Stats Final Project - Group 2

2022-11-30

##Setup: load packages, import test and train datasets, recode variables

library(readr)  
library(fastDummies)  
library(glmnet)  
library(tidyr)  
library(dplyr)  
library(ggplot2)  
library(caret)  
library(xgboost)  
library(gbm)

train <- read\_csv("train.csv") #ID 1460 is the last training house  
test <- read\_csv("test.csv") #ID 1461 is the first testing house, 2919 is the last house  
  
test$SalePrice <- 0 #placeholder for later predictions  
  
combined <- rbind(train, test)

The variables we are particularly interested in are:

* MSZoning: The general zoning classification
* LotArea: Lot size in square feet
* LotConfig: Lot configuration
* HouseStyle: Style of dwelling
* OverallCond: Overall condition rating
* YearBuilt: Original construction date
* YearRemodAdd: Remodel date
* BsmtCond: General condition of the basement
* HeatingQC: Heating quality and condition
* CentralAir: Central air conditioning
* FullBath: Full bathrooms above grade
* Bedroom: Number of bedrooms above basement level
* Kitchen: Number of kitchens
* GarageType: Garage location
* GarageCars: Size of garage in car capacity
* PavedDrive: Paved driveway
* Fence: Fence quality
* MoSold: Month Sold
* YrSold: Year Sold

We want to use these variables to predict the SalePrice of a house.

In order to do any sort of analysis, we need to clean up the data. We can start by recoding many of the variables to be factors instead of characters.

#Redefine levels for some variables  
  
combined$MSZoning <- as.factor(combined$MSZoning)  
levels(combined$MSZoning) <- c("C", "FV", "RH", "RL", "RM")  
combined$MSZoning <- as.character(combined$MSZoning)  
  
combined$OverallCond <-as.character(combined$OverallCond)  
  
combined$Fence <- as.factor(combined$Fence)  
levels(combined$Fence) <- c("Fence", "Fence", "Fence", "Fence")  
combined$Fence <- as.character(combined$Fence)  
  
combined$PavedDrive <- as.factor(combined$PavedDrive)  
levels(combined$PavedDrive) <- c("NotPaved", "NotPaved", "Paved")  
combined$PavedDrive <- as.character(combined$PavedDrive)  
  
combined$GarageType <- as.factor(combined$GarageType)  
levels(combined$GarageType) <- c("Attchd", "Attchd", "Attchd", "Attchd", "Detchd", "Detchd")  
combined$GarageType <- as.character(combined$GarageType)  
  
combined$MoSold <- as.factor(combined$MoSold)  
levels(combined$MoSold) <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")  
combined$MoSold <- as.character(combined$MoSold)  
  
#Replace NA values  
  
combined$Fence <- combined$Fence %>% replace\_na("NoFence")  
combined$GarageType <- combined$GarageType %>% replace\_na("NoGarage")  
combined$BsmtCond <- combined$BsmtCond %>% replace\_na("NoBsmt")  
  
#New binary variable for if a house was sold during/after 2008 recession  
  
combined$SaleInRecession <- ifelse(combined$YrSold >= 2008, "Yes", "No")  
  
#Select only variables we want  
  
combined <- select(combined, Id, SalePrice, MSZoning, LotArea, LotConfig, HouseStyle, OverallCond, YearBuilt, YearRemodAdd, BsmtCond, HeatingQC, CentralAir, FullBath, BedroomAbvGr, KitchenAbvGr, GarageCars, GarageType, PavedDrive, Fence, MoSold, YrSold, SaleInRecession)

#One-hot variable encoding  
  
combined <- dummy\_cols(combined, remove\_first\_dummy = FALSE, remove\_selected\_columns = TRUE)

#Split combined data back into train and test sets  
  
train <- combined[1:1460,]  
test <- combined[1461:2919,]

Make training and validation subsets from the original training dataset.

train\_set = train %>%  
 sample\_frac(0.75)  
  
test\_set = train %>%  
 setdiff(train\_set)  
  
test\_set\_price\_preds <- select(test\_set, Id, SalePrice)  
  
train\_y <- as.integer(train\_set$SalePrice) - 2  
test\_y <- as.integer(test\_set$SalePrice) - 2  
train\_x <- train\_set %>% select(-c(SalePrice, Id))  
test\_x <- test\_set %>% select(-c(SalePrice, Id))

##Baseline modeling: Multiple regression

We can start with a basic multiple regression model as a baseline for comparison.

lm <- lm(SalePrice ~ . + -(Id), data = train\_set)  
summary(lm)

##   
## Call:  
## lm(formula = SalePrice ~ . + -(Id), data = train\_set)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -153603 -28282 -5924 19659 433944   
##   
## Coefficients: (15 not defined because of singularities)  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -2.700e+06 4.763e+06 -0.567 0.570904   
## Id -3.243e+00 3.605e+00 -0.900 0.368549   
## LotArea 1.168e+00 1.607e-01 7.271 7.01e-13 \*\*\*  
## YearBuilt 2.518e+02 1.118e+02 2.253 0.024468 \*   
## YearRemodAdd 2.956e+02 1.193e+02 2.479 0.013338 \*   
## FullBath 2.248e+04 3.999e+03 5.623 2.42e-08 \*\*\*  
## BedroomAbvGr 1.626e+03 2.333e+03 0.697 0.485997   
## KitchenAbvGr -2.553e+04 8.042e+03 -3.174 0.001545 \*\*   
## GarageCars 4.748e+04 3.116e+03 15.238 < 2e-16 \*\*\*  
## YrSold 8.640e+02 2.373e+03 0.364 0.715883   
## MSZoning\_C -3.628e+04 1.880e+04 -1.930 0.053896 .   
## MSZoning\_FV 1.176e+03 9.098e+03 0.129 0.897208   
## MSZoning\_RH 1.581e+04 1.380e+04 1.145 0.252336   
## MSZoning\_RL 1.610e+04 5.027e+03 3.203 0.001400 \*\*   
## MSZoning\_RM NA NA NA NA   
## MSZoning\_NA NA NA NA NA   
## LotConfig\_Corner 5.731e+02 4.063e+03 0.141 0.887852   
## LotConfig\_CulDSac 7.922e+02 6.138e+03 0.129 0.897329   
## LotConfig\_FR2 -1.721e+04 8.481e+03 -2.029 0.042721 \*   
## LotConfig\_FR3 -1.253e+04 2.890e+04 -0.434 0.664662   
## LotConfig\_Inside NA NA NA NA   
## HouseStyle\_1.5Fin 2.533e+04 8.680e+03 2.918 0.003594 \*\*   
## HouseStyle\_1.5Unf 1.831e+04 1.855e+04 0.987 0.323845   
## HouseStyle\_1Story 9.999e+03 7.153e+03 1.398 0.162462   
## HouseStyle\_2.5Fin 5.308e+04 2.036e+04 2.606 0.009283 \*\*   
## HouseStyle\_2.5Unf 5.278e+04 1.896e+04 2.784 0.005467 \*\*   
## HouseStyle\_2Story 1.678e+04 7.571e+03 2.217 0.026851 \*   
## HouseStyle\_SFoyer -3.322e+03 1.159e+04 -0.287 0.774460   
## HouseStyle\_SLvl NA NA NA NA   
## OverallCond\_1 NA NA NA NA   
## OverallCond\_2 9.651e+03 3.262e+04 0.296 0.767426   
## OverallCond\_3 -5.454e+04 1.961e+04 -2.782 0.005502 \*\*   
## OverallCond\_4 -4.484e+04 1.572e+04 -2.853 0.004413 \*\*   
## OverallCond\_5 -3.726e+04 1.382e+04 -2.697 0.007106 \*\*   
## OverallCond\_6 -3.462e+04 1.374e+04 -2.520 0.011878 \*   
## OverallCond\_7 -2.558e+04 1.351e+04 -1.894 0.058557 .   
## OverallCond\_8 -2.699e+04 1.437e+04 -1.879 0.060581 .   
## OverallCond\_9 NA NA NA NA   
## BsmtCond\_Fa -2.308e+03 9.722e+03 -0.237 0.812395   
## BsmtCond\_Gd 4.044e+03 7.470e+03 0.541 0.588358   
## BsmtCond\_NoBsmt -3.246e+04 9.686e+03 -3.351 0.000834 \*\*\*  
## BsmtCond\_Po -5.496e+04 5.841e+04 -0.941 0.346999   
## BsmtCond\_TA NA NA NA NA   
## HeatingQC\_Ex 1.928e+04 4.169e+03 4.625 4.21e-06 \*\*\*  
## HeatingQC\_Fa 9.065e+02 9.413e+03 0.096 0.923298   
## HeatingQC\_Gd 6.019e+02 4.837e+03 0.124 0.901006   
## HeatingQC\_Po NA NA NA NA   
## HeatingQC\_TA NA NA NA NA   
## CentralAir\_N 2.543e+03 7.676e+03 0.331 0.740443   
## CentralAir\_Y NA NA NA NA   
## GarageType\_Attchd -3.446e+04 9.330e+03 -3.694 0.000232 \*\*\*  
## GarageType\_Detchd -5.110e+04 9.154e+03 -5.583 3.02e-08 \*\*\*  
## GarageType\_NoGarage NA NA NA NA   
## PavedDrive\_NotPaved -8.434e+02 6.674e+03 -0.126 0.899463   
## PavedDrive\_Paved NA NA NA NA   
## Fence\_Fence -9.893e+02 4.011e+03 -0.247 0.805217   
## Fence\_NoFence NA NA NA NA   
## MoSold\_Apr -8.278e+03 8.614e+03 -0.961 0.336759   
## MoSold\_Aug -4.366e+03 8.661e+03 -0.504 0.614331   
## MoSold\_Dec -5.380e+03 1.027e+04 -0.524 0.600426   
## MoSold\_Feb -2.386e+04 1.060e+04 -2.251 0.024605 \*   
## MoSold\_Jan -1.761e+04 1.042e+04 -1.690 0.091369 .   
## MoSold\_Jul -4.893e+03 8.089e+03 -0.605 0.545338   
## MoSold\_Jun -1.140e+04 7.963e+03 -1.432 0.152493   
## MoSold\_Mar -3.686e+03 8.974e+03 -0.411 0.681367   
## MoSold\_May -9.535e+03 8.239e+03 -1.157 0.247397   
## MoSold\_Nov -1.695e+03 9.492e+03 -0.179 0.858303   
## MoSold\_Oct -1.343e+04 9.138e+03 -1.470 0.141961   
## MoSold\_Sep NA NA NA NA   
## SaleInRecession\_No 3.916e+03 6.219e+03 0.630 0.528990   
## SaleInRecession\_Yes NA NA NA NA   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 48720 on 1039 degrees of freedom  
## Multiple R-squared: 0.6278, Adjusted R-squared: 0.6081   
## F-statistic: 31.86 on 55 and 1039 DF, p-value: < 2.2e-16

pred\_y\_lm = predict(lm, test\_set)  
  
test\_set\_price\_preds$lm\_prices <- pred\_y\_lm  
test\_set\_price\_preds$lm\_prices\_error <- abs(test\_set\_price\_preds$SalePrice - test\_set\_price\_preds$lm\_prices)  
test\_set\_price\_preds$lm\_accuracy <- 1 - (test\_set\_price\_preds$lm\_prices\_error / test\_set\_price\_preds$SalePrice)  
  
rmse\_lm = RMSE(test\_y, pred\_y\_lm)  
cat("RMSE for lm: ", rmse\_lm)

## RMSE for lm: 52316.64

As expected, it produces mediocre results, with many insignificant variables and a low R^2 value.

##Advanced modeling: Lasso regression and gradient boosted decision trees

Next, we can try using lasso regression, which addresses variables that exhibit multicolinearity or have little to no influence on SalePrice.

y <- train\_set$SalePrice  
x <- as.matrix(select(train\_set, -(Id), -(SalePrice)))  
  
cv\_model <- cv.glmnet(x, y, alpha = 1)  
  
#find optimal lambda value that minimizes test MSE  
  
best\_lambda <- cv\_model$lambda.min  
best\_lambda

## [1] 905.5206

#find coefficients of best model  
  
best\_model <- glmnet(x, y, alpha = 1, lambda = best\_lambda)  
coef(best\_model)

## 70 x 1 sparse Matrix of class "dgCMatrix"  
## s0  
## (Intercept) -9.282498e+05  
## LotArea 1.102949e+00  
## YearBuilt 1.132130e+02  
## YearRemodAdd 3.852060e+02  
## FullBath 2.321042e+04  
## BedroomAbvGr 1.794125e+03  
## KitchenAbvGr -2.078580e+04  
## GarageCars 4.486016e+04  
## YrSold .   
## MSZoning\_C -2.576386e+04  
## MSZoning\_FV .   
## MSZoning\_RH 3.061001e+03  
## MSZoning\_RL 1.165578e+04  
## MSZoning\_RM -2.243328e+03  
## MSZoning\_NA .   
## LotConfig\_Corner .   
## LotConfig\_CulDSac .   
## LotConfig\_FR2 -1.154244e+04  
## LotConfig\_FR3 .   
## LotConfig\_Inside .   
## HouseStyle\_1.5Fin 4.334573e+03  
## HouseStyle\_1.5Unf .   
## HouseStyle\_1Story -4.899198e+03  
## HouseStyle\_2.5Fin 2.518624e+04  
## HouseStyle\_2.5Unf 1.852584e+04  
## HouseStyle\_2Story .   
## HouseStyle\_SFoyer -1.234995e+04  
## HouseStyle\_SLvl -1.164142e+04  
## OverallCond\_1 .   
## OverallCond\_2 1.425278e+04  
## OverallCond\_3 -1.176771e+04  
## OverallCond\_4 -6.750923e+03  
## OverallCond\_5 .   
## OverallCond\_6 .   
## OverallCond\_7 4.401183e+03  
## OverallCond\_8 .   
## OverallCond\_9 2.361743e+04  
## BsmtCond\_Fa .   
## BsmtCond\_Gd 1.937067e+03  
## BsmtCond\_NoBsmt -2.976829e+04  
## BsmtCond\_Po .   
## BsmtCond\_TA .   
## HeatingQC\_Ex 1.863115e+04  
## HeatingQC\_Fa .   
## HeatingQC\_Gd .   
## HeatingQC\_Po .   
## HeatingQC\_TA -3.214094e+02  
## CentralAir\_N .   
## CentralAir\_Y .   
## GarageType\_Attchd .   
## GarageType\_Detchd -1.799080e+04  
## GarageType\_NoGarage 2.413985e+04  
## PavedDrive\_NotPaved .   
## PavedDrive\_Paved .   
## Fence\_Fence .   
## Fence\_NoFence .   
## MoSold\_Apr .   
## MoSold\_Aug 6.900013e+02  
## MoSold\_Dec .   
## MoSold\_Feb -1.054703e+04  
## MoSold\_Jan -5.870437e+03  
## MoSold\_Jul 1.412937e+03  
## MoSold\_Jun -1.276423e+03  
## MoSold\_Mar 1.998696e+03  
## MoSold\_May .   
## MoSold\_Nov 1.036153e+03  
## MoSold\_Oct -2.229808e+03  
## MoSold\_Sep 4.076101e+03  
## SaleInRecession\_No 6.304619e+02  
## SaleInRecession\_Yes -3.434003e-10

pred\_y\_lasso <- predict(best\_model, as.matrix(test\_x))  
  
test\_set\_price\_preds$lasso\_prices <- pred\_y\_lasso  
test\_set\_price\_preds$lasso\_prices\_error <- abs(test\_set\_price\_preds$SalePrice - test\_set\_price\_preds$lasso\_prices)  
test\_set\_price\_preds$lasso\_accuracy <- 1 - (test\_set\_price\_preds$lasso\_prices\_error / test\_set\_price\_preds$SalePrice)  
  
rmse\_lasso = RMSE(test\_y, pred\_y\_lasso)  
cat("RMSE for lasso: ", rmse\_lasso)

## RMSE for lasso: 52159.14

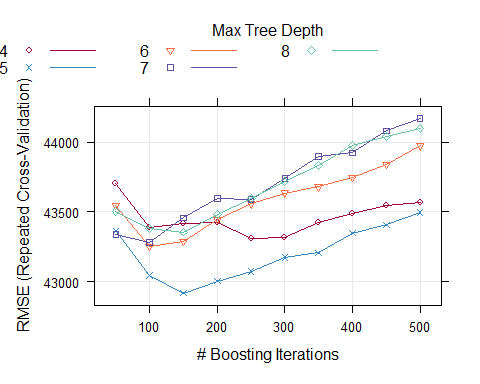
Next, let’s try using gradient boosted decision tree models.

Let’s first try stochastic gradient boosted decision trees. This method takes quite a while to run, as it uses repeated cross-validation on many different levels of tree depth to find the best possible model.

#Just as a warning: this code block takes a while to run.  
  
fitControl <- trainControl(method = "repeatedcv",  
 number = 10,  
 repeats = 10,  
 allowParallel = TRUE  
 )  
  
gbmGrid <- expand.grid(interaction.depth = c(4, 5, 6, 7, 8),  
 n.trees = (1:10)\*50,   
 shrinkage = 0.1,  
 n.minobsinnode = 20  
 )  
  
gbm\_model <- train(SalePrice ~ . + -(Id), data = train\_set,   
 method = "gbm",   
 trControl = fitControl,  
 verbose = FALSE,  
 tuneGrid = gbmGrid  
 )  
gbm\_model

## Stochastic Gradient Boosting   
##   
## 1095 samples  
## 70 predictor  
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 10 times)   
## Summary of sample sizes: 986, 985, 986, 984, 986, 985, ...   
## Resampling results across tuning parameters:  
##   
## interaction.depth n.trees RMSE Rsquared MAE   
## 4 50 43702.81 0.6880482 28963.04  
## 4 100 43389.19 0.6927030 28777.52  
## 4 150 43419.50 0.6928765 28866.61  
## 4 200 43421.75 0.6933047 28945.43  
## 4 250 43311.16 0.6949583 28907.45  
## 4 300 43315.75 0.6949050 28967.14  
## 4 350 43421.88 0.6934489 29054.47  
## 4 400 43487.42 0.6924954 29159.49  
## 4 450 43547.81 0.6918359 29192.57  
## 4 500 43567.38 0.6917298 29214.59  
## 5 50 43362.82 0.6934387 28644.00  
## 5 100 43043.28 0.6980262 28510.29  
## 5 150 42916.87 0.7001708 28584.16  
## 5 200 43003.56 0.6992144 28697.82  
## 5 250 43071.36 0.6982740 28790.77  
## 5 300 43174.33 0.6971357 28885.83  
## 5 350 43205.38 0.6968032 28958.69  
## 5 400 43347.25 0.6949705 29094.03  
## 5 450 43405.78 0.6944637 29181.81  
## 5 500 43495.93 0.6935106 29261.38  
## 6 50 43546.29 0.6909460 28712.73  
## 6 100 43252.23 0.6949116 28615.35  
## 6 150 43286.89 0.6946993 28817.28  
## 6 200 43446.09 0.6927613 28963.20  
## 6 250 43557.35 0.6917759 29072.38  
## 6 300 43634.03 0.6907939 29141.05  
## 6 350 43678.75 0.6902280 29217.56  
## 6 400 43745.66 0.6895082 29267.36  
## 6 450 43839.80 0.6885210 29365.87  
## 6 500 43973.86 0.6869547 29456.58  
## 7 50 43339.36 0.6934486 28485.21  
## 7 100 43280.50 0.6947182 28572.06  
## 7 150 43456.32 0.6923683 28782.07  
## 7 200 43595.76 0.6908047 28967.65  
## 7 250 43588.15 0.6907401 29063.35  
## 7 300 43741.22 0.6887111 29187.71  
## 7 350 43896.09 0.6867794 29294.04  
## 7 400 43922.88 0.6867222 29386.95  
## 7 450 44080.76 0.6846644 29530.34  
## 7 500 44169.30 0.6835590 29641.16  
## 8 50 43502.44 0.6907216 28553.35  
## 8 100 43381.81 0.6924624 28651.85  
## 8 150 43355.35 0.6931682 28783.91  
## 8 200 43479.85 0.6919659 28931.82  
## 8 250 43592.85 0.6910488 29107.58  
## 8 300 43714.46 0.6893170 29258.80  
## 8 350 43835.01 0.6875902 29410.00  
## 8 400 43979.29 0.6860190 29535.22  
## 8 450 44041.98 0.6852008 29627.82  
## 8 500 44100.25 0.6844638 29721.95  
##   
## Tuning parameter 'shrinkage' was held constant at a value of 0.1  
##   
## Tuning parameter 'n.minobsinnode' was held constant at a value of 20  
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were n.trees = 150, interaction.depth =  
## 5, shrinkage = 0.1 and n.minobsinnode = 20.

trellis.par.set(caretTheme())  
plot(gbm\_model)



We will also see how well the best model from this process performs.

pred\_y\_gbm = predict(gbm\_model, test\_set)  
  
test\_set\_price\_preds$gbm\_prices <- pred\_y\_gbm  
test\_set\_price\_preds$gbm\_prices\_error <- abs(test\_set\_price\_preds$SalePrice - test\_set\_price\_preds$gbm\_prices)  
test\_set\_price\_preds$gbm\_accuracy <- 1 - (test\_set\_price\_preds$gbm\_prices\_error / test\_set\_price\_preds$SalePrice)  
  
rmse\_gbm = RMSE(test\_y, pred\_y\_gbm)  
cat("RMSE for stochastic gbm: ", rmse\_gbm)

## RMSE for stochastic gbm: 43370.38

The best option seems to be using a max depth of 4.

We will use this information for another gradient boosted decision tree model type using xgboost since it’s far faster than a gbm model.

First, we can use xgboost cross-validation to find the best number of iterations.

xgb\_train = xgb.DMatrix(data = as.matrix(train\_x), label = train\_y)  
xgb\_test = xgb.DMatrix(data = as.matrix(test\_x), label = test\_y)  
  
params <- list(  
 booster = "gbtree",  
 eta = 0.01,  
 max\_depth = as.integer(gbm\_model$bestTune$interaction.depth),  
 gamma = 5,  
 subsample = 0.5  
)  
  
xgbcv <- xgb.cv(  
 params = params,   
 data = xgb\_train,   
 nrounds = 1500,   
 nfold = 5,  
 prediction = TRUE,  
 showsd = TRUE,   
 stratified = TRUE,   
 print\_every\_n = 50,   
 early\_stopping\_rounds = 50,   
 maximize = FALSE  
 )

## [1] train-rmse:194734.870345+1372.476514 test-rmse:194647.544208+5474.311040   
## Multiple eval metrics are present. Will use test\_rmse for early stopping.  
## Will train until test\_rmse hasn't improved in 50 rounds.  
##   
## [51] train-rmse:124221.692019+1003.032879 test-rmse:125056.869564+5956.775676   
## [101] train-rmse:82507.185876+781.564650 test-rmse:84913.293249+6504.869726   
## [151] train-rmse:58195.738916+670.574606 test-rmse:62752.366838+6531.822995   
## [201] train-rmse:44395.270413+546.312718 test-rmse:51388.561662+5954.774886   
## [251] train-rmse:36724.548181+515.895034 test-rmse:46015.490478+5144.466157   
## [301] train-rmse:32355.435272+505.181065 test-rmse:43577.043285+4298.307705   
## [351] train-rmse:29677.758450+534.094079 test-rmse:42465.354117+3727.707814   
## [401] train-rmse:27901.548599+524.215030 test-rmse:41878.283139+3220.915578   
## [451] train-rmse:26642.418127+519.309242 test-rmse:41679.389211+2907.263798   
## [501] train-rmse:25563.243970+529.784197 test-rmse:41507.827012+2644.687667   
## [551] train-rmse:24641.695579+516.337239 test-rmse:41404.130853+2431.621627   
## [601] train-rmse:23811.840234+464.101113 test-rmse:41338.638775+2322.422145   
## [651] train-rmse:23029.924599+443.985276 test-rmse:41302.568476+2198.257880   
## [701] train-rmse:22272.246995+426.690531 test-rmse:41285.818188+2122.433182   
## [751] train-rmse:21567.070335+395.013369 test-rmse:41281.621635+2033.594930   
## [801] train-rmse:20897.333624+389.646012 test-rmse:41321.460292+1966.875608   
## Stopping. Best iteration:  
## [756] train-rmse:21500.955272+403.709164 test-rmse:41272.303157+2028.636140

print(xgbcv$best\_iteration)

## [1] 756

min(xgbcv$evaluation\_log$test\_rmse\_mean)

## [1] 41272.3

We will use the best number of iterations in an xgboost model. Using this model, we can predict the SalePrice in the xgb\_test set and check our error.

xgb\_model <- xgb.train(   
 params = params,  
 data = xgb\_train,  
 nrounds = xgbcv$best\_iteration  
)  
  
xgb\_model

## ##### xgb.Booster  
## raw: 1.5 Mb   
## call:  
## xgb.train(params = params, data = xgb\_train, nrounds = xgbcv$best\_iteration)  
## params (as set within xgb.train):  
## booster = "gbtree", eta = "0.01", max\_depth = "5", gamma = "5", subsample = "0.5", validate\_parameters = "TRUE"  
## xgb.attributes:  
## niter  
## callbacks:  
## cb.print.evaluation(period = print\_every\_n)  
## # of features: 69   
## niter: 756  
## nfeatures : 69

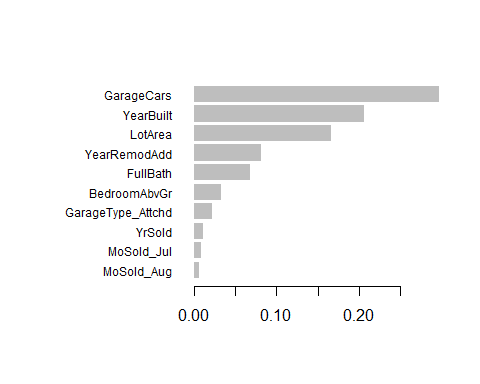
pred\_y\_xgb = predict(xgb\_model, xgb\_test)  
  
test\_set\_price\_preds$xgb\_prices <- pred\_y\_xgb  
test\_set\_price\_preds$xgb\_prices\_error <- abs(test\_set\_price\_preds$SalePrice - test\_set\_price\_preds$xgb\_prices)  
test\_set\_price\_preds$xgb\_accuracy <- 1 - (test\_set\_price\_preds$xgb\_prices\_error / test\_set\_price\_preds$SalePrice)  
  
rmse\_xgb = RMSE(test\_y, pred\_y\_xgb)  
cat("RMSE for xgb: ", rmse\_xgb)

## RMSE for xgb: 40290.59

importance\_matrix <- xgb.importance(  
 feature\_names = colnames(xgb\_train),   
 model = xgb\_model  
)  
importance\_matrix

## Feature Gain Cover Frequency  
## 1: GarageCars 2.970793e-01 8.413065e-02 0.0453935899  
## 2: YearBuilt 2.055048e-01 1.254247e-01 0.1329623173  
## 3: LotArea 1.659285e-01 2.714014e-01 0.2828215100  
## 4: YearRemodAdd 8.178932e-02 1.068672e-01 0.0943408878  
## 5: FullBath 6.770118e-02 6.141320e-02 0.0326538823  
## 6: BedroomAbvGr 3.277196e-02 6.913387e-02 0.0564570202  
## 7: GarageType\_Attchd 2.188935e-02 1.830424e-02 0.0119350945  
## 8: YrSold 1.134965e-02 1.402357e-02 0.0382191230  
## 9: MoSold\_Jul 8.349183e-03 5.748318e-03 0.0113316347  
## 10: MoSold\_Aug 5.833134e-03 9.834348e-03 0.0120691967  
## 11: MoSold\_Jan 5.802439e-03 7.826789e-03 0.0114657369  
## 12: HeatingQC\_Ex 5.607889e-03 1.338389e-02 0.0130079120  
## 13: MoSold\_Mar 4.512700e-03 5.017185e-03 0.0082472844  
## 14: HouseStyle\_1Story 4.218074e-03 8.220476e-03 0.0136113719  
## 15: Fence\_Fence 3.929443e-03 2.187041e-03 0.0071744669  
## 16: MoSold\_Feb 3.908250e-03 5.241171e-03 0.0087836932  
## 17: LotConfig\_Inside 3.752570e-03 6.341049e-03 0.0089177954  
## 18: MoSold\_Nov 3.646236e-03 4.454775e-03 0.0085825399  
## 19: MSZoning\_RM 3.576481e-03 8.334915e-03 0.0087166421  
## 20: MoSold\_Oct 3.443206e-03 5.587909e-03 0.0072415180  
## 21: BsmtCond\_NoBsmt 3.340686e-03 8.975573e-03 0.0091189486  
## 22: KitchenAbvGr 3.229115e-03 3.039948e-03 0.0046935765  
## 23: OverallCond\_7 3.175183e-03 1.219207e-02 0.0103258683  
## 24: HouseStyle\_2Story 3.167168e-03 9.275362e-03 0.0075097224  
## 25: MoSold\_Dec 3.028728e-03 6.793911e-03 0.0097894596  
## 26: CentralAir\_N 2.944746e-03 6.655021e-03 0.0068392115  
## 27: MoSold\_Sep 2.621177e-03 6.251553e-03 0.0072415180  
## 28: LotConfig\_Corner 2.556797e-03 2.271158e-03 0.0088507443  
## 29: BsmtCond\_Gd 2.552998e-03 1.282001e-02 0.0069062626  
## 30: MoSold\_May 2.467864e-03 3.327021e-03 0.0075767735  
## 31: PavedDrive\_NotPaved 2.363934e-03 6.255465e-03 0.0067051093  
## 32: MoSold\_Jun 2.298037e-03 4.694411e-03 0.0082472844  
## 33: LotConfig\_CulDSac 2.282781e-03 5.312573e-03 0.0063028027  
## 34: OverallCond\_9 2.104140e-03 1.603993e-02 0.0052970363  
## 35: MSZoning\_C 2.072175e-03 4.251819e-03 0.0071074159  
## 36: OverallCond\_5 2.015377e-03 5.859333e-03 0.0071744669  
## 37: MSZoning\_RL 2.009478e-03 3.904592e-03 0.0052970363  
## 38: OverallCond\_6 1.934563e-03 5.668114e-03 0.0069733137  
## 39: BsmtCond\_TA 1.602157e-03 4.990287e-03 0.0045594743  
## 40: OverallCond\_4 1.577920e-03 7.728001e-03 0.0055652407  
## 41: OverallCond\_8 1.499345e-03 5.283230e-03 0.0046935765  
## 42: HeatingQC\_TA 1.383624e-03 1.805091e-03 0.0030843503  
## 43: MSZoning\_FV 1.330674e-03 3.632679e-03 0.0051629342  
## 44: LotConfig\_FR2 9.940973e-04 3.612139e-03 0.0042242189  
## 45: HouseStyle\_1.5Fin 9.763800e-04 3.200846e-03 0.0042242189  
## 46: HeatingQC\_Gd 6.417464e-04 1.333156e-03 0.0036207590  
## 47: OverallCond\_3 6.110023e-04 3.657621e-03 0.0036207590  
## 48: HouseStyle\_SLvl 5.819204e-04 2.163077e-03 0.0026820437  
## 49: MoSold\_Apr 5.801851e-04 1.266645e-03 0.0032184525  
## 50: LotConfig\_FR3 3.355193e-04 4.303658e-05 0.0006034598  
## 51: GarageType\_Detchd 2.362492e-04 6.059355e-04 0.0019444817  
## 52: HeatingQC\_Fa 1.756848e-04 8.460601e-04 0.0016762773  
## 53: HouseStyle\_SFoyer 1.383675e-04 1.110148e-04 0.0008046131  
## 54: OverallCond\_2 1.309062e-04 8.964325e-04 0.0012739708  
## 55: HouseStyle\_2.5Unf 1.239521e-04 9.702793e-04 0.0009387153  
## 56: HouseStyle\_2.5Fin 1.084193e-04 6.548407e-04 0.0003352555  
## 57: MSZoning\_RH 1.025545e-04 2.963656e-04 0.0008716642  
## 58: BsmtCond\_Fa 7.982769e-05 2.601757e-04 0.0004693577  
## 59: HouseStyle\_1.5Unf 2.031075e-05 1.027009e-04 0.0004023066  
## 60: BsmtCond\_Po 1.058007e-05 7.580307e-05 0.0001341022  
## Feature Gain Cover Frequency

xgb.plot.importance(importance\_matrix, top\_n = 10)



##Results: Generating predictions for Test dataset

Let’s calculate the accuracies of each model’s price predictions.

cat("lm mean error: $", mean(test\_set\_price\_preds$lm\_prices\_error), "\n")

## lm mean error: $ 33705.5

cat("lm accuracy:", (mean(test\_set\_price\_preds$lm\_accuracy))\*100, "%\n")

## lm accuracy: 81.04099 %

cat("lasso mean error: $", mean(test\_set\_price\_preds$lasso\_prices\_error), "\n")

## lasso mean error: $ 33364.79

cat("lasso accuracy:", (mean(test\_set\_price\_preds$lasso\_accuracy))\*100, "%\n")

## lasso accuracy: 81.3602 %

cat("gbm mean error: $", mean(test\_set\_price\_preds$gbm\_prices\_error), "\n")

## gbm mean error: $ 28643.43

cat("gbm accuracy:", (mean(test\_set\_price\_preds$gbm\_accuracy))\*100, "%\n")

## gbm accuracy: 83.53856 %

cat("xgb mean error: $", mean(test\_set\_price\_preds$xgb\_prices\_error), "\n")

## xgb mean error: $ 27500.84

cat("xgb accuracy:", (mean(test\_set\_price\_preds$xgb\_accuracy))\*100, "%\n")

## xgb accuracy: 84.32213 %

Now we can use our best model to generate SalePrice predictions for the test data.

test\_y\_final <- as.integer(test$Id) - 1  
test\_x\_final <- test %>% select(-c(Id, SalePrice))  
  
xgb\_test\_final <- xgb.DMatrix(data = as.matrix(test\_x\_final), label = test\_y\_final)  
  
final\_pred <- predict(xgb\_model, xgb\_test\_final)  
  
test\_price\_pred <- select(test, Id)  
test\_price\_pred$SalePrice <- final\_pred

Our final output is a data set with just test house ID and its predicted SalePrice.

write.csv(test\_price\_pred, "test\_preds.csv")