Homework 6 - Alex Hyman

Alex Hyman

11/18/2018

## Introduction

Using the pixel intensity of hand drawn digits (0-9) can a supervised machine learning algorithm correctly classift which number is drawn? The data provided contains the pixel intensity for 42,000 different 28x28 images. Each of the images are a digit, 0-9 and each of the values provided are the pixel intensities of a specific pixel in the given image.

The generic preprocessing for this problem involved separating the labeled dataset into a training and test set. The training set contained 80% of the data and the test set contained the remaining 20% of the labeled data. The labels also needed to be converted into factors because this is not a regression problem, but a classification problem. The final training set contained 33,604 hand-drawn digits, while the test set contained 8,396 different hand-drawn digits.

library(dplyr)

library(tidyr)  
library(caret)

library(klaR)

library(rpart)  
library(rpart.plot)  
library(matrixStats)

library(arules)

library(plyr)

#Reading the training file  
data <- read.csv("train.csv")  
#seeing if any missing data  
sum(is.na(data))

## [1] 0

#converting the label to a factor  
data$label <- as.factor(data$label)  
#Reading the data that will be prdicted  
validation <- read.csv("test.csv")  
#getting dimensions of data  
dim(data)

## [1] 42000 785

set.seed(1000)  
#Getting the index for the training data  
train\_ix <- createDataPartition(data$label, p = 0.8, list = F)  
#Training data frame  
train\_df <- data[train\_ix, ]  
#Test data set  
test\_df <- data[-train\_ix, ]  
  
#getting new size of the train index  
dim(train\_df)

## [1] 33604 785

dim(test\_df)

## [1] 8396 785

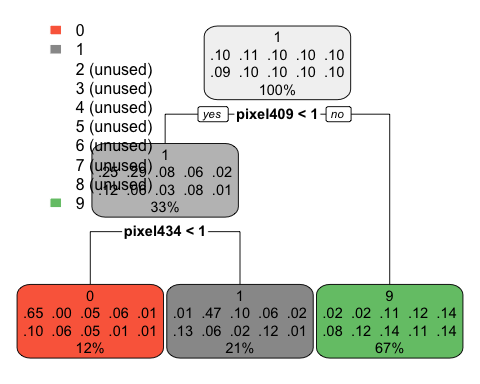
## Decision Tree

An initial decision tree model was created to classify the different hand-drawn digits. A 3-fold cross-validation was conducted on the training set and a cross-validation accuracy of 0.31 was found for the initial model. A plot of the decision tree model shows that the model only had three different leaf nodes, for digits 0, 1, and 9. This would only allow the model to correctly identify 30% of all the digits in the training set.

#Making the training parameters  
tc <- trainControl(method = "cv", number = 3)  
#intial decision tree model  
dt <- train(label ~ ., data = train\_df, trControl = tc, method = "rpart")  
#printing the decision tree  
print(dt)

## CART   
##   
## 33604 samples  
## 784 predictor  
## 10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'   
##   
## No pre-processing  
## Resampling: Cross-Validated (3 fold)   
## Summary of sample sizes: 22402, 22403, 22403   
## Resampling results across tuning parameters:  
##   
## cp Accuracy Kappa   
## 0.07358655 0.3161541 0.23681696  
## 0.09083601 0.2197044 0.12996575  
## 0.09177385 0.1381094 0.03260562  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.07358655.

#printing the decision tree  
rpart.plot(dt$finalModel)



The confusion matrix for the decision tree model shows the issue with only having three leaf nodes, While the performance is decent on the digits 0, 1, and 9, everything else is misclassified as it the model does not have any branches that take digits 2-8 into account. A newly tuned decision tree should be made that allows for more branches.

#Predicting the   
preds <- predict(dt, test\_df, "raw")  
#Confusion matrix  
confusionMatrix(preds, test\_df$label)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 674 2 53 60 8 105 59 63 9 14  
## 1 12 833 180 113 33 211 93 38 203 19  
## 2 0 0 0 0 0 0 0 0 0 0  
## 3 0 0 0 0 0 0 0 0 0 0  
## 4 0 0 0 0 0 0 0 0 0 0  
## 5 0 0 0 0 0 0 0 0 0 0  
## 6 0 0 0 0 0 0 0 0 0 0  
## 7 0 0 0 0 0 0 0 0 0 0  
## 8 0 0 0 0 0 0 0 0 0 0  
## 9 140 101 602 697 773 443 675 779 600 804  
##   
## Overall Statistics  
##   
## Accuracy : 0.2753   
## 95% CI : (0.2657, 0.2849)  
## No Information Rate : 0.1115   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.193   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.81598 0.88996 0.00000 0.0000 0.00000 0.0000  
## Specificity 0.95073 0.87909 1.00000 1.0000 1.00000 1.0000  
## Pos Pred Value 0.64374 0.48012 NaN NaN NaN NaN  
## Neg Pred Value 0.97932 0.98454 0.90055 0.8964 0.90305 0.9096  
## Prevalence 0.09838 0.11148 0.09945 0.1036 0.09695 0.0904  
## Detection Rate 0.08028 0.09921 0.00000 0.0000 0.00000 0.0000  
## Detection Prevalence 0.12470 0.20665 0.00000 0.0000 0.00000 0.0000  
## Balanced Accuracy 0.88335 0.88452 0.50000 0.5000 0.50000 0.5000  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.0000 0.0000 0.00000 0.96057  
## Specificity 1.0000 1.0000 1.00000 0.36367  
## Pos Pred Value NaN NaN NaN 0.14321  
## Neg Pred Value 0.9015 0.8952 0.90329 0.98814  
## Prevalence 0.0985 0.1048 0.09671 0.09969  
## Detection Rate 0.0000 0.0000 0.00000 0.09576  
## Detection Prevalence 0.0000 0.0000 0.00000 0.66865  
## Balanced Accuracy 0.5000 0.5000 0.50000 0.66212

Multple complexity parameter were attempted to find a better decision tree model to predict which class each of the hand-written digits belonged to, and the best parameter found was 0.0005. This value resulted in a cross-validation accuracy of 0.82, and an accuracy on the test set of 0.82. The similarity between the cross-validation accuracy and the test accuracy shows that the model was not overfit, and the model performed fairly well. There were not any pre-processing steps that were required when building the decision tree model, as the decision tree dies not expect any distributions. This decision tree model was too big to print.

#Adjusting the complexity parameter  
dt\_grid <- expand.grid("cp" = 0.0005)  
#New decision tree   
dt\_tuned <- train(label ~ ., data = train\_df, trControl = tc,   
 tuneGrid = dt\_grid, method = "rpart")  
#cross validation accuracy  
dt\_tuned

## CART   
##   
## 33604 samples  
## 784 predictor  
## 10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'   
##   
## No pre-processing  
## Resampling: Cross-Validated (3 fold)   
## Summary of sample sizes: 22402, 22403, 22403   
## Resampling results:  
##   
## Accuracy Kappa   
## 0.817224 0.7968291  
##   
## Tuning parameter 'cp' was held constant at a value of 5e-04

#PRedictinf with newly tuned decision tree  
preds\_tuned <- predict(dt\_tuned$finalModel, test\_df, type = "class")  
#Confusion matrix of new decision tree  
confusionMatrix(preds\_tuned, test\_df$label)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 752 1 26 4 6 23 24 22 5 5  
## 1 2 880 22 12 17 4 8 10 27 5  
## 2 13 11 677 45 12 12 28 28 30 9  
## 3 7 11 22 635 8 46 3 2 33 23  
## 4 6 4 19 12 659 18 20 12 11 40  
## 5 14 4 7 48 5 547 17 8 22 12  
## 6 5 3 18 15 19 32 688 0 15 3  
## 7 9 3 20 31 4 18 3 747 4 12  
## 8 13 9 10 46 22 27 20 7 627 14  
## 9 5 10 14 22 62 32 16 44 38 714  
##   
## Overall Statistics  
##   
## Accuracy : 0.8249   
## 95% CI : (0.8166, 0.833)  
## No Information Rate : 0.1115   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8054   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.91041 0.9402 0.81078 0.72989 0.80958 0.72069  
## Specificity 0.98468 0.9857 0.97514 0.97940 0.98127 0.98206  
## Pos Pred Value 0.86636 0.8916 0.78266 0.80380 0.82272 0.79971  
## Neg Pred Value 0.99017 0.9924 0.97902 0.96910 0.97959 0.97251  
## Prevalence 0.09838 0.1115 0.09945 0.10362 0.09695 0.09040  
## Detection Rate 0.08957 0.1048 0.08063 0.07563 0.07849 0.06515  
## Detection Prevalence 0.10338 0.1176 0.10303 0.09409 0.09540 0.08147  
## Balanced Accuracy 0.94754 0.9629 0.89296 0.85464 0.89543 0.85137  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.83192 0.84886 0.77217 0.85305  
## Specificity 0.98547 0.98616 0.97785 0.96785  
## Pos Pred Value 0.86216 0.87779 0.78868 0.74608  
## Neg Pred Value 0.98171 0.98237 0.97566 0.98347  
## Prevalence 0.09850 0.10481 0.09671 0.09969  
## Detection Rate 0.08194 0.08897 0.07468 0.08504  
## Detection Prevalence 0.09505 0.10136 0.09469 0.11398  
## Balanced Accuracy 0.90869 0.91751 0.87501 0.91045

## Naive Bayes

Three naive bayes models were also created to classify the hand-written digits. The preprocessing steps involved discretizing each pixel into three categories: “empty”, “edge”, and “mark”. Empty pixels were pixels that had a value of zero as in the pixel. Empty pixels were likely to be pixels that did not have any ink on the pixel Edge pixels were pixels that had a pixel value between 0 and 100. These pixels were likely pixels that were on the edge of a digit and were not necessarily required, but point where the ink ran into the next pixel. Marked pixels were the pixels that had a value between 100 and 256. These pixels were where the writer likely intended the to fully mark on the image.

#Getting only the labels  
label <- train\_df$label  
#Creating a new daa frame with only those columns  
train\_nb <- train\_df[,2:ncol(train\_df)]  
  
#Discretizing the columns  
for (col in colnames(train\_nb)){  
 train\_nb[,col] <- discretize(train\_nb[,col], method = "fixed", breaks = c(-1, 0.1, 100, 256),  
 labels = c("empty", "edge", "mark"))  
}  
  
#Final training data frame  
train\_nb <- data.frame(label = label, train\_nb)  
  
#Fixing the test data frame  
test\_nb <- test\_df[,2:ncol(test\_df)]  
#Discretizing the test set  
for (col in colnames(test\_nb)){  
 test\_nb[,col] <- discretize(test\_nb[,col], method = "fixed", breaks = c(-1, 0.1, 100, 256),  
 labels = c("empty", "edge", "mark"))  
}  
#getting test labels  
test\_label <- test\_df$label  
#creating the test data frame  
test\_nb <- data.frame(label = test\_label, test\_nb)

The first Naive Bayes model was created to not use a kernel, and to have a LaPlace factor of 1. The cross-validation for this set would take too long to determine a cross-validation accuracy, but the model had an accuracy of 0.83 on the test set.

#training the model  
nb <- NaiveBayes(label ~ ., train\_nb, fL = 1, usekernel = F)  
#Predicting the test set  
preds <- suppressWarnings(predict(nb, test\_nb, type = "raw"))  
#Confusion matrix  
confusionMatrix(preds$class, test\_df$label)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 745 0 11 9 4 16 17 4 8 9  
## 1 0 893 17 21 6 9 22 27 25 8  
## 2 5 9 684 44 6 9 13 11 10 2  
## 3 3 3 14 674 0 123 1 1 53 12  
## 4 3 1 17 2 669 23 5 17 8 66  
## 5 38 12 1 24 4 524 28 3 21 5  
## 6 10 4 32 11 10 20 737 1 7 0  
## 7 0 0 13 13 2 4 0 745 2 35  
## 8 22 14 41 47 22 9 4 20 644 29  
## 9 0 0 5 25 91 22 0 51 34 671  
##   
## Overall Statistics  
##   
## Accuracy : 0.8321   
## 95% CI : (0.8239, 0.84)  
## No Information Rate : 0.1115   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8133   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.90194 0.9541 0.81916 0.77471 0.82187 0.69038  
## Specificity 0.98970 0.9819 0.98558 0.97210 0.98127 0.98219  
## Pos Pred Value 0.90522 0.8687 0.86255 0.76244 0.82491 0.79394  
## Neg Pred Value 0.98930 0.9942 0.98014 0.97391 0.98088 0.96962  
## Prevalence 0.09838 0.1115 0.09945 0.10362 0.09695 0.09040  
## Detection Rate 0.08873 0.1064 0.08147 0.08028 0.07968 0.06241  
## Detection Prevalence 0.09802 0.1224 0.09445 0.10529 0.09659 0.07861  
## Balanced Accuracy 0.94582 0.9680 0.90237 0.87340 0.90157 0.83629  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.89117 0.84659 0.79310 0.80167  
## Specificity 0.98745 0.99082 0.97257 0.96984  
## Pos Pred Value 0.88582 0.91523 0.75587 0.74638  
## Neg Pred Value 0.98810 0.98219 0.97773 0.97786  
## Prevalence 0.09850 0.10481 0.09671 0.09969  
## Detection Rate 0.08778 0.08873 0.07670 0.07992  
## Detection Prevalence 0.09909 0.09695 0.10148 0.10707  
## Balanced Accuracy 0.93931 0.91871 0.88284 0.88575

A second Naive Bayes model was also created with the regularization parameter fL set to 50 to see if there was any overfitting in the problem. The same test set that had been used in the first Naive Bayes model was used to find the accuarcy on the test set, and an accuracy of 0.82 was acheived. This was a slightly worse accuracy than original naive bayes model scored. This would seem to suggest that the model is not overfit.

#training the model  
nb2 <- suppressWarnings(NaiveBayes(label ~., train\_nb, fL = 50))  
#Predicting the test set  
preds2 <- suppressWarnings(predict(nb2, test\_nb, type = "raw"))  
#Confusion matric for the test set  
confusionMatrix(preds2$class, test\_nb$label)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 742 0 11 10 3 16 17 5 8 9  
## 1 0 902 23 22 11 28 29 34 36 16  
## 2 5 7 667 40 5 7 11 12 9 2  
## 3 1 2 15 657 0 112 1 0 51 11  
## 4 3 1 21 1 660 22 6 17 8 66  
## 5 42 8 1 35 3 517 24 2 20 6  
## 6 12 4 32 10 11 22 736 1 7 0  
## 7 0 0 16 12 3 5 0 737 2 34  
## 8 20 12 44 54 19 8 3 17 635 24  
## 9 1 0 5 29 99 22 0 55 36 669  
##   
## Overall Statistics  
##   
## Accuracy : 0.8244   
## 95% CI : (0.8161, 0.8325)  
## No Information Rate : 0.1115   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8048   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.89831 0.9637 0.79880 0.75517 0.81081 0.68116  
## Specificity 0.98956 0.9733 0.98704 0.97436 0.98088 0.98154  
## Pos Pred Value 0.90378 0.8193 0.87190 0.77294 0.81988 0.78571  
## Neg Pred Value 0.98891 0.9953 0.97798 0.97177 0.97971 0.96873  
## Prevalence 0.09838 0.1115 0.09945 0.10362 0.09695 0.09040  
## Detection Rate 0.08838 0.1074 0.07944 0.07825 0.07861 0.06158  
## Detection Prevalence 0.09778 0.1311 0.09111 0.10124 0.09588 0.07837  
## Balanced Accuracy 0.94393 0.9685 0.89292 0.86476 0.89584 0.83135  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.88996 0.83750 0.78202 0.79928  
## Specificity 0.98692 0.99042 0.97350 0.96732  
## Pos Pred Value 0.88144 0.91100 0.75957 0.73035  
## Neg Pred Value 0.98796 0.98115 0.97659 0.97754  
## Prevalence 0.09850 0.10481 0.09671 0.09969  
## Detection Rate 0.08766 0.08778 0.07563 0.07968  
## Detection Prevalence 0.09945 0.09636 0.09957 0.10910  
## Balanced Accuracy 0.93844 0.91396 0.87776 0.88330

A third naive bayes model was created to see if using teh kernel denisty estimate would perform better than using the normal distribution. a Laplace factor of 1 was also used for this model. After using the model to predict on the test set, the model acheived an accuracy of 0.8321, the same accuracy acheived in the first naive bayes model. Becasue the normal distribution model acheived the same score, the first model was determinewd to be our best model

#training the model  
nb3 <- suppressWarnings(NaiveBayes(label ~., train\_nb, fL = 1, usekernel = T))  
#Predicting the test set  
preds3 <- suppressWarnings(predict(nb3, test\_nb, type = "raw"))  
#Confusion matric for the test set  
confusionMatrix(preds3$class, test\_nb$label)

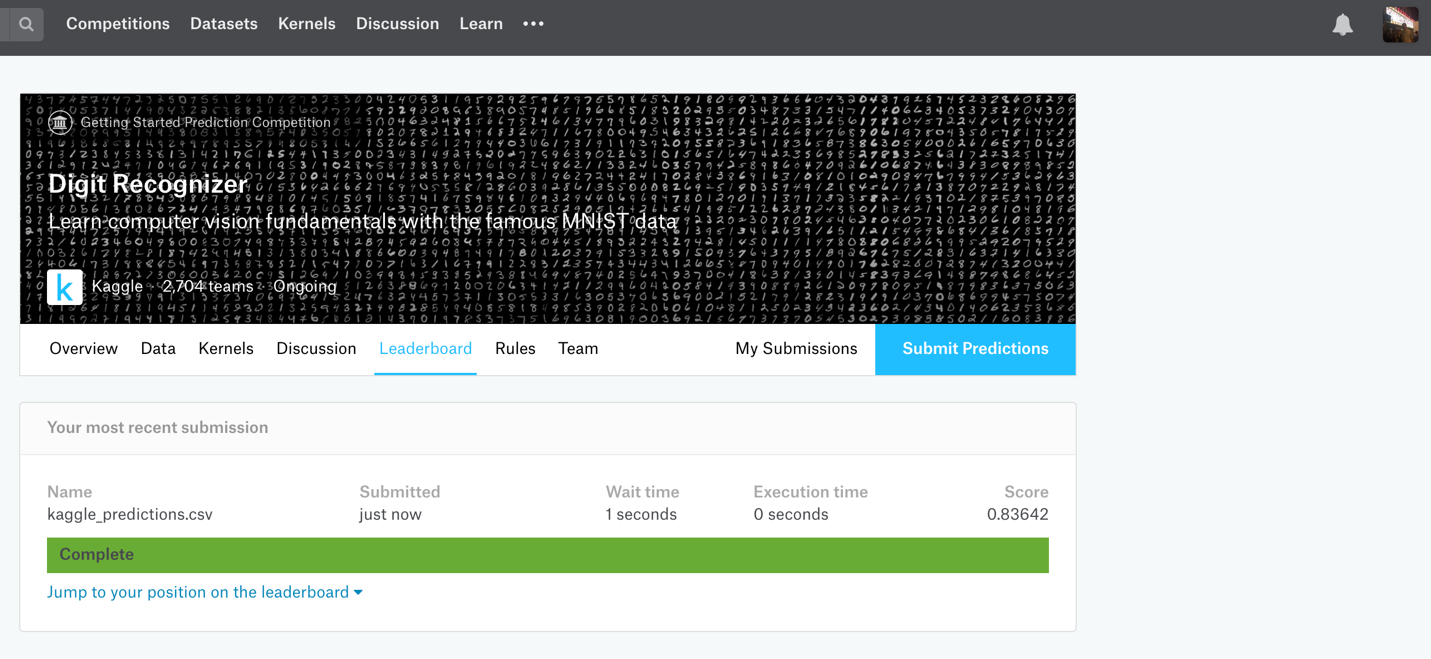
## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 745 0 11 9 4 16 17 4 8 9  
## 1 0 893 17 21 6 9 22 27 25 8  
## 2 5 9 684 44 6 9 13 11 10 2  
## 3 3 3 14 674 0 123 1 1 53 12  
## 4 3 1 17 2 669 23 5 17 8 66  
## 5 38 12 1 24 4 524 28 3 21 5  
## 6 10 4 32 11 10 20 737 1 7 0  
## 7 0 0 13 13 2 4 0 745 2 35  
## 8 22 14 41 47 22 9 4 20 644 29  
## 9 0 0 5 25 91 22 0 51 34 671  
##   
## Overall Statistics  
##   
## Accuracy : 0.8321   
## 95% CI : (0.8239, 0.84)  
## No Information Rate : 0.1115   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8133   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.90194 0.9541 0.81916 0.77471 0.82187 0.69038  
## Specificity 0.98970 0.9819 0.98558 0.97210 0.98127 0.98219  
## Pos Pred Value 0.90522 0.8687 0.86255 0.76244 0.82491 0.79394  
## Neg Pred Value 0.98930 0.9942 0.98014 0.97391 0.98088 0.96962  
## Prevalence 0.09838 0.1115 0.09945 0.10362 0.09695 0.09040  
## Detection Rate 0.08873 0.1064 0.08147 0.08028 0.07968 0.06241  
## Detection Prevalence 0.09802 0.1224 0.09445 0.10529 0.09659 0.07861  
## Balanced Accuracy 0.94582 0.9680 0.90237 0.87340 0.90157 0.83629  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.89117 0.84659 0.79310 0.80167  
## Specificity 0.98745 0.99082 0.97257 0.96984  
## Pos Pred Value 0.88582 0.91523 0.75587 0.74638  
## Neg Pred Value 0.98810 0.98219 0.97773 0.97786  
## Prevalence 0.09850 0.10481 0.09671 0.09969  
## Detection Rate 0.08778 0.08873 0.07670 0.07992  
## Detection Prevalence 0.09909 0.09695 0.10148 0.10707  
## Balanced Accuracy 0.93931 0.91871 0.88284 0.88575

## Performance Comparison

When comparing the two models, both performed similarly in terms of accuracy on the test set; however the naive bayes model with a Laplace factor of 1 and the normal distribution kernel had the best overall accuracy on the test set with an accuracy of 0.83. The biggest difference between the two models were that the Decision Tree model took longer to train, but the Naive Bayes model took longer to predict on the test set. The decision tree model likely took longer to train because there are so many different varaibles to parse through and split on. The Naive Bayes model took no time at all to train, but predictions took much longer. This is likely because there are only a set number of calculations the Naive Bayes model needs to work through, but having to conduct that calculation for each of the observations would take a while.

## Kaggle Results

Because the naive bayes model performed slightly better, I decided to use the naive bayes model to predict the unknown data for the kaggle competition. A data frame was created to contain the image ID and the predicted class, and the row names were excluded so only image IDs and the classification was submitted. The final score was 0.83642, which was right within the range of my cross validation/test accuracies!



#Discretizing the validation set  
for (col in colnames(validation)){  
 validation[,col] <- discretize(validation[,col], method = "fixed", breaks = c(-1, 0.1, 100, 256),  
 labels = c("empty", "edge", "mark"))  
}  
#Predictions with best model  
final\_preds <- suppressWarnings(predict(nb, validation, type = "raw"))  
#image ids  
ids <- seq\_along(final\_preds$class)  
#Saving in data frame  
predictions <- data.frame("ImageId" = ids, Label = final\_preds$class)  
#writing to file  
write.csv(predictions, "kaggle\_predictions.csv", row.names = F)