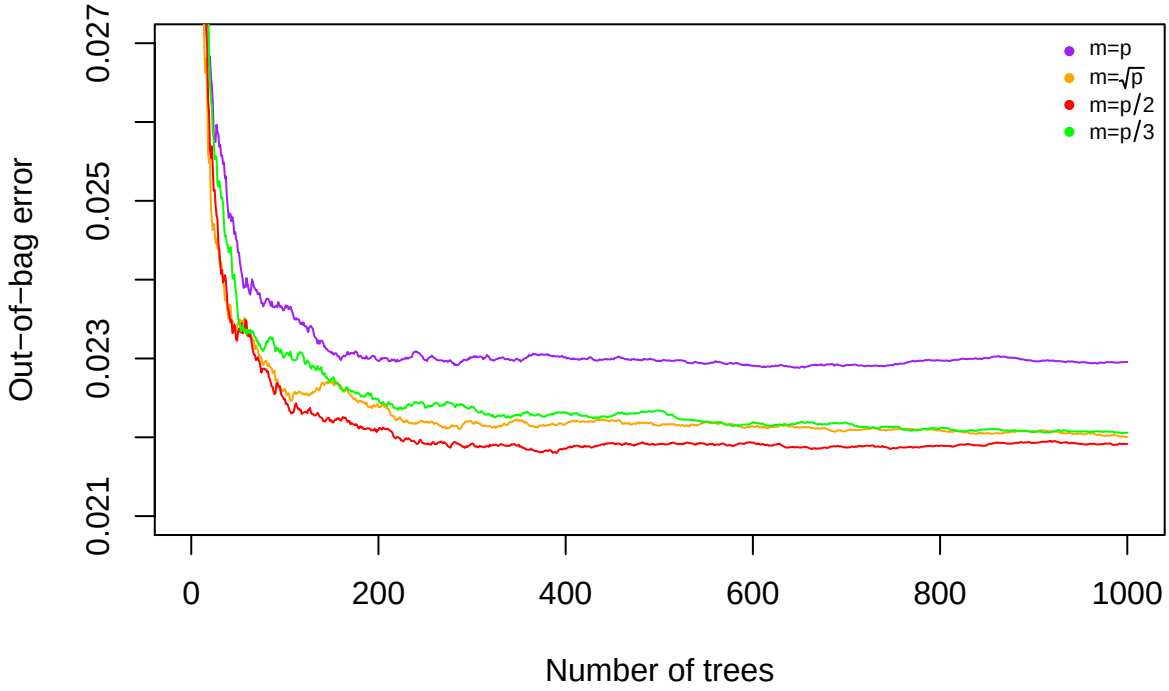


Figure 5: Bagging and random forest error plot

randomly and thereby reducing the variance and decorrelating the trees.[7] The model was fitted using $m = p/2$, $m = p/3$ and $m = \sqrt{p}$. In Figure 5, it can be shown that the random forests have the lower out-of-bag error and performs better than bagging ($m = p$).

2.4 Boosting

Boosting is a method that fits the tree to the residuals instead of the response variable. A *shrinkage parameter*, λ , also known as the *learning rate* helps to control the rate at which the algorithm learns, while the *interaction depth*, d paramater controls the number of splits in each tree.[8] The number of trees $B = 1000$, $\lambda = 0.01$ and $d \in (1, 2, 3)$ were chosen to fit and compare various models. 10-fold cross validation was used to assess error and from this, cross validation errors were obtained. Figure 6 gives for each depth parameter $d \in (1, 2, 3)$ various cross validation graphs. It is worth noting that around tree size 500, the cross validation error tend to stabilise for depth levels 2 and 3, but tree with $d = 3$ gives a generally smaller cross validation error relatively to the others.

[7] James,Witten et al, 2013.(Pg 319-320)

[8] James, Witten et al, 2013 (Pg 321)

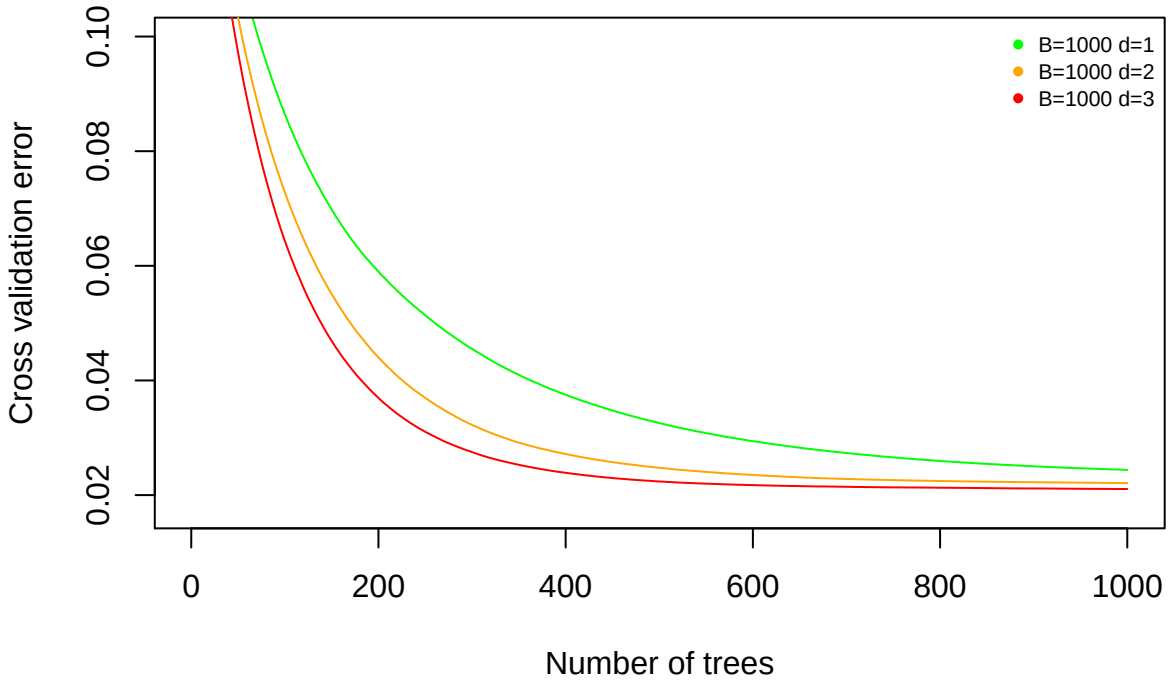


Figure 6: Boosting error plot

2.5 Linear regression

The final method considered to aid prediction of `SalePrice` was linear regression. The motivation for this is that as set out in the introduction, our response variable is measured on a contionous scale. Furthermore, `log(SalePrice)` is approximately normally distributed around its mean. The linear regression procedure was done under a generalized linear model (`glm`) framework. Here, we make use of the gaussian/normal family as the proposed distribution family and select the identity link function as the function that will give the relationship between the linear predictor and the mean of the response variable.

A full model with all 22 explanatory variables (`model 1`) was fit and a summary of the output was obtained. Not all 22 explanatory variales gave a significant relationship with the response. To measure this relationship, the `p-values` were used as a metric and only variables with very small `p-values` (to about a 5% significance level) were considered significant. This was our `model 2` and methods such as stepwise regression was used to choose various other models. These models can be found in the appendix.

Cross validation was carried out on the various models described above. This is the

Table 1: Cross validation error for linear models

Model	model 1	model 2	model 3	model 4	model 5
cv.error	0.1620859	0.0195533	0.01924226	0.019688	0.01971477

procedure we adopt to compare the different linear models and select the linear model that has the lowest cross validation (cv-error). We choose the lowest cv-error as it gives an approximate low test error if we were to incorporate the chosen model on the test set. The table below shows the different cv errors obtained for the four different linear models:

As seen from the above table, the linear `model 3` gave the lowest cv error of 0.01924226. Therefore, this model was the prefered linear model and model diagnostics are carried out on the model to ensure that linear regression assumptions hold.

2.5.1 Model checking

To ensure our `model 3` is a valid model, we carry out model diagnostics. That is, ascertaining as to whether regression assumptions hold. As can be seen from the plots in the appendix, seen in the normal q-q plot, the plot gives almost an approximate straight line with no obvious patterns. We can infer from this that the errors are normally distributed. Next, we look to see whether the residuals have the same variance. We go to the residuals vs fitted plot and hope to see a random scatterplot around the mean of the residuals which is 0. Indeed as see in the appendix, we do see this from the residual vs fitted plot and hence infer that the spread of the residuals is approximately the same over the whole range. The mean of the errors are also centered around zero and we can therefore conclude that the regression assumptions hold.

3 Results

3.1 Model Selection

We first seek to compare random forest and bagging models as they are somewhat similar and in comparison between random forest and bagging, we note that the random forest model performs better. This was shown in Figure 5.

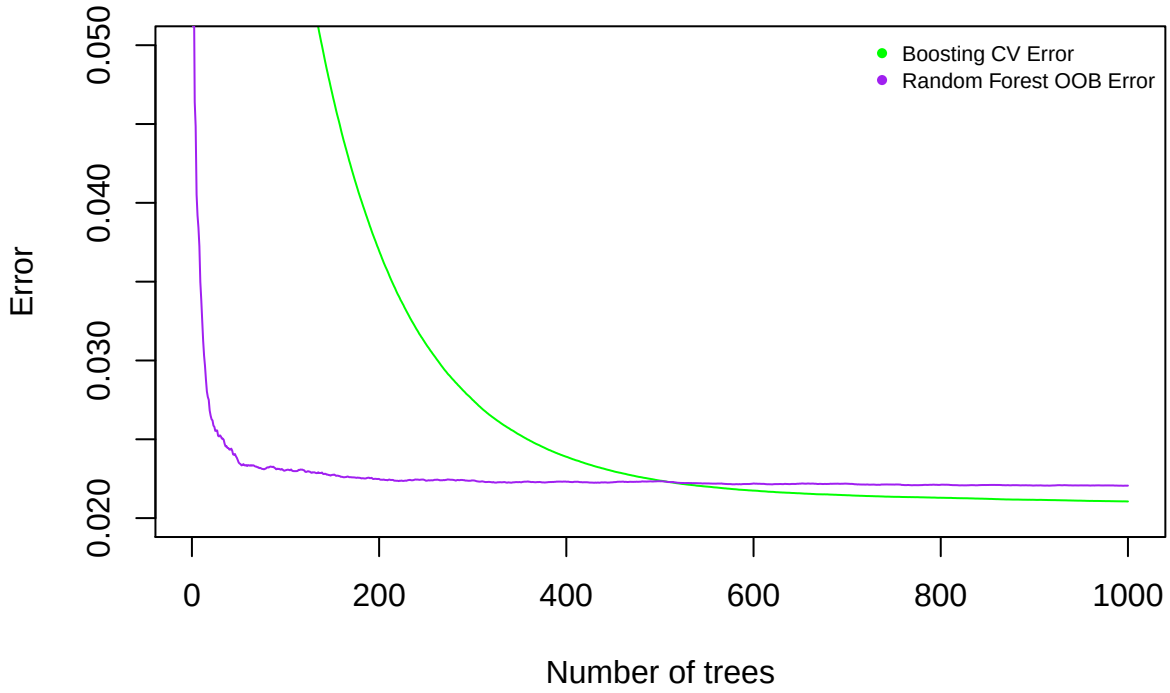


Figure 7: Bagging and Boosting error plot

Table 2: Compare mean square error values on test set

RegressionTree	Boosting	RandomForest
0.0406	0.0134	0.0135

Using our selected random forest ($m = p/3$) model, we compare this with the boosting model($B = 1000$ and $d = 3$) and we use the cross-validation(cv) and out-of-bag (OOB) errors as performance metrics each time taking the lowest. Figure 7 shows that the errors from the random forest and boosting become approximately the same as the number of trees of grows. From Table 1, boosting has the lowest mean square error than random forest and regression tree on the *test data set*.

```
Using 1000 trees...
```

3.2 Model Assessment

The final tree based model selected that predicts the Sale Price most accurately given the predictors considered is the *boosting regression model* using 1000 trees and interaction

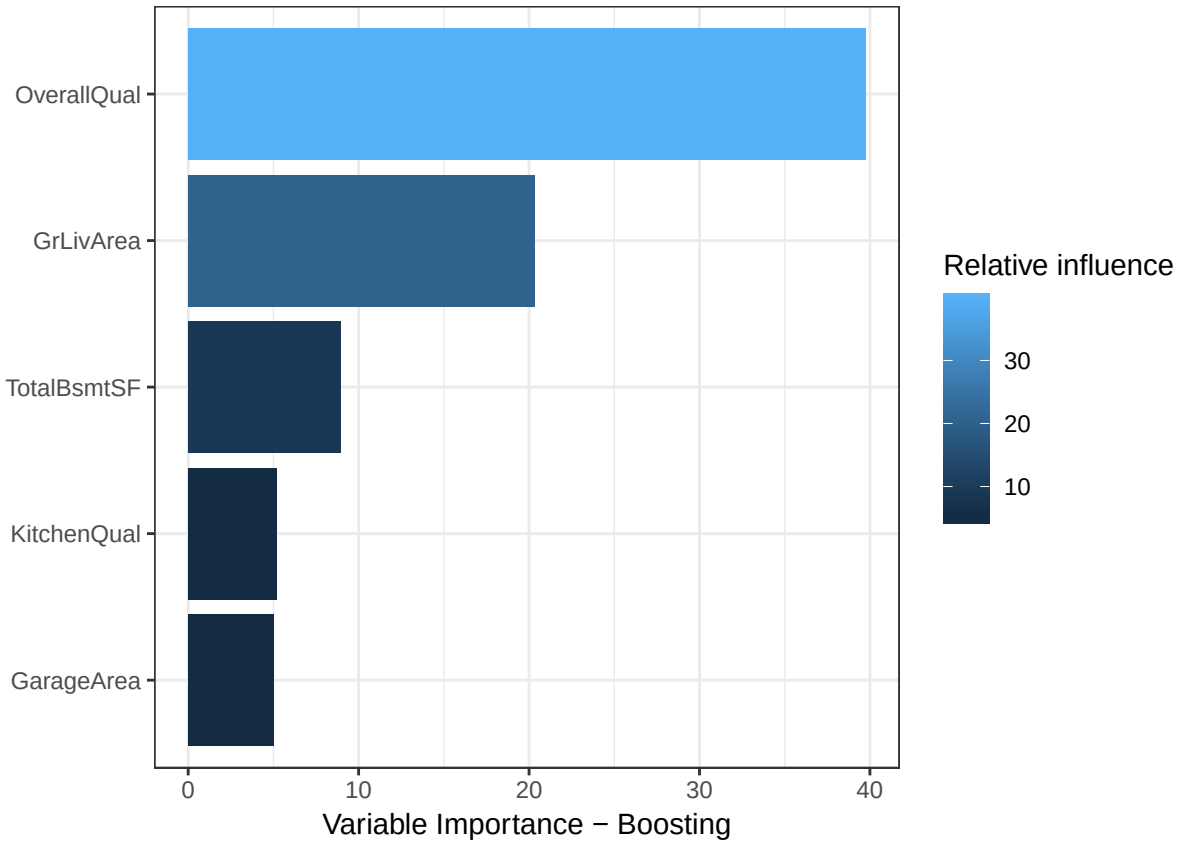


Figure 8: Variable Importance Plot of Boosting

depth = 3 with shrinkage parameter 0.01.

Figure 8 shows the 5 most important variables in the model with *OverallQual* being the most influential one. Relative influence is evaluated as the overall improvement in *mean square error* or *node purity* at each split in the tree.

3.3 Boosting vs Linear regression model

Lastly, we seek to compare the performance of the linear regression model to the prefered tree method selected above. We consider using the test set mean square error (MSE) and prediction accuracy on the test set when the models are fit to the test data. We make refernce to table 1 which indicates an MSE on test set value of 0.0134 for boosting. The MSE produced from the linear model was 0.0163. Drawing from the two resuts, boosting would be considered the more prefered model to carry out the task at hand as it presents a lower MSE. A possible reason for this would be that the boosting method is able to find

more complex dependencies in the data.

4 Conclusion

Three tree based methods and the traditional linear regression method were used to model Sale price of houses in Iowa. From the three tree based methods used, it was seen that the boosting regression tree method performed better than the random forest and bagging methods. Furthermore, the boosting model obtained was compared with a linear model that was built on the same data. Again here it was seen that the boosting model out performed the linear model and the mean square error was used as a metric to determine this. All in all, it can be concluded that a boosting model predicts housing prices more accurately than the other three models considered.

5 References

• Miles.J and Shevlin. M. (2001). Applying regression & correlation. • Frees.Edward W. (2010). Regression modeling with actuarial and financial applications. • James. G, Witten. D, Hastie.T and Tibshirani.R. (2013). An introduction to statistical leanring with applications to R. • De cock. D.(2011). Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project.

Hyperparameter tuning using *h2o* package in R

March 14, 2019

Abstract

This report aims to assess the performance of random forests and boosting by tuning the hyperparameters. An analysis will be carried out to explain the effects of hyperparameters on the algorithms. Performance metrics such as logarithmic loss and deviance on the out-of-bag samples or cross validation samples will be used to compare the models. Confusion matrices, sensitivity and specificity analyses will be carried out to select and compare the best models from the random forest and boosting models.

Keywords: Random Forest, Boosting, Logarithmic loss, Deviance, Sensitivity, Specificity

1 Introduction

1.1 Hyperparameter tuning

Algorithms such as random forest and boosting have different hyperparameters that can be set and tuned in order to optimize performance. Performance measures such as logarithmic loss and prediction accuracy are appropriate for classification problems. Models incorporating the two methods were built using the `h2o` package in R and the performance of each was measured. The `h2o` package offers parallelized implementations of many supervised and unsupervised machine learning algorithms.[1]

1.2 Data

The problem concerns classifying blocks of the page layout of a document that have been detected by a segmentation process. Therefore, the response scale is categorical. This classification problem aims to identify the *class* category of blocks of a page in a document using the following 10 predictor variables: height, length, area, eccen, p_black, p_and, mean_tr, blackpix, blackand, wb_trans.

No form of data wranggling was required as the data was mostly clean although it can be seen in Fig (1) that the proportion of the observations in the response category are unbalanced. Hence, if we consider the data in its current form, *Prediction accuracy* may not be a good performance measure as classes such as `class 3` may not even appear in the test set. Hence, we will mainly focus on *logarithmic loss*(log loss). For a more accurate representation of the original dataset, 80% of observations from each *class* category will be used to train the model while the remaining 20% will be used as the test set. This ensures that we have all labels or classes present in both the training and test set.

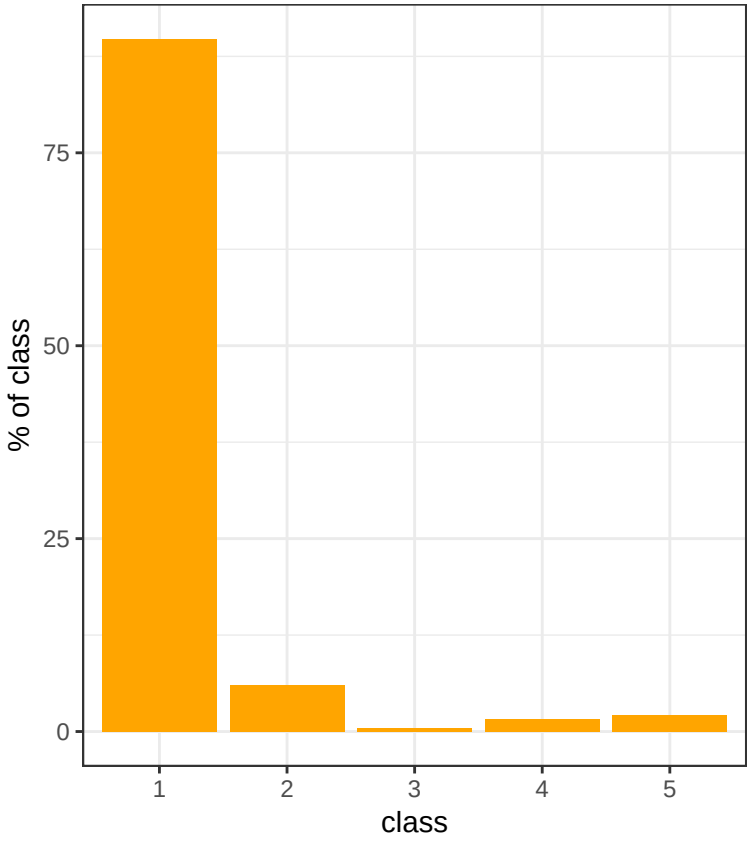


Figure 1: Distribution

Table 1: Summary of hyperparameters in random forest

Hyperparameter	Description	Values
mtries	Number of randomly selected predictor variables	1, 3, 5, 7, 9
ntrees	Number of trees grown	500, 1000, 1500, 2000
max_depth	Depth of tree from root to terminal nodes	1, 4, 8

2 Random Forest

2.1 Influence of hyperparameters in Random Forest

The hyperparameters tuned were: *mtries*, *ntrees* and *max_depth*. Table 1 shows the values used for each hyperparameter.

Figure 2 shows that there is no significant difference across the different number of trees *ntrees*. However, when *max depth* increases, the out-of-bag (OOB) log loss decreases which indicates an improvement of the performance the more the depth is increased. Across the different depths, we see that the log loss decreases as *mtries* increases to approximately 7 but the performance decreases for *mtries* greater than 7 as seen by an increase in OOB log loss in the graph with max depth 8.

2.2 Model building and selection for random forest.

Model building was carried out for each of the different combinations of the hyperparameters. After the model building was done for each of the different combinations of hyperparameters, the log loss plot for each of the models is plotted below. We see that a random forest model with hyperparameters ntrees = 1500, mtries = 5 and max_depth = 8 performs well as seen from a low log loss but equally a random forest model with hyperparmeters ntrees = 2000, mtries = 5 and max_depth = 8 also does fairly well. However, if we look at the test log loss a random forest model with hyperparameters ntrees = 500, mtries = 5 and max_depth = 8 is seen to be competing with a random forest model with hyperparameters ntrees = 2000, mtries = 5 and max_depth = 8 both giving low test log loss.

[1] `https://cran.r-project.org/web/packages/h2o/h2o.pdf`

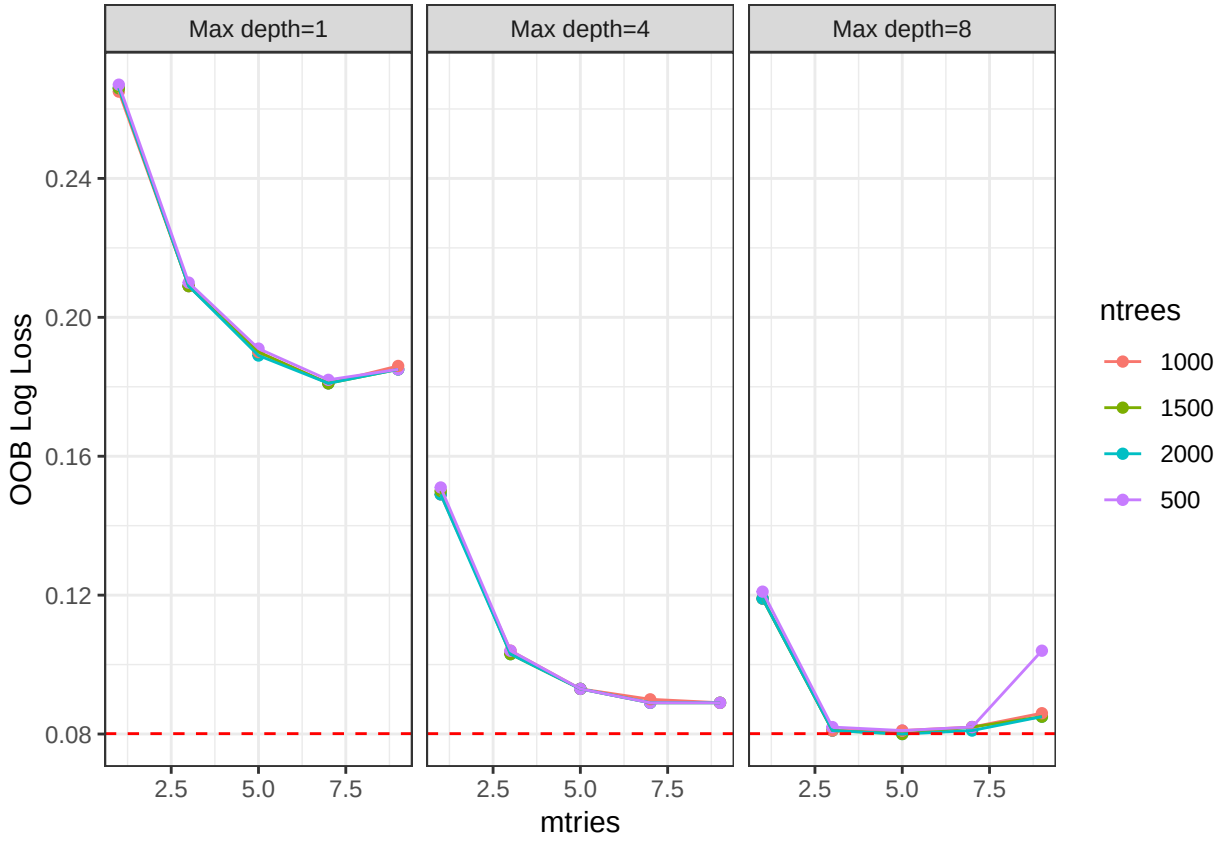


Figure 2: Influence of hyperparameters on random forest

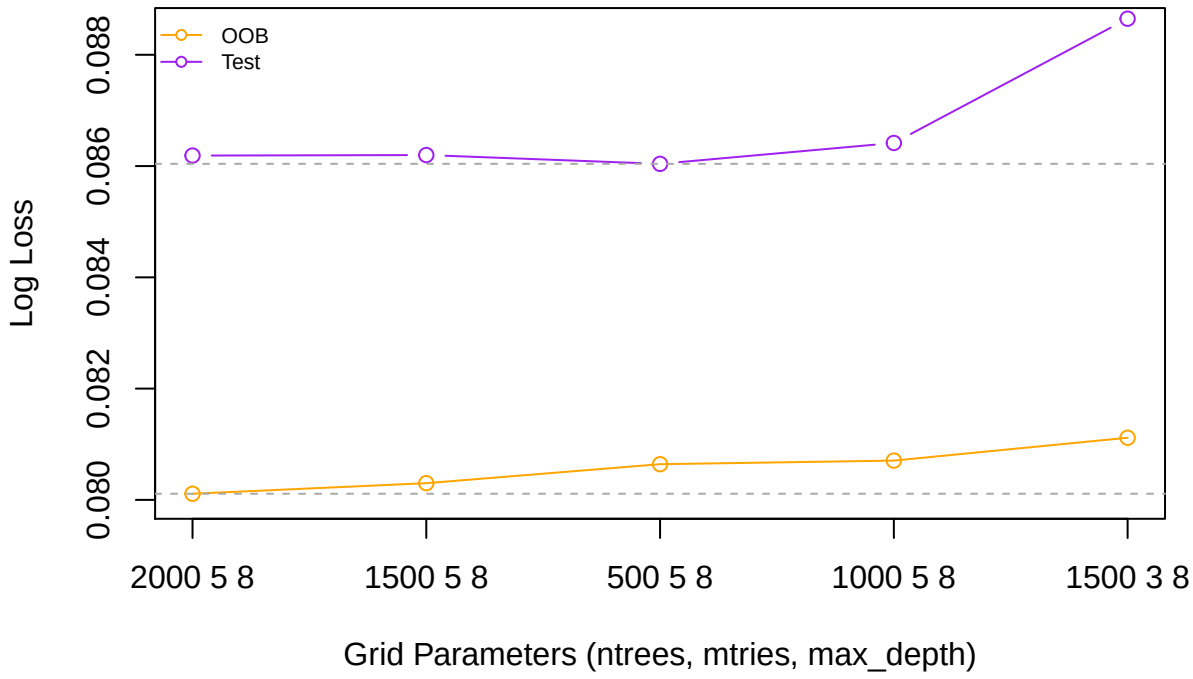


Figure 3: Performance of random forest models

Table 2: Summary of hyperparameters used in boosting

Hyperparameter	Description	Values
learning_rate	Rate at which the learning occurs	.001, 0.01
ntrees	Number of trees grown	500, 1000, 1500
max_depth	Depth of tree from root to terminal nodes	4, 8

Based on the OOB log loss, the random forest with $mtries = 5$, $maxdepth = 8$ and $ntrees = 2000$ is the best model. We consider this as it was done on the training set. Figure 2 also confirms the assertion discussed towards the end of the previous paragraph. However, if we only consider the OOB log loss we see that therandom forest with $mtries = 5$, $maxdepth = 8$ and $ntrees = 2000$ is the best model.

3 Boosting

3.1 Influence of hyperparameters on Boosting.

As done above for the random forest procedure, a selection of different boosting hyperparameters were considered.

The hyperparameters tuned were: *learning_rate*, *ntrees* and *max_depth*. Table 2 shows the values used for each hyperparameter.

Using h2o, boosting models were built for different combinations of the hyperparameters. Cross validation log loss was considered as a performance metric for these models. Figure 4 shows that when $learning.rate = 0.001$, performance based on log loss, improves as the number of trees increases, but there is no significant effect for *max_depth*. On the other hand, for $learnrate = 0.01$, log loss decreases as the number of trees and *max_depth* decreases.

3.2 Model selection

After the boosting procedure (model building), the performance of different combinations of hyperparameters boosting models is plotted in Figure 5:

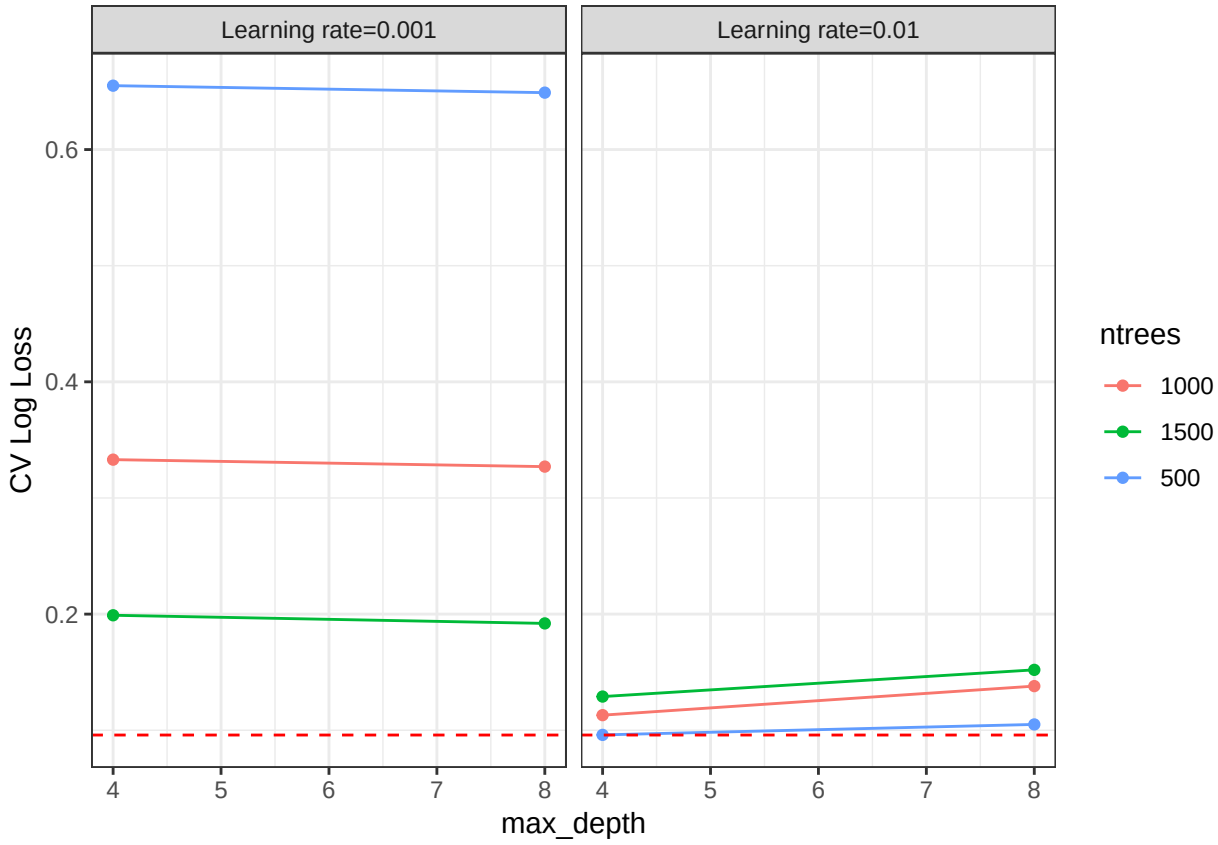


Figure 4: Influence of hyperparameters for boosting

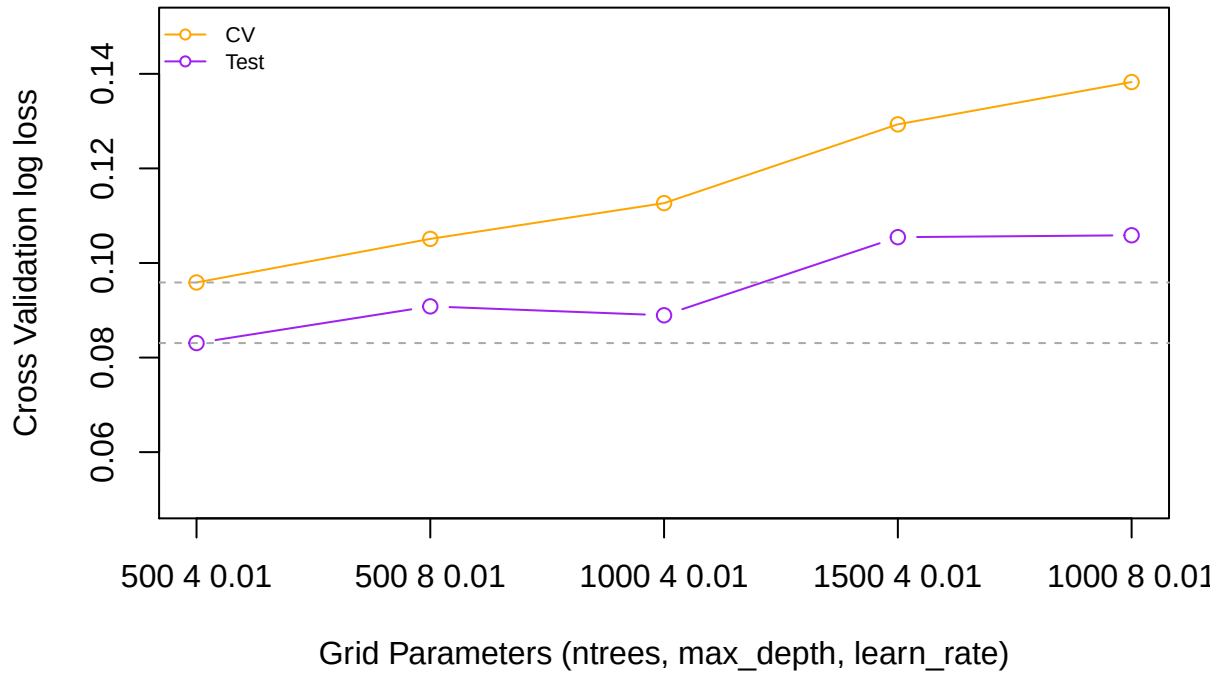


Figure 5: Performance of 5 best boosting models

Figure 5 shows that the best boosting model had hyperparameters $ntrees = 500$, $max_depth = 4$ and $learnrate = 0.01$. As can be seen, both the cv log loss for the training set and test cv log loss give a low value for this particular boosting model.

4 Comparing the performance of random forest and boosting.

The random forest ($mtries = 5$, $maxdepth = 8$ and $ntrees = 2000$) performs better than boosting based on *log loss* and *prediction accuracy* as shown in table 3.

Prediction accuracies for both models show that 2.65568% of the classes were misclassified under the random forest and a 2.83883% misclassification error seen for the boosting model. These two values are quite evident if we consider looking at the individual confusion matrices. We see that if we consider the confusion matrix under the random forest model, 4 classes were predcited to be class 1 instead of the true prediction of class 2 and we make note that similar low misclassifications were made for other class labels. The same can be seen in the boosting model confusion matrix.

Table 3: Comparing performance metrics

	Log loss	Accuracy	Misclassification
Random Forest	0.0801101	0.9734432	0.0265568
Boosting	0.0958794	0.9716117	0.0283883

In terms of sensitivity and specificity performance, table 4 shows that there is no significant difference across both Random Forest and Boosting. While the specificity score is relatively high for classes with low number of observations (*class 2, 3 ,4 and 5*),the sensitivity score is particularly very low for *class 5*.

```
          Reference
Prediction   1   2   3   4   5
         1 968   4   0   1  10
         2   6  61   0   0   0
         3   5   0   5   0   0
         4   2   0   0  16   0
         5   1   0   0   0  13

          Reference
Prediction   1   2   3   4   5
         1 967   5   0   1   9
         2   7  60   0   0   0
         3   3   0   5   0   1
         4   2   0   0  16   0
         5   3   0   0   0  13
```

As we have considered the random forest as the better model, we plot a variable of importance plot as seen in figure 6 below. A variable of importance plot is a plot that ranks how important an explanatory variables is in relation to the response variable. Figure 6 shows that `height` is the most significant variable followed by `mean_tr` and so on.

Table 4: Comparing sensitivity and specificity

	RF Sensitivity	Boosting Sensitivity	RF Specificity	Boosting Specificity
Class: 1	0.986	0.985	0.864	0.864
Class: 2	0.938	0.923	0.994	0.993
Class: 3	1.000	1.000	0.995	0.996
Class: 4	0.941	0.941	0.998	0.998
Class: 5	0.565	0.565	0.999	0.997

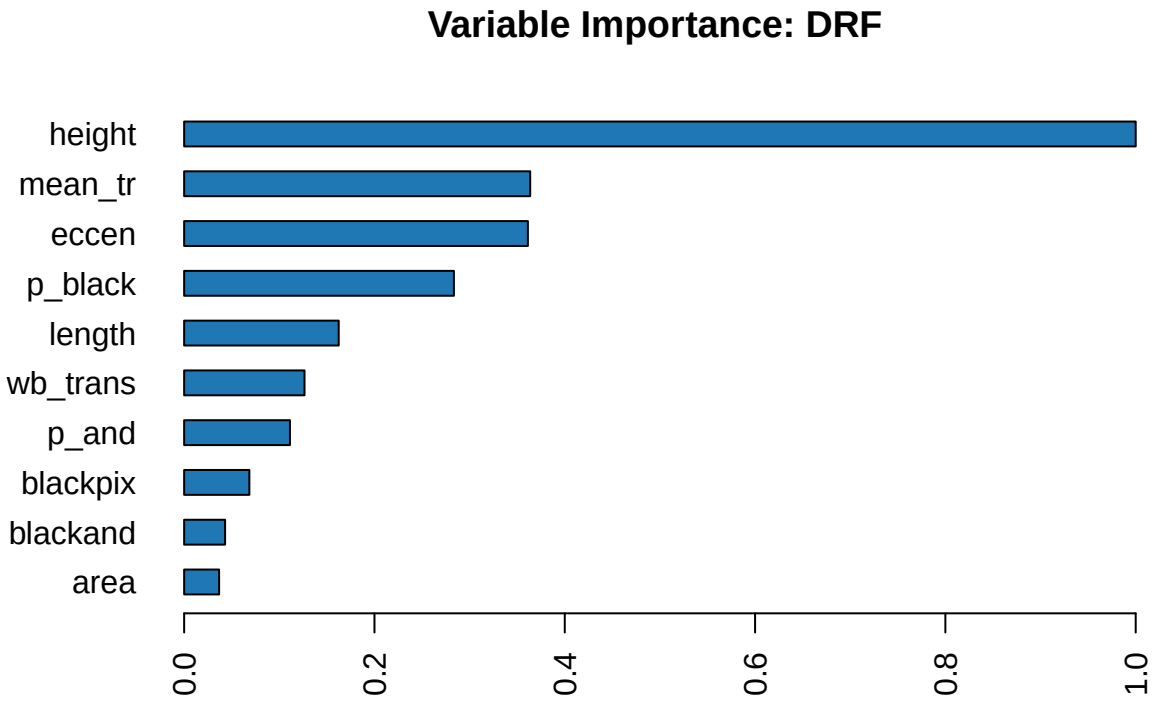


Figure 6: Variable importance plot for Random Forest

Appendix

Linear models:

• **model 1**: Full model.
• **model 2**: SalePrice~1
• **model 3**: SalePrice ~ BsmtQual + CentralAir + KitchenQual + FireplaceQu + GarageFinish + MSZoning + OverallQual + YearBuilt + YearRemodAdd + TotalBsmtSF + GrLivArea + FullBath + TotRmsAbvGrd + GarageArea
• **model 4**: SalePrice ~ BsmtQual + CentralAir + KitchenQual + YearRemodAdd + TotalBsmtSF + GrLivArea + GarageArea + MSZoning + OverallQual + GarageFinish

Code:

```
#QUESTION 1

# Load Packages required
#library
library(tidyr)
library(tidyverse)
library(randomForest)
library(gbm)
library(tree)
library(ggcorrplot)
library(knitr)
library(leaps)
library(doParallel)
library(foreach)
 # Set seed to project number
set.seed(23)

#loading data
load("clean_data.RData")
par(mfrow = c(1,2))
hist(df$SalePrice, xlab = "SalePrice",
     main ="Histogram of SalePrice", freq = F, col ="yellow")
hist(log(df$SalePrice), xlab = "log(SalePrice)",
     main ="Histogram of log(SalePrice)", freq = F, col ="yellow")

load("clean_data.RData") # Load data set
inds <- 1:81
num.vars <- sapply(inds, function(x){is.numeric(df[,x])})
ind.num.vars <- which(num.vars) # Indices of all numeric variables
num.df <- df[,c(ind.num.vars)] # DF of all numeric vars
cor <- cor(num.df[,-1]) # Correlation(without X)
cor <- as.data.frame(cor)
cor.SALES <- cor[37,] # Find correlation values with sale price
high.ind <- which(abs(cor.SALES)>.5) # Extract only highly correlated values
high.cor.sales <- cor[c(high.ind),c(high.ind)] # Rebuild correlation matrix
min <- min(high.cor.sales)
max <- max(high.cor.sales)
```

```r
# lab = T gives vals
ggcorrplot(high.cor.sales)+ scale_fill_gradient2(
  limit = c(min-0.02,max), low = "white", high = "darkblue",
  mid = "purple", midpoint = (min+max)/2)


plot(df$SalePrice~df$GrLivArea,
     col=ifelse(df$GrLivArea>=4000,"red","black"),
     xlab="GrLivArea",ylab="Sale Prices", pch=16)
abline(v=4000, lty=2, col='red')

load("FINALDATA.RData")

#data wranggling

load("factors.RData")

library(ggplot2)
library(reshape2)

melt.data <- melt(df.factors)

# 1-10
df.melts1= melt(df.factors[,1:10], id.vars = c("SalePrice"),
             measure.vars = colnames(df.factors)[2:10])
p <- ggplot(data = df.melts1, aes(x=value, y=SalePrice)) +
  geom_boxplot(aes(fill="orange"))
+theme(axis.text.x =
                    element_text(angle = 90, hjust = 1))
p + facet_wrap( ~ variable, scales="free")


df.final <- na.omit(df.keep) # Remove NAS

df.final <- df.final %>% filter(GrLivArea <=4000) # Ref decock pdf # outliers

# Set training and test data set
train=sample(1:nrow(df.final), nrow(df.final)*.85)
df.test = df.final[-train,]
df.train = df.final[train,]

# Number of parameters
p<-22

tree1 <- tree(log(SalePrice)~., data = df.train)
# summary(tree1)
plot(tree1)
text(tree1,pretty=0)

#cost complexity pruning.
#we grew the full tree, now we prune.
# plot(prune.tree(tree1))
```

```r
#extract alpha/cost complexity values
prune.tree(tree1)
alphas<- prune.tree(tree1)$k
alpha <- c(-Inf,1.782,2.691,2.839,3.697,4.403,8.645,12.279)
nodes <- c(10,9,8,7,6,5,4,3)
kable(rbind(alpha,nodes), format = "latex")

bag.train = randomForest(log(SalePrice) ~.,
                         data=df.train,mtry = 22,
                         ntree= 1000, importance = TRUE)


ran.forest <- randomForest(log(SalePrice)~., data = df.train,
                           ntree = 1000, mtry = sqrt(p),
                           importance = TRUE, na.action = na.exclude)
ran.forest1 <- randomForest(log(SalePrice)~., data = df.train,
                            ntree = 1000, mtry = p/2,
                            importance = TRUE, na.action = na.exclude)
ran.forest2 <- randomForest(log(SalePrice)~., data = df.train,
                            ntree = 1000, mtry = p/3,
                            importance = TRUE, na.action = na.exclude)


# PLOT OOB ERROR
plot(bag.train$mse, type='l',col='purple', xlab="Number of trees",
     ylab="Out-of-bag error",ylim=c(0.021,0.027))
lines(ran.forest$mse, type='l', col='orange')
lines(ran.forest1$mse, type='l',col='red')
lines(ran.forest2$mse, type='l', col='green')
legend("topright",legend=c(expression(paste("m=",p)),
                           expression(paste("m=",sqrt(p))),
                           expression(paste("m=",p/2)),
                           expression(paste("m=",p/3))),
       col=c("purple","orange","red","green"),pch=16, cex=.7,
       box.lty = 0)

df.boost <- df.train %>% mutate(SalePrice=log(SalePrice))

# B=1000
# Boosting using d=1

boost.tree1 <- gbm(SalePrice ~., data=df.boost,
                   distribution="gaussian",
                   n.trees=1000, interaction.depth=1,
                   shrinkage=0.01,
                   bag.fraction=1, cv.folds=10, n.cores=3)
# Boosting using d=2
boost.tree2 <- gbm(SalePrice ~., data=df.boost,
                    distribution="gaussian",
                    n.trees=1000, interaction.depth=2,
                    shrinkage=0.01,
                    bag.fraction=1, cv.folds=10, n.cores=3)
# Boosting using d=3
```

```r
boost.tree3 <- gbm(SalePrice ~., data=df.boost,
                   distribution="gaussian",
                   n.trees=1000, interaction.depth=3,
                   shrinkage=0.01,
                   bag.fraction=1, cv.folds=10, n.cores=3)


#plot OOB
plot(boost.tree1$cv.error,type='l',col='green',
     ylab="Cross validation error", xlab="Number of trees",
     ylim=c(0.0175, 0.1))
lines(boost.tree2$cv.error, type='l', col='orange')
lines(boost.tree3$cv.error, type='l', col='red')
# lines(boost.tree12$cv.error, type='l',col='blue')
# lines(boost.tree22$cv.error, type='l',col='darkgreen')
# lines(boost.tree32$cv.error, type='l',col='violet')
legend("topright",
       legend=c("B=1000 d=1","B=1000 d=2","B=1000 d=3"),col=c("green","orange","red"),pch=16,cex=.7,
       box.lty=0)
Model <-c("model 1","model 2","model 3","model 4","model 5")
cv.error <- c(0.1620859,0.0195533,0.01924226,0.019688,0.01971477)
kable(rbind(Model,cv.error),format="latex")


df.final <- na.omit(df.keep) # Remove NAS
df.final$SalePrice <- log(df.final$SalePrice)
df.final <- df.final %>% filter(GrLivArea <=4000)
# Ref decock pdf # outliers

# Set training and test data set
train=sample(1:nrow(df.final), nrow(df.final)*.85)
df.test = df.final[-train,]
df.train = df.final[train,]

# FACTORS NOT IN TEST SETS
df.train = df.train[,-c(grep("Heating",colnames(df.train)), grep("Exterior1st",colnames(df.train)),grep

                        grep("GarageQual",colnames(df.train)))]


#fit linear model with all variables
model.1 <- glm(log(SalePrice)~., family = gaussian("identity"),
               data = df.train)
model.1.1 <- glm(SalePrice~., family = gaussian("identity"),
                 data = df.train)
model.1.2 <- glm(SalePrice~1, family = gaussian("identity"),
                 data = df.train)

summary(model.1)


##rebuild, only with reduced variables
model.2 <- glm(log(SalePrice) ~ BsmtQual + CentralAir + KitchenQual +
```

```
                  YearRemodAdd + TotalBsmtSF + GrLivArea + GarageArea +
                  MSZoning + OverallQual + GarageFinish,
                family = gaussian("identity"), data = df.train)
summary(model.2)

#stepAIC
y <- df.train$SalePrice

stepAIC(model.1.1, trace =T, scope = list(upper = y~., lower = ~1),
        direction = "backward") #backward AIC
stepAIC(model.1.2, trace =T, scope = list(upper = y~., lower = ~1),
        direction = "forward") #forward AIC

#all subset regression
#consider al subsets with method metric being adjusted r-sq
#leaps(x = df.train[,-23], y = df.train$SalePrice, int = TRUE,
method = c("adjr2"), names = names(df.train[,23]))

#cross validation
library(doParallel)
library(foreach)
detectCores()
cl <- makeCluster(max(1,detectCores()-1))
## Initiate cluster and never use all of them!
registerDoParallel(cl)


cv.lm <- function(formula, data, K){

  # create CV sets
  m <- model.frame(data)
  set.seed(23)
  rand <- sample(K, length(m[[1L]]), replace = TRUE)
  table(rand)

  rss <- numeric(0)
  predict <- numeric(0)
  X <- numeric(0)
  Observed <- numeric(0)

  uniquefolds = unique(rand)

  # Iterate through sets
  foreach(i=1:K)%dopar%{
    i = uniquefolds[i]
    trainingFold = m[rand != i, , drop = FALSE]
    testFold = m[rand ==i, , drop = FALSE]

    mod <- glm(formula,data=trainingFold,family = gaussian("identity"))
    predictTestFold <- predict(mod, testFold)

    X <- c(X, as.integer(row.names(testFold))) #Obs number
    Observed <- c(Observed, testFold$SalePrice) # Actual Values
```

```
    predict<- c(predict,predictTestFold) # Predicted
    rss<- c(rss,((predictTestFold-testFold$SalePrice)^2)) #RSS
  }

  mse <- mean(rss, na.rm=T) #MSE
  output= list(Formula=formula,
               X=X, Observed=Observed, Predictions=predict,
               ResidualSumSquares =rss, MeanSquareError=mse)
  return (output)
}

#fullmodel<-cv.lm(SalePrice~., df.train,10)
#comp<-as.data.frame(cbind(fullmodel$X, fullmodel$Predictions, fullmodel$Observed))

#fullmodel$MeanSquareError # gives MSE

# Want to compare different models

formula <- c("SalePrice~.", "SalePrice~1", "SalePrice ~ BsmtQual
+ CentralAir + KitchenQual + YearRemodAdd + TotalBsmtSF + GrLivArea
+ GarageArea +
MSZoning + OverallQual + GarageFinish","SalePrice ~ BsmtQual
+ CentralAir +
             KitchenQual + FireplaceQu + GarageFinish + MSZoning
+ OverallQual +
YearBuilt + YearRemodAdd + TotalBsmtSF + GrLivArea + FullBath +
             TotRmsAbvGrd + GarageArea", "SalePrice ~ BsmtQual +
CentralAir + KitchenQual  + GarageFinish + MSZoning + OverallQual +
YearRemodAdd + TotalBsmtSF + GrLivArea + FullBath + GarageArea")

# Apply CV to both models
output<-lapply(formula, cv.lm, K=10,data=df.train)

# Compare the models
output[[1]]$Formula
output[[1]]$MeanSquareError

output[[2]]$Formula
output[[2]]$MeanSquareError

output[[3]]$Formula
output[[3]]$MeanSquareError

output[[4]]$Formula
output[[4]]$MeanSquareError

output[[5]]$Formula
output[[5]]$MeanSquareError

#choose fourth model
#predict
model.3 <- glm(SalePrice ~ BsmtQual + CentralAir + KitchenQual +
                 FireplaceQu + GarageFinish + MSZoning + OverallQual
```

```
                + YearBuilt + YearRemodAdd
                + TotalBsmtSF + GrLivArea + FullBath
                + TotRmsAbvGrd
                + GarageArea, family = gaussian("identity"),
                data = df.train)

predictions <- predict(model.3, newdata = df.test) #predictions
obs <- df.test$SalePrice
error <- (obs - predictions)**2
mean(error) #mean square error for our given model.

#plots that depict our analysis
plot(model.3)

# COMPARE BAGGING AND BOOSTING
plot(boost.tree3$cv.error,type='l',col='green',
     ylab="Error", xlab="Number of trees", ylim=c(0.02,.05))
lines(ran.forest2$mse, type='l', col='purple')
legend("topright",legend=c("Boosting CV Error","Random Forest OOB Error"),
       col=c("green","purple"),pch=16, box.lty=0, cex=.7)

## RUN PREDICTIONS TO GET MSE


# Regression Tree Preds

predict.reg <- predict(tree2, newdata = df.test[,-23])
#carry out predictions using our refined model
log.tests <- log(df.test$SalePrice)
out <- data.frame(cbind(actuals = log.tests,
                        predicted = predict.reg))
#will calculate your prediction accuracy,
sees how closely correlated actuals are with preds

mse.reg <- mean((log.tests-predict.reg)^2)

# boosting

predict.boost <- predict(boost.tree3, newdata=df.test[,-23])
mse.boost <- mean((log.tests-predict.boost)^2)

#random forest

predict.rf<- predict(ran.forest2, newdata=df.test[,-23])
mse.rf <- mean((log.tests-predict.rf)^2)


compare.mse <- cbind(RegressionTree = round(mse.reg,4),
                     Boosting=round(mse.boost,4),
                     RandomForest = round(mse.rf,4))

kable(compare.mse, caption="Compare mean square error values on test set"
      , format="latex")
```

```r
boost.imp <- as.data.frame(summary(boost.tree3,plotit=F))

ggplot(boost.imp[1:5,], aes(x = reorder(var,rel.inf),
                            y = rel.inf, fill = rel.inf)) +
  labs(y = 'Variable Importance - Boosting', x = "",
       fill="Relative influence") + coord_flip() +
  theme(legend.position = "none") +geom_bar(stat = "identity")
+theme_bw()


#QUESTION2
#libraries
library(caret)
library(randomForest)
library(h2o)
library(knitr)

# Load data set
blocks <- read.csv("blocks.csv", header =T)
blocks <- blocks[,-1] # Remove X colums
blocks$class <- as.factor(blocks$class)
# Change response variable to factor - classification


# Set seed to project number
set.seed(23)


# To preserve overall class distribution of data set
# Take .8 proportion out of each y factor levels
# for accute representation of the data in training set

trainIndex <- createDataPartition(blocks$class, p = .8,
                                  list = FALSE,
                                  times = 1)
#head(trainIndex)

blocks_train <- blocks[trainIndex,] # Training set
blocks_test <- blocks[-trainIndex,] # Test set

# h2o

localH2O = h2o.init(ip = "localhost", port = 54321,
                    startH2O = TRUE,min_mem_size = "2g")
h2o.clusterInfo()

# create h2o objetcs

blocksTrain_h2o <- as.h2o(blocks_train)
blocksTest_h2o <- as.h2o(blocks_test)


# Analyse data set
```

```r
freq <- table(blocks$class)
freq = ((freq)/nrow(blocks))*100
freq = as.data.frame(freq) # get proportion of observations for each class
# compare this proportion and confusion matrix at the end of
# to see if model accurately predicts rare occuring events

# Imbalanced data
ggplot(data = freq, aes(x = Var1, y = Freq))
+ geom_bar(stat = "identity", fill ="orange") +
  labs(y = "% of class", x = "class")+theme_bw()


# RANDOM FOREST
p <-10 # number of predictor variables


# set hyperparameters
mtries <- seq(1,10, by=2)
ntrees = c(500,1000,1500,2000)
max_depth = c(1,4,8)


hyper_params <- list(mtries = mtries, ntrees = ntrees,
                     max_depth = max_depth)

# use OOB MSE as random forest no need for CV
rfGrid<-h2o.grid(algorithm = "randomForest",
                 model_id= "rf_grid",
                 hyper_params = hyper_params,
                 x = 1:10, y = 11,
                 training_frame = blocksTrain_h2o,
                 #validation_frame = datTest_h2o,
                 nfolds = 0,
                 seed = 23)



summary.rf <- rfGrid@summary_table # get summary of models in DF

# Reformat data frame
summary.rf$logloss <- as.double(summary.rf$logloss)
summary.rf$mtries <- as.integer(summary.rf$mtries)

col1 <- c("mtries","ntrees","max_depth")
col2 <- c("Number of randomly selected predictor variables",
          " Number of trees grown","Depth of tree from root
          to terminal nodes")
col3 <- c("1, 3, 5, 7, 9", "500, 1000, 1500, 2000","1, 4, 8")
tab <- cbind(col1, col2, col3)
colnames(tab) <- c("Hyperparameter","Description","Values")
kable(tab, caption="Summary of hyperparameters in random forest",
      format="latex")
```

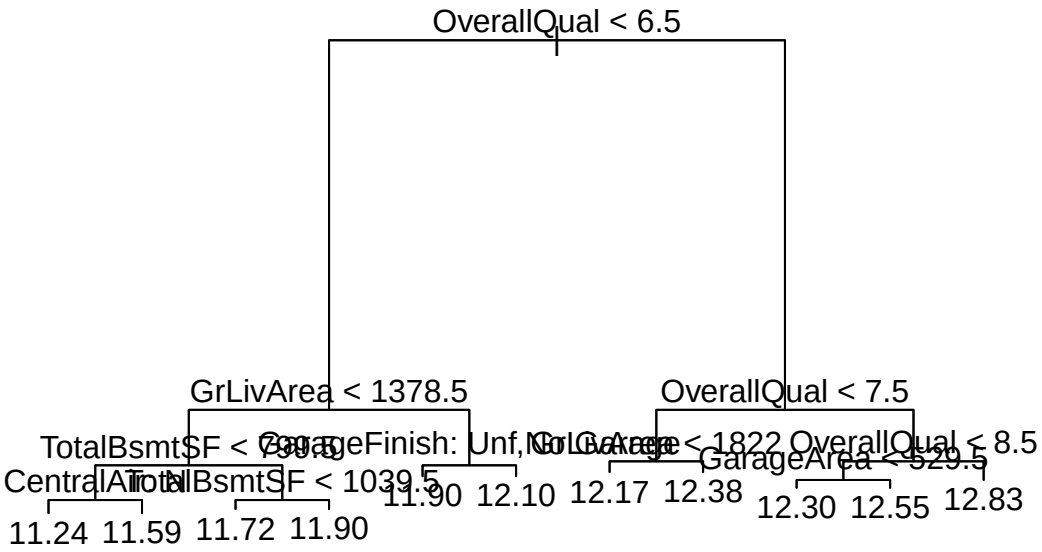
```
gridlabs <- c('1'='Max depth=1', '4'='Max depth=4','8'='Max depth=8')
ggplot(summary.rf, aes(x = mtries, y = round(logloss,3),
                       colour=ntrees,group = ntrees)) + labs(y="OOB Log Loss")+
  geom_point() + geom_line()+ geom_hline(yintercept=min(summary.rf$logloss), col="red",lty=2)+
  facet_grid( ~ max_depth,labeller=labeller(max_depth=gridlabs))+
  theme_bw()


# compare the performance of first 5 models

models.rf=list()
for(i in 1:5){
  models.rf[i] = h2o.getModel(rfGrid@model_ids[[i]])
}

# x axis label with parameters
xlabel.rf = NULL
for(i in 1:5){
  xlabel.rf[i] = paste(models.rf[[i]]@allparameters$ntrees,
                    models.rf[[i]]@allparameters$mtries,
                    models.rf[[i]]@allparameters$max_depth)
}

#  logloss
loglossOOB.rf = NULL
for(i in 1:5){
  loglossOOB.rf[i] = h2o.logloss(models.rf[[i]])
}

# test set  logloss
# TEST SET  logloss
loglossTest.rf = NULL
for (i in 1:5) {
  loglossTest.rf[i] <- h2o.logloss
  (h2o.performance(h2o.getModel(rfGrid@model_ids[[i]]),

                    newdata = blocksTest_h2o))
}

## plot logloss of first 5 models

plot(loglossTest.rf[1:5], xaxt = "n",xlab=
       "Grid Parameters (ntrees, mtries, max_depth)", type = "b",
     col ="purple",ylim=c(0.08,0.0885),ylab="Log Loss")
axis(1, at=1:5, labels=xlabel.rf[1:5])
lines(loglossOOB.rf[1:5], type = "o", col ="orange")
abline(h=min(loglossTest.rf[1:5]), lty=2, col='darkgrey')
abline(h=min(loglossOOB.rf[1:5]), lty=2, col='darkgrey')
legend("topleft",c("OOB","Test"),pch = 21, pt.bg = "white",
       lty = 1, col = c("orange",'purple'), box.lty=0, cex=.7)

max_depth.gbm=c(4,8)
n.trees=c(500,1000,1500)
```

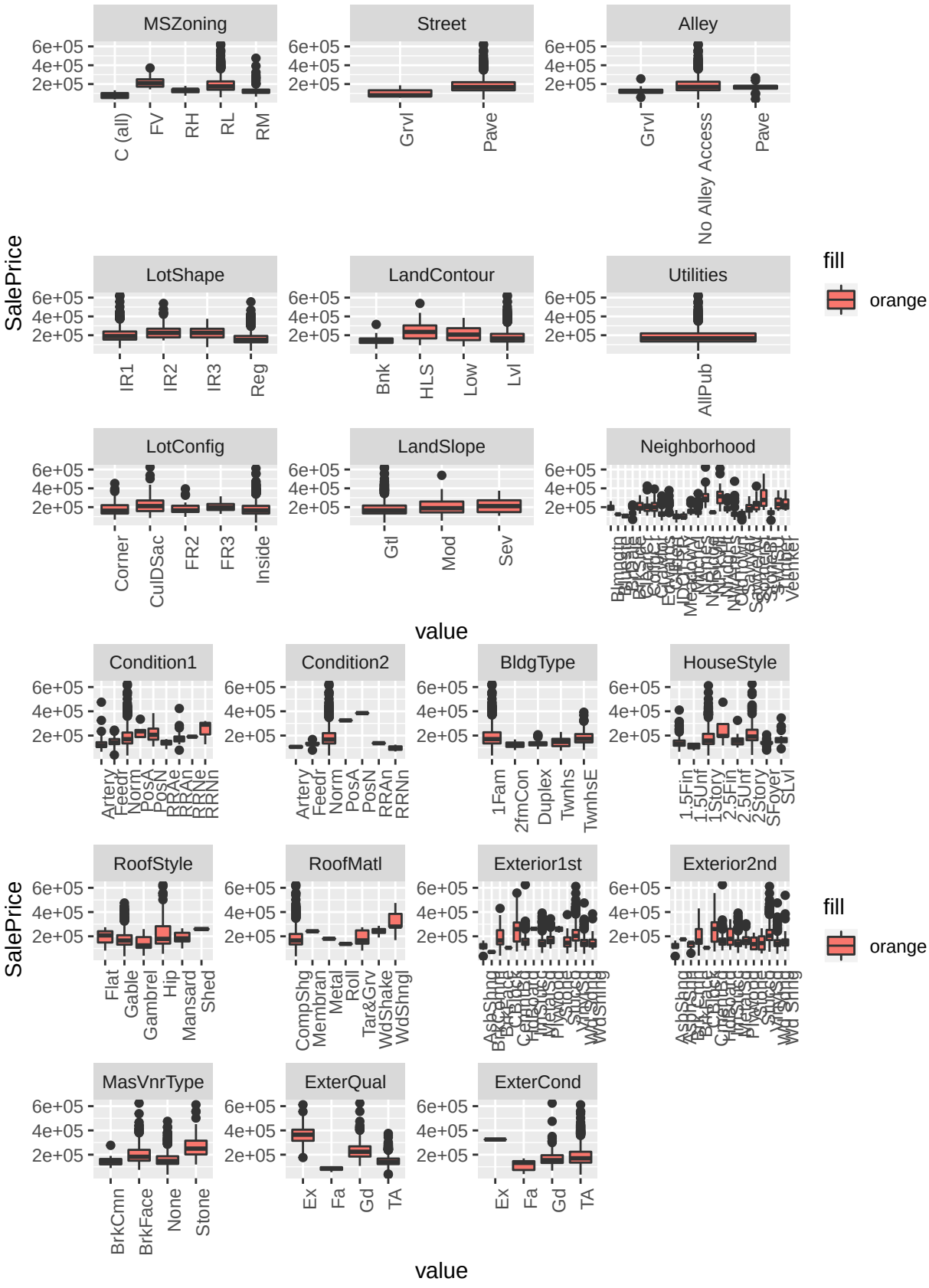
```
learning_rate=c(0.001,0.01)

hyper_params_gbm <- list(ntrees = n.trees,
                         max_depth = max_depth.gbm, learn_rate = learning_rate)

#10 fold CV <- CV error
gbm_grid <- h2o.grid(algorithm = "gbm",
                     hyper_params = hyper_params_gbm,
                     x = 1:10, y = 11,
                     training_frame = blocksTrain_h2o,
                     nfolds = 3,
                     seed = 23)

summary_gbm <- gbm_grid@summary_table
summary_gbm$logloss <- as.double(summary_gbm$logloss)
summary_gbm$max_depth <- as.integer(summary_gbm$max_depth)
col11 <- c("learning_rate","ntrees","max_depth")
col22 <- c(" Rate at which the learning occurs",
           " Number of trees grown",
           "Depth of tree from root to terminal nodes")
col33 <- c(".001, 0.01", "500, 1000, 1500", "4, 8")
tab1<-cbind(col11,col22,col33)
colnames(tab1) <-c("Hyperparameter",
                   "Description","Values")
kable(tab1, caption="Summary of hyperparameters used in boosting",
      format="latex")
# Use diagram to explain how the hyperparams affect performance
# want to minimize loglosss
gridlabs_gbm <- c('0.001'='Learning rate=0.001',
                  '0.01'='Learning rate=0.01')
ggplot(summary_gbm, aes(x = max_depth,
                        y = round(logloss,3),
                        colour=ntrees,group = ntrees))
+ labs(y='CV Log Loss') +
  geom_point() + geom_line()+ geom_hline(yintercept=min(summary_gbm$logloss), col="red",lty=2)+
  facet_grid( ~ learn_rate,labeller=labeller(learn_rate=gridlabs_gbm))+
  theme_bw()
# compare the performance of first 5 models

models.gbm=list()
for(i in 1:5){
  models.gbm[i] = h2o.getModel(gbm_grid@model_ids[[i]])
}

# x axis label with parameters
xlabel.gbm = NULL
for(i in 1:5){
  xlabel.gbm[i] = paste(models.gbm[[i]]@allparameters$ntrees,
                        models.gbm[[i]]@allparameters$max_depth,
                        models.gbm[[i]]@allparameters$learn_rate)
}

# Log loss
```

```
loglosscv.gbm = NULL
for(i in 1:5){
  loglosscv.gbm[i] = h2o.logloss(models.gbm[[i]],xval=TRUE)
}

# test set deviance
# TEST SET RMSE
loglossTest.gbm = NULL
for (i in 1:5) {
  loglossTest.gbm[i] <- h2o.logloss
  (h2o.performance(h2o.getModel(gbm_grid@model_ids[[i]]),
                                      newdata = blocksTest_h2o))
}

## plot logloss of first 5 models

plot(loglossTest.gbm[1:5], xaxt = "n",xlab=
       "Grid Parameters (ntrees, max_depth, learn_rate)", type = "b",
     col ="purple", ylab="Cross Validation log loss", ylim=c(0.05,0.15))
lines(loglosscv.gbm[1:5], type = "o", col ="orange")
axis(1, at=1:5, labels=xlabel.gbm[1:5])
abline(h=min(loglossTest.gbm[1:5]), lty=2, col='darkgrey')
abline(h=min(loglosscv.gbm[1:5]), lty=2, col='darkgrey')
legend("topleft",c("CV","Test"),pch = 21, pt.bg = "white",
       lty = 1, col = c("orange",'purple'), box.lty=0, cex=.7)

##get confusion matrix random forest
best_rf <- h2o.getModel(rfGrid@model_ids[[1]])
best_gbm <- h2o.getModel(gbm_grid@model_ids[[1]])

# get log loss
logloss_rf <- h2o.logloss(best_rf)
logloss_gbm <- h2o.logloss(best_gbm, xval=T)

# get predictions
predictionsTest_rf <- h2o.predict(best_rf, blocksTest_h2o)
yhatTest_rf = as.factor(as.matrix(predictionsTest_rf$predict))
cm_rf<-confusionMatrix(yhatTest_rf, blocks_test$class) # Confusion matrix

##get confusion matrix boosting
predictionsTest_gbm <- h2o.predict(best_gbm, blocksTest_h2o)
yhatTest_gbm = as.factor(as.matrix(predictionsTest_gbm$predict))
cm_gbm<-confusionMatrix(yhatTest_gbm, blocks_test$class)
# Confusion matrix  #CONFUSION MATRIX (1st)
cm_rf$table
# for rf#CONFUSION MATRIX (2)
cm_gbm$table
# for gbm
stats <- cbind(c(logloss_rf, logloss_gbm),rbind(cm_rf$overall[1],
                                                cm_gbm$overall[1]))
stats <- cbind(stats,1-stats[,2])
rownames(stats)= c("Random Forest", "Boosting")
colnames(stats)=c("Log loss", "Accuracy", "Misclassification")
```

```
kable(stats,caption="Comparing performance metrics", format="latex")


by.class <- cbind(round(cm_rf$byClass[,1],3), round(cm_gbm$byClass[,1],3),round(cm_rf$byClass[,2],3),
                  round(cm_gbm$byClass[,2],3))
colnames(by.class) <-cbind("RF Sensitivity","Boosting Sensitivity",
                           "RF Specificity", "Boosting Specificity")


kable(by.class, caption ="Comparing sensitivity and specificity", format="latex")h2o.varimp_plot(best_r
```

Plots:




MSZoning
Street
Alley
LotShape
LandContour
Utilities
LotConfig
LandSlope
Neighborhood
Condition1
Condition2
BldgType
HouseStyle
RoofStyle
RoofMatl
Exterior1st
Exterior2nd
MasVnrType
ExterQual
ExterCond
SalePrice
value
fill
orange
6e+05
4e+05
2e+05
C (all)
FV
RH
RL
RM
Grvl
Pave
No Alley Access
IR1
IR2
IR3
Reg
Bnk
HLS
Low
Lvl
AllPub
Corner
CulDSac
FR2
FR3
Inside
Gtl
Mod
Sev
Artery
Feedr
Norm
PosA
PosN
RRAe
RRAn
RRNe
RRNn
1Fam
2fmCon
Duplex
Twnhs
TwnhsE
1.5Fin
1.5Unf
1Story
2.5Fin
2.5Unf
2Story
SFoyer
SLvl
Flat
Gable
Gambrel
Hip
Mansard
Shed
CompShg
Membran
Metal
Roll
Tar&Grv
WdShake
WdShngl
BrkCmn
BrkFace
None
Stone
Ex
Fa
Gd
TA