Russell Cain

Stat 488 - Dr. Matthews

Due: Sat. December 15th

Kaggle House Price Predictions

Introduction

Unfortunately, Webster's Dictionary does not have a definition for 'Kaggle'. Thankfully, this paper will familiarize the reader with a particular competition from Kaggle.com which is rather characteristic of the website as a whole. This competition revolves around predicting the Sale Price for 1459 houses given their 79 predictive variables, a prediction which can then be tested against a hidden answer key on the website. This paper will discuss the steps taken to construct that prediction, from data cleaning to handling missing data, arriving at the prediction and resulting score.

Data Cleaning

This data contains a few complications which need to first be tidied up before any meaningful prediction can be made. Before we get into those, we will first construct a megatraining dataset, combining the testing and training. This serves two purposes: First, any of the variable cleaning needed can now be applied to just one data frame. Second, when we need to impute missing data later, our educated guesses for the Null values will take into account far more information, able to pull on inter-column relationships from twice the number of rows.

The datasets `test` and `train` have been read in from .csv files downloaded from the competition's page. Unfortunately, the training dataset has a column that `test` does not have, the SalePrice of houses already sold. We will store these off to the side, for safe keeping, while we combine all other columns of the training dataset with the testing data.

```r
train_SP <- train$SalePrice
train$SalePrice <- NULL
megatrain <- rbind(train,test)
```

While playing around and attempting to fit a Random Forest on what I had believed was clean data, I learned that none of our column names could begin with a number. This is the case with variables which describe the first and second floor and a particular type of porch which allows you to enjoy three of the four seasons. Let us take care of that right quick.

```r
names(megatrain)[names(megatrain) == '1stFlrSF'] <- 'FirstFlrSF'
names(megatrain)[names(megatrain) == '2ndFlrSF'] <- 'SecondFlrSF'
names(megatrain)[names(megatrain) == '3SsnPorch'] <- 'ThreeSeasonPorch'
```

Reviewing the variable descriptions, included on the competition's website, we see that many of the variables are factors which discuss quality (ranging from Excellent to Poor). As a the competition calls for the best final prediction, rather than an inference of which variables contribute to price in certain ways, we can reclassify these character factors as numeric rankings from 0 to 5, or however many distinct categories there are. This, to me, is a cleaner way of dealing with factors.

To do this, we will need to grab all of the columns which are classified as "character" and first convert them to factors, grouping by their ranking. Once this is done, we can create a list of the unique different factor levels and convert them to integer values. We will experiment and display the output on one example variable, Condition 1, which has each house classified in one of the following categories: Artery,Feedr, Norm, PosA, PosN, RRAe, RRAn, RRNe, and RRNn.

```r
class(megatrain$Condition1)
summary(as.factor(megatrain$Condition1))
sort(unique(megatrain$Condition1)) #The various categories
megatrain$Condition1 <- as.integer(factor(megatrain$Condition1),
                                   levels = sort(unique(megatrain$Condition1)))
summary(megatrain$Condition1) #Integer values ranging from 1 to 9. Nice.
```

This looks quite promising, so let's write a for loop that does this to every character variable in our `megatrain` dataset.

```r
for(i in colnames(megatrain)){
  if(class(megatrain[[i]]) == 'character'){
    megatrain[[i]] <- as.integer(factor(megatrain[[i]]),
                                  levels = sort(unique(megatrain[[i]])))
  }
}
```

Beautiful! Now all of our categorical variables can parade around as continuous variables while retaining their meaning. I think we are ready to handle the missing data, which occurs in 34 of our columns.

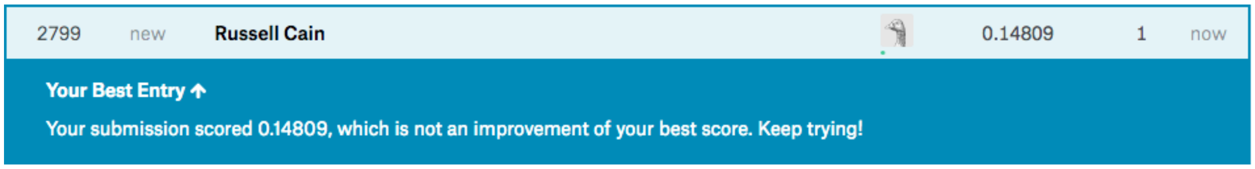
Single Imputation & First Entry

Pulling from some fine instruction received this semester, we will use the principle that although guesses from many people may not be accurate, their average will be quite nice. This will be applied by running multiple imputations, storing their results in a matrix, and constructing our final dataframe from the average of the many imputations, hoping that this is quite accurate to the true value of the missing values. To best display this principle, we will first run a single imputation, fit a random forest, and see what score that enjoy gains us.

```r
set.seed(2424)
imputed = complete(mice(megatrain,m=1))
imputed_train <- imputed[1:1460,]
imputed_train$SalePrice <- train_SP
imputed_test <- imputed[1461: 2919,]
```

Below is our first attempt at a prediction, just to see where a Random Forest fitted on data with just one MICE imputation would leave us. This paper will continue on to discover ways to improve the score, but it nice to establish a lower benchmark by which to compare future entries. As already mentioned, this can be done by averaging multiple imputations' predictions.

```r
rf1 <- randomForest(SalePrice ~ ., data = imputed_train)
prediction1 <- predict(rf1, imputed_test)
prediction1 <- cbind(test$Id, prediction1)
colnames(prediction1) <- c('Id', 'SalePrice')
write.csv(prediction1,file = 'FirstPrediction.csv', row.names = F)
```



2799 new **Russell Cain** 0.14809 1 now

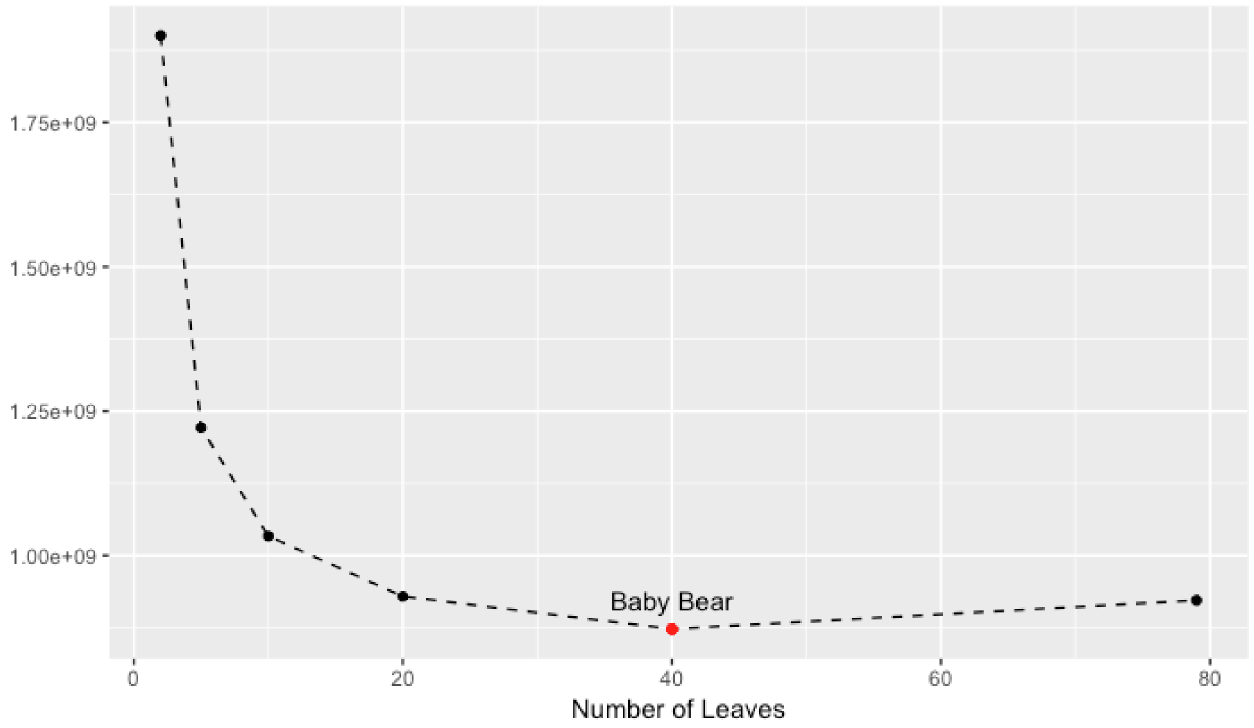
Your Best Entry ↑

Your submission scored 0.14809, which is not an improvement of your best score. Keep trying!

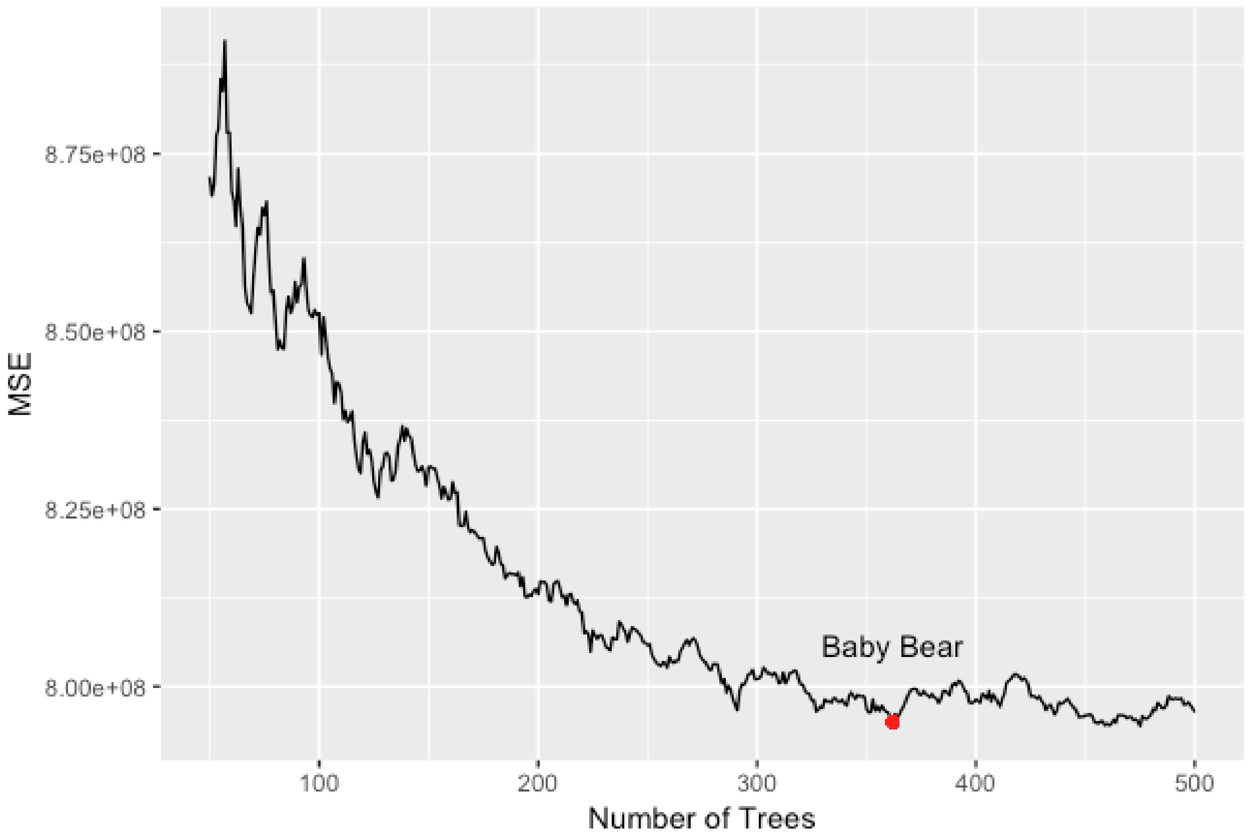
And keep trying we shall! Let's see where the averaged multiple imputation gets us.

Random Forest Cross Validation

Using the imputed data generated above, we will try to cross validate our random forest technique. Who knows, maybe we will even find a baby bear hanging out in one of the trees of this random forest! I think it is worth a shot.



We can see that our Baby Bear resides at setting our number of randomly sampled candidates, m, equal to 40. Therefore, for our remaining random forests, we will set the max number of nodes equal to 40. We see that the value of an additional tree in our forest drops off after we hit about 350 trees. That being said, as we have time to kill, we will let `randomForest()` to stick to the preset 500 trees.



Multiple Imputation & Second Entry

Below is a defined function which will help us generate random forest predictions for different missing data imputations. It is my understanding that, upon imputing the data, we should make a random forest model on the training portion of the megatrain-ing dataset and apply it to the testing portion of the newly imputed dataset. As we saw above, the baby bear for our random forest on this data, roughly speaking, is a forest which no more than 40 nodes.

```
multImp <- function(dataset, m = 5){
  imputation <- mice(dataset, m)
  predictions <- data.frame(ID = test$Id)
  totaldata <- complete(imputation,1)
  for(i in 1:m){
    randomforestdata <- complete(imputation, i)
    if(i != 1){
      totaldata <- totaldata + randomforestdata
    }
    rfd <- randomforestdata[1:1460,]
    rfd$SalePrice <- train_SP
    rf <- randomForest(SalePrice ~ ., data = rfd, maxnodes = 40)
    pred <- predict(rf, randomforestdata[1461:2919,])
    predictions[i+1] <- pred
  }
  predictions[,1] <- NULL
  totaldata <- totaldata/m
  filler <- (1:1460)*0
  totaldata$prediction <- c(filler,rowSums(predictions)/m)
  return(totaldata)
}

totaldata <- multImp(megatrain, 7)

finpred <- data.frame(test$Id, totaldata[1461:2919,]$prediction)
colnames(finpred) <- c('Id', 'SalePrice')
write.csv(finalprediction,file = 'BetterPrediction.csv', row.names = F, append = F)
```

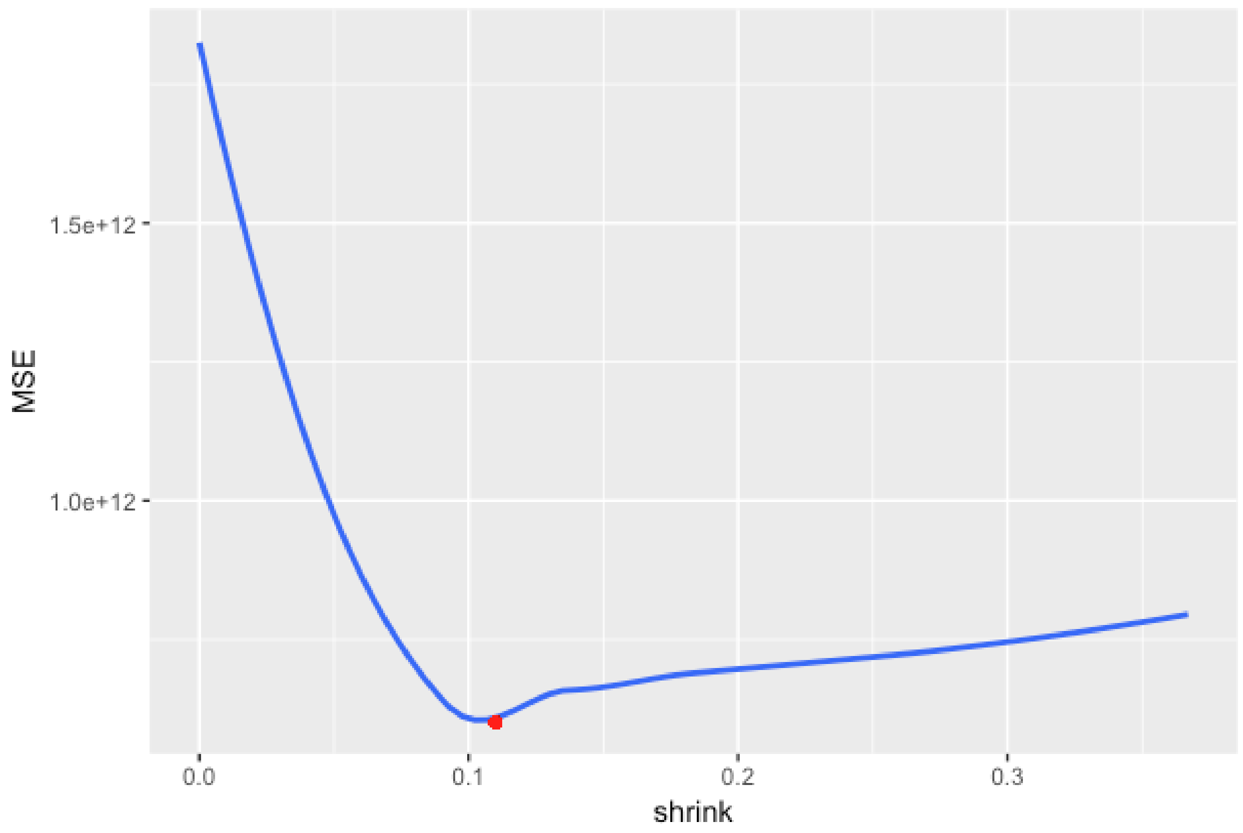
This refinement brings up our score a fair bit, but still leaves us in the 2,000s on the leaderboard, which I don't much like. To improve, we will try one last modeling attempt with boosting and see if this helps us break the top 1,999. Thankfully, our function above returns an averaged version of our imputed data, which will theoretically be a fairly reasonable estimate of the missing data.

Boosting

```
totaldata$prediction <- NULL
imputed_train <- totaldata[1:1460,]
imputed_train$SalePrice <- train_SP
imputed_test <- totaldata[1461: 2919,]
boostyboy <- gbm(SalePrice ~ . , data = imputed_train , distribution=
"gaussian" , n.trees = 5000)
newpred <- predict(boostyboy, imputed_test, n.trees = 5000)
newpred <- data.frame(test$Id, newpred)
colnames(newpred) <- c('Id', 'SalePrice')
write.csv(finalprediction,file = 'BetterPrediction.csv', row.names = F, append = F)
```

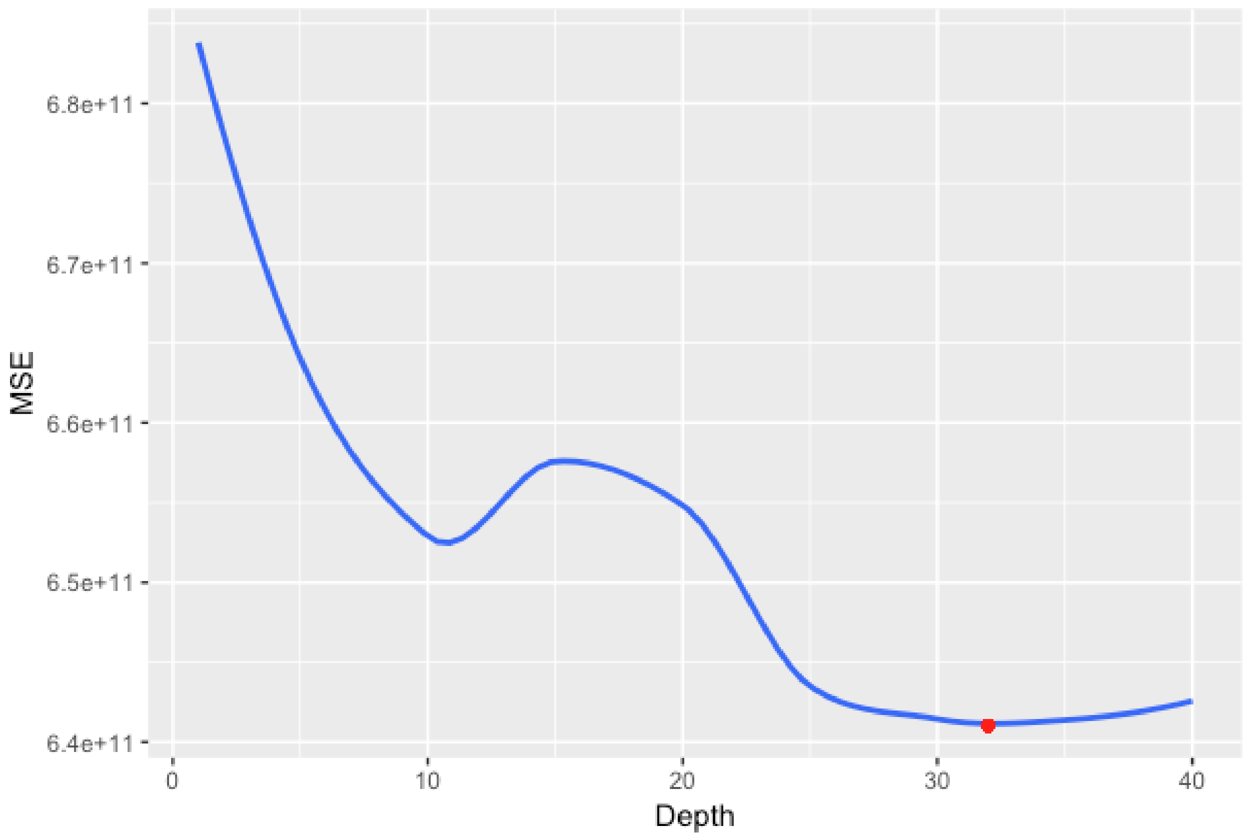
This wasn't much of an improvement! I suppose this reaffirms the section of our textbook which notes that a more complex modeling technique won't necessarily perform better than a well-fitted and cross-validated, albeit simpler, technique. For one last ditch effort to improve our score, we will try to cross validate the boosting model.

```
train_train <- imputed_train[1:730,]
train_test <- imputed_train[731:1460,]
gbm_results = data.frame(Shrinker=numeric(),
                         MSE = numeric())
for (i in seq(0.05, 0.2, by=.05)){
  gbm_temp <- gbm(SalePrice ~ ., data = train_train, distribution =
                    "gaussian", interaction.depth = 3,
                  cv.folds = 5, keep.data = TRUE,
                  n.cores = 2, shrinkage = i)
  Yhat <- predict(gbm_temp, train_test)
  mse <- sum((train_test$SalePrice - Yhat)^2)
gbm_results <- rbind(gbm_results, c(i,mse))
}
colnames(gbm_results) <- c("shrink", 'MSE')
ggplot(gbm_results, aes(x = shrink, y = MSE)) + geom_line()
```



It would appear that we should choose a shrinkage parameter of .115, which isn't very away from 0.1, but we will give it a try.

```
gbm_results = data.frame(Depth=numeric(),
                         MSE = numeric())
for (i in 1:7){
  gbm_temp <- gbm(SalePrice ~ ., data = train_train, distribution =
                    "gaussian", interaction.depth = i,
                  cv.folds = 5, keep.data = TRUE,
                  n.cores = 2, shrinkage = .115)
  Yhat <- predict(gbm_temp, train_test)
  mse <- sum((train_test$SalePrice - Yhat)^2)
  gbm_results <- rbind(gbm_results, c(i,mse))
}
colnames(gbm_results) <- c("Depth", 'MSE')
ggplot(gbm_results, aes(x = shrink, y = MSE)) + geom_line()
```



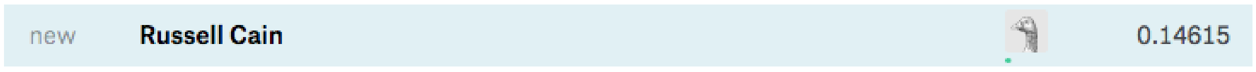
Just as important, we have now fit our depth parameter. It appears to be roughly 33.

```
gbm_fitboy <- gbm(SalePrice ~ ., data = train_train, distribution =
                    "gaussian", interaction.depth = 33,
                  cv.folds = 5, keep.data = TRUE,
                  n.cores = 2, shrinkage = 0.115)

newpred <- predict(gbm_fitboy, imputed_test)
newpred <- data.frame(test$Id, newpred)
colnames(newpred) <- c('Id', 'SalePrice')
write.csv(finalprediction,file = 'BestPrediction.csv', row.names = F, append = F)
```

Conclusion:

Somehow, none of this cross-validation nor the boosting appears to have helped. As it stands, the accurately fit random forest has been my best prediction. Although the description of my work ends here, I will certainly be continuing on outside of this report to improve my score. Areas of further development include assessing the variables' relationships to one another and perhaps engineering those a bit better. If these additions lead to any progress, I will document it and upload a new version. For the time being, I would like to think this is a solid description and attempt at a Kaggle contest, yielding a score better than some.



new Russell Cain 0.14615