Digit Recognizer

## Introduction

The following report regards using multiple models in preparation for the Kaggle's Digit Recognizer competition, which can be found here >> <https://www.kaggle.com/c/digit-recognizer/data>

Kaggle provides a file of grayscale images of hand-drawn digits, from zero through nine, in both a training and testing set. The competition is submitting sample files to Kaggle to achieve the highest prediction accuracy rate. This report aims to use multiple methods, demonstrating tuning and model evaluation to create a sample to submit to Kaggle.

The following models are to be used:

* Decision tree
* Naïve Bayes
* kNN
* SVM
* Random Forest

## Analysis and Models

The data provided by Kaggle is in both a train and test file. The training file contains labels, while the test file does not. The files contain 785 and 784 columns, respectively. Each has 42,000 records. The columns each represent a pixel and its grayscale representative number for the image the row represents and describes.

## Getting the Data

The data is first downloaded from Kaggle (<https://www.kaggle.com/c/digit-recognizer/data>) and then loaded into R. The 42,000 row and 785 column counts are confirmed.

#First, load the training data in CSV format, then convert "label" to a nominal variable.  
filename <-"digit\_train.csv"  
DigitTotalDF <- read.csv(filename, header = TRUE, stringsAsFactors = TRUE)  
DigitTotalDF$label<-as.factor(DigitTotalDF$label)  
dim(DigitTotalDF)

## [1] 42000 785

Next, we view the head of the file to confirm the contents.

head(DigitTotalDF)

## label pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9  
## 1 1 0 0 0 0 0 0 0 0 0 0  
## 2 0 0 0 0 0 0 0 0 0 0 0  
## 3 1 0 0 0 0 0 0 0 0 0 0  
## 4 4 0 0 0 0 0 0 0 0 0 0  
## 5 0 0 0 0 0 0 0 0 0 0 0  
## 6 0 0 0 0 0 0 0 0 0 0 0  
## pixel10 pixel11 pixel12 pixel13 pixel14 pixel15 pixel16 pixel17 pixel18  
## 1 0 0 0 0 0 0 0 0 0  
## 2 0 0 0 0 0 0 0 0 0  
## 3 0 0 0 0 0 0 0 0 0  
## 4 0 0 0 0 0 0 0 0 0  
## 5 0 0 0 0 0 0 0 0 0  
## 6 0 0 0 0 0 0 0 0 0  
## pixel19 pixel20 pixel21 pixel22 pixel23 pixel24 pixel25 pixel26 pixel27  
## 1 0 0 0 0 0 0 0 0 0  
## 2 0 0 0 0 0 0 0 0 0  
## 3 0 0 0 0 0 0 0 0 0  
## 4 0 0 0 0 0 0 0 0 0  
## 5 0 0 0 0 0 0 0 0 0  
## 6 0 0 0 0 0 0 0 0 0  
## pixel28 pixel29 pixel30 pixel31 pixel32 pixel33 pixel34 pixel35 pixel36  
## 1 0 0 0 0 0 0 0 0 0  
## 2 0 0 0 0 0 0 0 0 0  
## 3 0 0 0 0 0 0 0 0 0  
## 4 0 0 0 0 0 0 0 0 0  
## 5 0 0 0 0 0 0 0 0 0  
## 6 0 0 0 0 0 0 0 0 0  
## pixel37 pixel38 pixel39 pixel40 pixel41 pixel42 pixel43 pixel44 pixel45  
## 1 0 0 0 0 0 0 0 0 0  
## 2 0 0 0 0 0 0 0 0 0  
## 3 0 0 0 0 0 0 0 0 0  
## 4 0 0 0 0 0 0 0 0 0  
## 5 0 0 0 0 0 0 0 0 0  
## 6 0 0 0 0 0 0 0 0 0  
## pixel46 pixel47 pixel48 pixel49 pixel50 pixel51 pixel52 pixel53 pixel54  
## 1 0 0 0 0 0 0 0 0 0  
## 2 0 0 0 0 0 0 0 0 0  
## 3 0 0 0 0 0 0 0 0 0  
## 4 0 0 0 0 0 0 0 0 0  
## 5 0 0 0 0 0 0 0 0 0  
## 6 0 0 0 0 0 0 0 0 0  
## pixel55 pixel56 pixel57 pixel58 pixel59 pixel60 pixel61 pixel62 pixel63  
## 1 0 0 0 0 0 0 0 0 0  
## 2 0 0 0 0 0 0 0 0 0  
## 3 0 0 0 0 0 0 0 0 0  
## 4 0 0 0 0 0 0 0 0 0  
## 5 0 0 0 0 0 0 0 0 0  
## 6 0 0 0 0 0 0 0 0 0  
## pixel64 pixel65 pixel66 pixel67 pixel68 pixel69 pixel70 pixel71 pixel72  
## 1 0 0 0 0 0 0 0 0 0  
## 2 0 0 0 0 0 0 0 0 0  
## 3 0 0 0 0 0 0 0 0 0  
## 4 0 0 0 0 0 0 0 0 0  
## 5 0 0 0 0 0 0 0 0 0  
## 6 0 0 0 0 0 0 0 0 0  
## pixel73 pixel74 pixel75 pixel76 pixel77 pixel78 pixel79 pixel80 pixel81  
## 1 0 0 0 0 0 0 0 0 0  
## 2 0 0 0 0 0 0 0 0 0  
## 3 0 0 0 0 0 0 0 0 0  
## 4 0 0 0 0 0 0 0 0 0  
## 5 0 0 0 0 0 0 0 0 0  
## 6 0 0 0 0 0 0 0 0 0

*Output abbreviated for readability.*

## Data Preparation

Next, a sample data set is created from a random twenty-five percent (25%) selection from the entire data set, static variables are set, a holdout data set is created, a function for model evaluation is created, and a binarized version of the test and train datasets was created.

#Create a random sample of n% of train data set  
percent <- .25  
set.seed(275)  
DigitSplit <- sample(nrow(DigitTotalDF),nrow(DigitTotalDF)\*percent)  
DigitDF <- DigitTotalDF[DigitSplit,]  
dim(DigitDF)

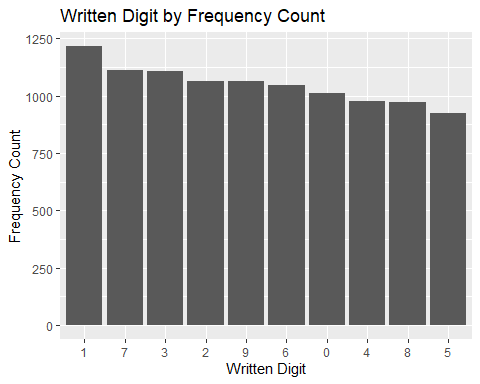
## [1] 10500 785

trainset <- DigitDF  
  
# Setting static variables used throughout the Models section  
N <- nrow(trainset)  
kfolds <- 3  
set.seed(30)  
holdout <- split(sample(1:N), 1:kfolds)  
  
# Function for model evaluation  
get\_accuracy\_rate <- function(results\_table, total\_cases) {  
 diagonal\_sum <- sum(c(results\_table[[1]], results\_table[[12]], results\_table[[23]], results\_table[[34]],  
 results\_table[[45]], results\_table[[56]], results\_table[[67]], results\_table[[78]],  
 results\_table[[89]], results\_table[[100]]))  
 (diagonal\_sum / total\_cases)\*100  
}  
  
# Discretizing at 87%  
binarized\_trainset <- trainset  
for (col in colnames(binarized\_trainset)) {  
 if (col != "label") {  
 binarized\_trainset[, c(col)] <- ifelse(binarized\_trainset[, c(col)] > 131, 1, 0)  
 }  
}  
for (col in colnames(binarized\_trainset)) {  
 if (col != "label") {  
 binarized\_trainset[, c(col)] <- as.factor(binarized\_trainset[, c(col)])  
 }  
}

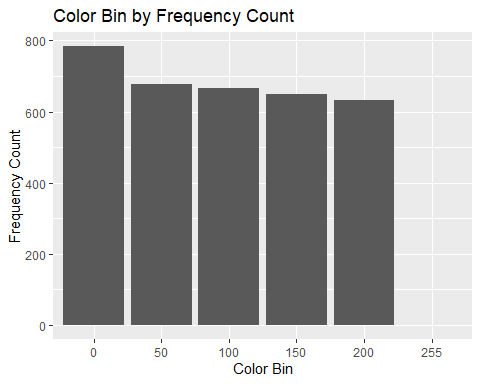
## Exploratory Data Analysis & Visualization

Next, visualizations of the trainset of data created in data preparation is done to evaluate and familiarize with the data set used in models going forward.

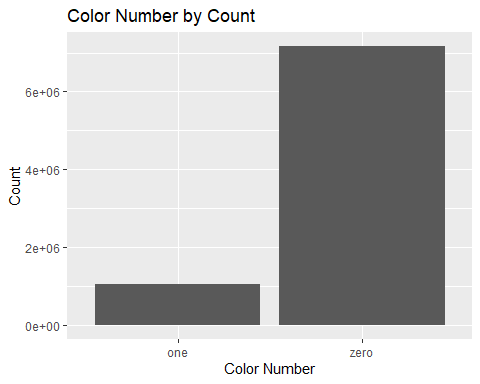
digit\_freq <- sqldf("SELECT label, COUNT(label) as count  
 FROM trainset  
 GROUP BY label")  
ggplot(digit\_freq, aes(x=reorder(label, -count), y=count)) + geom\_bar(stat="identity") + xlab("Written Digit") + ylab("Frequency Count") + ggtitle("Written Digit by Frequency Count")



zero <- 0  
fifty <- 0  
one\_hundred <- 0  
one\_hundred\_fifty <- 0  
two\_hundred <- 0  
two\_hundred\_fifty\_five <- 0  
for (col in colnames(trainset)) {  
 if (col != "label") {  
 #binarized\_trainset[,c(col)] <- ifelse(binarized\_trainset[,c(col)] > 131, 1, 0)  
 ifelse(trainset[,c(col)] == 0, zero <- zero + 1, ifelse(  
 trainset[,c(col)] < 51, fifty <- fifty + 1, ifelse(  
 trainset[,c(col)] < 101, one\_hundred <- one\_hundred + 1, ifelse(  
 trainset[,c(col)] < 151, one\_hundred\_fifty <- one\_hundred\_fifty + 1, ifelse(  
 trainset[,c(col)] < 201, two\_hundred <- two\_hundred + 1, two\_hundred\_fifty\_five + 1  
 )  
 )  
 )  
 )  
 )  
 }  
}  
  
color\_bins <- data.frame("color\_bin"=c("0", "50", "100", "150", "200", "255"),  
 "count"=c(zero, fifty, one\_hundred, one\_hundred\_fifty, two\_hundred, two\_hundred\_fifty\_five))  
ggplot(color\_bins, aes(x=reorder(color\_bin, -count), y=count)) + geom\_bar(stat="identity") + xlab("Color Bin") + ylab("Frequency Count") + ggtitle("Color Bin by Frequency Count")



color\_freq <- data.frame("0"=c(), "1"=c())  
for (col in colnames(binarized\_trainset)) {  
 if (col != "label") {  
 zero <- c(length(which(binarized\_trainset[,c(col)] == 0)))  
 one <- c(length(which(binarized\_trainset[,c(col)] == 1)))  
 color\_freq <- rbind(color\_freq, data.frame("0"=zero, "1"=one))  
 }  
}  
colnames(color\_freq) <- c("zero", "one")  
color\_freq <- data.frame("number"=c("zero", "one"), "count"=c(sum(color\_freq$zero), sum(color\_freq$one)))  
  
ggplot(color\_freq, aes(x=number, y=count)) + geom\_bar(stat="identity") + xlab("Color Number") + ylab("Count") + ggtitle("Color Number by Count")



**Models**

## Decision Trees

First, for decision tree modeling, the trainset is split into a train set of sixty percent (60%) and a test set of forty percent (40%).

A decision tree model is created with a cp of .02. After numerous updates to the parameter .02 was chosen as the most appropriate and best result with this data set.

#DecisionTree Model  
train\_tree1 <- rpart(label ~ ., data = train, method="class", control=rpart.control(cp=.02))  
#summary(train\_tree1)

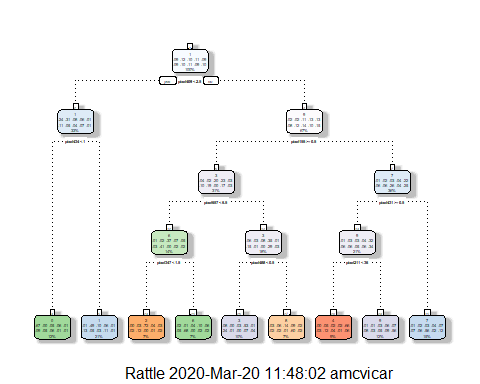
## Decision Tree Model Prediction

Next, the test segment is used with the decision tree model to create a prediction file that can then be tested for accuracy.

##   
## Classification tree:  
## rpart(formula = label ~ ., data = train, method = "class", control = rpart.control(cp = 0.02))  
##   
## Variables used in tree construction:  
## [1] pixel155 pixel211 pixel347 pixel409 pixel431 pixel434 pixel488 pixel657  
##   
## Root node error: 5561/6300 = 0.8827  
##   
## n= 6300   
##   
## CP nsplit rel error xerror xstd  
## 1 0.094228 0 1.00000 1.00000 0.0045928  
## 2 0.089193 1 0.90577 0.90919 0.0056819  
## 3 0.074987 2 0.81658 0.80669 0.0064629  
## 4 0.070311 3 0.74159 0.75364 0.0067356  
## 5 0.060780 4 0.67128 0.69358 0.0069545  
## 6 0.052509 5 0.61050 0.61751 0.0071075  
## 7 0.044596 6 0.55799 0.56483 0.0071365  
## 8 0.042079 7 0.51340 0.53084 0.0071224  
## 9 0.020000 8 0.47132 0.49002 0.0070713

## Warning in rsq.rpart(train\_tree1): may not be applicable for this method

|  |  |
| --- | --- |
|  |  |
|  |  |



The tree graphic is difficult to read. A confusion matrix is generated to show accuracy and other important statistics.

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 344 1 35 36 5 47 34 26 6 9  
## 1 4 423 100 54 16 117 57 20 118 16  
## 2 0 2 183 12 12 1 20 3 7 5  
## 3 32 1 13 212 7 96 4 2 26 9  
## 4 5 18 9 4 241 10 47 14 8 15  
## 5 0 0 0 0 0 0 0 0 0 0  
## 6 6 1 24 29 22 19 190 0 5 2  
## 7 8 18 20 25 50 33 38 348 13 91  
## 8 5 11 33 21 13 7 4 1 162 5  
## 9 11 3 11 42 36 32 29 24 51 271  
##   
## Overall Statistics  
##   
## Accuracy : 0.5652   
## 95% CI : (0.5501, 0.5803)  
## No Information Rate : 0.1138   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.5151   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.82892 0.8849 0.42757 0.48736 0.59950 0.00000  
## Specificity 0.94742 0.8651 0.98356 0.94954 0.96577 1.00000  
## Pos Pred Value 0.63352 0.4573 0.74694 0.52736 0.64960 NaN  
## Neg Pred Value 0.98059 0.9832 0.93805 0.94128 0.95795 0.91381  
## Prevalence 0.09881 0.1138 0.10190 0.10357 0.09571 0.08619  
## Detection Rate 0.08190 0.1007 0.04357 0.05048 0.05738 0.00000  
## Detection Prevalence 0.12929 0.2202 0.05833 0.09571 0.08833 0.00000  
## Balanced Accuracy 0.88817 0.8750 0.70557 0.71845 0.78264 0.50000  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.44917 0.79452 0.40909 0.64066  
## Specificity 0.97141 0.92132 0.97371 0.93672  
## Pos Pred Value 0.63758 0.54037 0.61832 0.53137  
## Neg Pred Value 0.94029 0.97469 0.94058 0.95881  
## Prevalence 0.10071 0.10429 0.09429 0.10071  
## Detection Rate 0.04524 0.08286 0.03857 0.06452  
## Detection Prevalence 0.07095 0.15333 0.06238 0.12143  
## Balanced Accuracy 0.71029 0.85792 0.69140 0.78869

## Naive Bayes

Next, the train set is used with Naive Bayes.

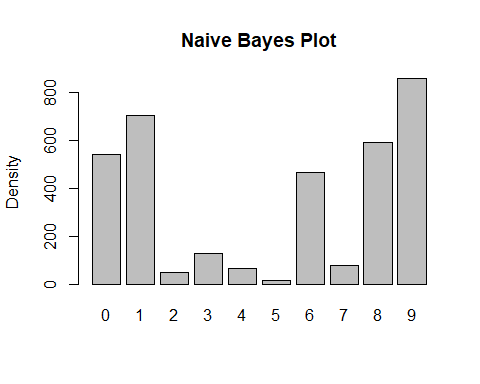
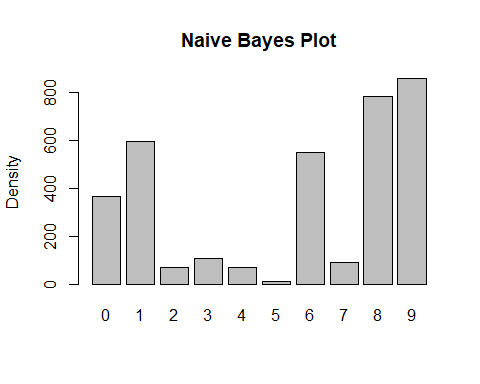
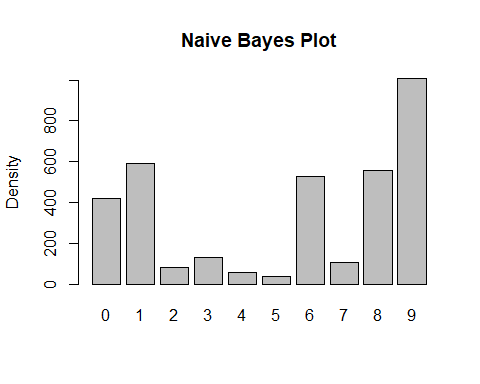
A confusion matrix is generated to show accuracy and other important statistics.

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 377 0 42 41 7 53 10 3 9 5  
## 1 2 469 28 33 19 27 18 26 114 35  
## 2 3 0 63 2 0 1 1 0 0 0  
## 3 0 0 63 142 2 8 0 3 4 0  
## 4 3 0 6 1 84 7 2 8 4 7  
## 5 1 0 2 1 0 18 1 0 8 0  
## 6 9 5 110 22 37 25 384 4 10 1  
## 7 0 0 3 0 0 0 0 114 1 1  
## 8 12 3 109 132 44 187 5 25 201 11  
## 9 8 1 2 61 209 36 2 255 45 363  
##   
## Overall Statistics  
##   
## Accuracy : 0.5274   
## 95% CI : (0.5121, 0.5426)  
## No Information Rate : 0.1138   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.4738   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.90843 0.9812 0.14720 0.32644 0.20896 0.049724  
## Specificity 0.95509 0.9189 0.99814 0.97875 0.98999 0.996613  
## Pos Pred Value 0.68921 0.6083 0.90000 0.63964 0.68852 0.580645  
## Neg Pred Value 0.98960 0.9974 0.91162 0.92634 0.92202 0.917486  
## Prevalence 0.09881 0.1138 0.10190 0.10357 0.09571 0.086190  
## Detection Rate 0.08976 0.1117 0.01500 0.03381 0.02000 0.004286  
## Detection Prevalence 0.13024 0.1836 0.01667 0.05286 0.02905 0.007381  
## Balanced Accuracy 0.93176 0.9500 0.57267 0.65259 0.59947 0.523168  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.90780 0.26027 0.50758 0.85816  
## Specificity 0.94096 0.99867 0.86120 0.83611  
## Pos Pred Value 0.63262 0.95798 0.27572 0.36965  
## Neg Pred Value 0.98915 0.92061 0.94382 0.98135  
## Prevalence 0.10071 0.10429 0.09429 0.10071  
## Detection Rate 0.09143 0.02714 0.04786 0.08643  
## Detection Prevalence 0.14452 0.02833 0.17357 0.23381  
## Balanced Accuracy 0.92438 0.62947 0.68439 0.84713

##Naive Bayes 3 K-fold

The Naive Bayes model is adjusted to use 3 K-fold validation.

#Run training and Testing for each of the k-folds  
  
all\_results <- data.frame(orig = c(), pred =c ())  
  
for (k in 1:kfolds){  
DigitDF\_Test <- DigitDF[holdout[[k]], ]  
DigitDF\_Train=DigitDF[-holdout[[k]], ]  
## View the created Test and Train sets  
(head(DigitDF\_Train))  
(table(DigitDF\_Test$Label))  
## Make sure you take the labels out of the testing data  
(head(DigitDF\_Test))  
DigitDF\_Test\_noLabel<-DigitDF\_Test[-c(1)]  
DigitDF\_Test\_justLabel<-DigitDF\_Test$label  
(head(DigitDF\_Test\_noLabel))  
   
#### Naive Bayes prediction using e1071 package  
#Naive Bayes Train model  
train\_naibayes<-naiveBayes(label~., data=DigitDF\_Train, na.action = na.pass)  
#train\_naibayes  
#summary(train\_naibayes)  
  
#Naive Bayes model Prediction   
nb\_Pred <- predict(train\_naibayes, DigitDF\_Test\_noLabel)  
#nb\_Pred  
  
  
#Testing accuracy of naive bayes model with Kaggle train data subset  
(confusionMatrix(nb\_Pred, DigitDF\_Test$label))  
  
# Accumulate results from each fold, if you like  
all\_results <- rbind(all\_results, data.frame(orig=DigitDF\_Test\_justLabel, pred=nb\_Pred))  
   
##Visualize  
plot(nb\_Pred, ylab = "Density", main = "Naive Bayes Plot")  
  
}



A confusion matrix is generated to show accuracy and other important statistics.

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 896 0 105 124 18 131 17 11 15 8  
## 1 3 1195 67 113 28 76 43 61 234 73  
## 2 2 0 183 2 2 3 3 0 2 0  
## 3 1 0 77 266 1 9 1 2 5 1  
## 4 0 0 11 3 132 5 2 11 6 18  
## 5 1 0 9 6 3 24 6 0 10 2  
## 6 44 5 316 76 79 56 933 11 20 1  
## 7 0 0 4 8 3 0 0 253 1 3  
## 8 48 8 266 376 129 508 18 49 508 25  
## 9 17 9 27 135 581 112 23 714 173 934  
##   
## Overall Statistics  
##   
## Accuracy : 0.507   
## 95% CI : (0.4974, 0.5167)  
## No Information Rate : 0.1159   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.4513   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.88538 0.9819 0.17183 0.23986 0.13525 0.025974  
## Specificity 0.95478 0.9248 0.99852 0.98967 0.99412 0.996136  
## Pos Pred Value 0.67623 0.6313 0.92893 0.73278 0.70213 0.393443  
## Neg Pred Value 0.98736 0.9974 0.91439 0.91684 0.91815 0.913785  
## Prevalence 0.09638 0.1159 0.10143 0.10562 0.09295 0.088000  
## Detection Rate 0.08533 0.1138 0.01743 0.02533 0.01257 0.002286  
## Detection Prevalence 0.12619 0.1803 0.01876 0.03457 0.01790 0.005810  
## Balanced Accuracy 0.92008 0.9534 0.58517 0.61476 0.56468 0.511055  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.89197 0.2275 0.52156 0.87700  
## Specificity 0.93569 0.9980 0.85020 0.81017  
## Pos Pred Value 0.60545 0.9301 0.26253 0.34275  
## Neg Pred Value 0.98739 0.9160 0.94559 0.98315  
## Prevalence 0.09962 0.1059 0.09276 0.10143  
## Detection Rate 0.08886 0.0241 0.04838 0.08895  
## Detection Prevalence 0.14676 0.0259 0.18429 0.25952  
## Balanced Accuracy 0.91383 0.6127 0.68588 0.84359

##kNN

A kNN model is created using 7 as the k guess parameter.

k\_guess = 7# round(sqrt(nrow(trainset)))  
all\_results <- data.frame(orig = c(), pred =c ())  
for (k in 1:kfolds) {  
 new\_test <- trainset[holdout[[k]], ]  
 new\_train <- trainset[-holdout[[k]], ]  
   
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
   
 pred <- knn(train=new\_train, test=new\_test, cl=new\_train$label, k=k\_guess, prob=FALSE)  
   
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
}  
  
table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 996 1 2 1 0 4 6 0 1 1  
## 1 0 1209 1 0 2 0 0 3 0 2  
## 2 16 35 951 11 2 1 5 36 6 2  
## 3 4 12 5 1051 0 10 1 6 10 10  
## 4 1 20 0 0 889 0 6 5 0 55  
## 5 6 9 0 22 2 849 19 1 1 15  
## 6 8 6 0 0 2 6 1023 0 1 0  
## 7 0 29 0 0 6 0 1 1053 0 23  
## 8 4 25 2 32 7 28 9 9 840 18  
## 9 3 9 1 11 14 4 0 33 1 989

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 93.80952

A confusion matrix is generated to show accuracy and other important statistics.

#Testing accuracy of naive bayes model with Kaggle train data subset  
confusionMatrix(all\_results$pred, all\_results$orig)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 996 0 16 4 1 6 8 0 4 3  
## 1 1 1209 35 12 20 9 6 29 25 9  
## 2 2 1 951 5 0 0 0 0 2 1  
## 3 1 0 11 1051 0 22 0 0 32 11  
## 4 0 2 2 0 889 2 2 6 7 14  
## 5 4 0 1 10 0 849 6 0 28 4  
## 6 6 0 5 1 6 19 1023 1 9 0  
## 7 0 3 36 6 5 1 0 1053 9 33  
## 8 1 0 6 10 0 1 1 0 840 1  
## 9 1 2 2 10 55 15 0 23 18 989  
##   
## Overall Statistics  
##   
## Accuracy : 0.9381   
## 95% CI : (0.9333, 0.9426)  
## No Information Rate : 0.1159   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9311   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.98419 0.9934 0.89296 0.9477 0.91086 0.91883  
## Specificity 0.99557 0.9843 0.99883 0.9918 0.99633 0.99447  
## Pos Pred Value 0.95954 0.8923 0.98857 0.9317 0.96212 0.94124  
## Neg Pred Value 0.99831 0.9991 0.98805 0.9938 0.99091 0.99219  
## Prevalence 0.09638 0.1159 0.10143 0.1056 0.09295 0.08800  
## Detection Rate 0.09486 0.1151 0.09057 0.1001 0.08467 0.08086  
## Detection Prevalence 0.09886 0.1290 0.09162 0.1074 0.08800 0.08590  
## Balanced Accuracy 0.98988 0.9888 0.94590 0.9698 0.95359 0.95665  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.97801 0.9469 0.86242 0.92864  
## Specificity 0.99503 0.9901 0.99790 0.98665  
## Pos Pred Value 0.95607 0.9188 0.97674 0.88700  
## Neg Pred Value 0.99756 0.9937 0.98610 0.99190  
## Prevalence 0.09962 0.1059 0.09276 0.10143  
## Detection Rate 0.09743 0.1003 0.08000 0.09419  
## Detection Prevalence 0.10190 0.1091 0.08190 0.10619  
## Balanced Accuracy 0.98652 0.9685 0.93016 0.95764

##SVM

A SVM model is created.

cols\_to\_remove = c()   
for (col in colnames(trainset)) {   
 if (col != "label") {   
 if (length(unique(trainset[, c(col)])) == 1) { cols\_to\_remove <- c(cols\_to\_remove, col)   
 }   
 }   
 }   
  
svm\_trainset <- trainset[-which(colnames(trainset) %in% cols\_to\_remove)]  
  
  
# Baseline SVM - no changes to data  
all\_results <- data.frame(orig=c(), pred=c())  
for (k in 1:kfolds) {  
 new\_test <- svm\_trainset[holdout[[k]], ]  
 new\_train <- svm\_trainset[-holdout[[k]], ]  
   
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
   
 test\_model <- svm(label ~ ., new\_train, na.action=na.pass)  
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
   
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
}  
  
#table(all\_results$orig, all\_results$pred)  
#get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

A confusion matrix is generated to show accuracy and other important statistics.

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 0 0 0 0 0 0 0 0 0 0  
## 1 1012 1217 1065 1109 976 924 1046 1112 974 1065  
## 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 0 0 0 0 0 0 0 0 0 0  
##   
## Overall Statistics  
##   
## Accuracy : 0.1159   
## 95% CI : (0.1098, 0.1222)  
## No Information Rate : 0.1159   
## P-Value [Acc > NIR] : 0.5045   
##   
## Kappa : 0   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.00000 1.0000 0.0000 0.0000 0.00000 0.000  
## Specificity 1.00000 0.0000 1.0000 1.0000 1.00000 1.000  
## Pos Pred Value NaN 0.1159 NaN NaN NaN NaN  
## Neg Pred Value 0.90362 NaN 0.8986 0.8944 0.90705 0.912  
## Prevalence 0.09638 0.1159 0.1014 0.1056 0.09295 0.088  
## Detection Rate 0.00000 0.1159 0.0000 0.0000 0.00000 0.000  
## Detection Prevalence 0.00000 1.0000 0.0000 0.0000 0.00000 0.000  
## Balanced Accuracy 0.50000 0.5000 0.5000 0.5000 0.50000 0.500  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.00000 0.0000 0.00000 0.0000  
## Specificity 1.00000 1.0000 1.00000 1.0000  
## Pos Pred Value NaN NaN NaN NaN  
## Neg Pred Value 0.90038 0.8941 0.90724 0.8986  
## Prevalence 0.09962 0.1059 0.09276 0.1014  
## Detection Rate 0.00000 0.0000 0.00000 0.0000  
## Detection Prevalence 0.00000 0.0000 0.00000 0.0000  
## Balanced Accuracy 0.50000 0.5000 0.50000 0.5000

##SVM Binarized Data

The same SVM model is run, but using a binarized data set.

cols\_to\_remove = c()   
for (col in colnames(binarized\_trainset)) {   
if (col != "label") {   
if (length(unique(binarized\_trainset[, c(col)])) == 1) { cols\_to\_remove <- c(cols\_to\_remove, col)   
}   
}   
}  
  
binarized\_trainset <- binarized\_trainset[-which(colnames(binarized\_trainset) %in% cols\_to\_remove)]  
  
all\_results <- data.frame(orig=c(), pred=c())   
  
for (k in 1:kfolds) {   
 new\_test <- binarized\_trainset[holdout[[k]], ]   
 new\_train <- binarized\_trainset[-holdout[[k]], ]  
   
new\_test\_no\_label <- new\_test[-c(1)]   
new\_test\_just\_label <- new\_test[c(1)]  
  
test\_model <- svm(label ~ ., new\_train, na.action=na.pass)   
  
pred <- predict(test\_model, new\_test\_no\_label, type = c("class"))  
  
all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))   
  
}   
  
#table(all\_results$orig, all\_results$pred)  
#get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), #length(all\_results$pred))

A confusion matrix is generated to show accuracy and other important statistics.

#Testing accuracy of naive bayes model with Kaggle train data sub set  
confusionMatrix(all\_results$pred, all\_results$orig)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 976 0 13 3 4 8 10 3 3 5  
## 1 0 1194 13 9 4 16 5 20 25 9  
## 2 1 3 922 18 4 3 5 11 7 4  
## 3 5 0 13 977 0 36 1 0 23 17  
## 4 3 2 21 0 906 9 9 13 7 37  
## 5 11 5 6 48 1 805 12 5 34 9  
## 6 11 1 21 8 5 18 997 1 9 1  
## 7 2 2 23 12 1 4 0 1016 4 41  
## 8 3 6 28 25 5 15 7 4 842 7  
## 9 0 4 5 9 46 10 0 39 20 935  
##   
## Overall Statistics  
##   
## Accuracy : 0.9114   
## 95% CI : (0.9058, 0.9168)  
## No Information Rate : 0.1159   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9015   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.96443 0.9811 0.86573 0.88097 0.92828 0.87121  
## Specificity 0.99484 0.9891 0.99406 0.98988 0.98940 0.98632  
## Pos Pred Value 0.95220 0.9220 0.94274 0.91138 0.89970 0.86004  
## Neg Pred Value 0.99620 0.9975 0.98498 0.98600 0.99263 0.98756  
## Prevalence 0.09638 0.1159 0.10143 0.10562 0.09295 0.08800  
## Detection Rate 0.09295 0.1137 0.08781 0.09305 0.08629 0.07667  
## Detection Prevalence 0.09762 0.1233 0.09314 0.10210 0.09590 0.08914  
## Balanced Accuracy 0.97963 0.9851 0.92990 0.93543 0.95884 0.92877  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.95315 0.91367 0.86448 0.87793  
## Specificity 0.99207 0.99052 0.98950 0.98590  
## Pos Pred Value 0.93004 0.91946 0.89384 0.87547  
## Neg Pred Value 0.99480 0.98978 0.98619 0.98622  
## Prevalence 0.09962 0.10590 0.09276 0.10143  
## Detection Rate 0.09495 0.09676 0.08019 0.08905  
## Detection Prevalence 0.10210 0.10524 0.08971 0.10171  
## Balanced Accuracy 0.97261 0.95209 0.92699 0.93192

##Random Forest

Lastly, several variations of the Random Forest model are created, including a test for the number of trees. The first model is run as is, without any parameters.

all\_results <- data.frame(orig=c(), pred=c())  
for (k in 1:kfolds) {  
 new\_test <- trainset[holdout[[k]], ]  
 new\_train <- trainset[-holdout[[k]], ]  
   
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
   
 test\_model <- randomForest(label ~ ., new\_train, na.action=na.pass)  
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
   
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
}  
  
#table(all\_results$orig, all\_results$pred)  
#get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

A confusion matrix is generated to show accuracy and other important statistics.

#Testing accuracy of naive bayes model with Kaggle train data subset  
confusionMatrix(all\_results$pred, all\_results$orig)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 990 0 9 3 4 10 9 3 1 4  
## 1 0 1196 6 4 1 6 2 11 7 3  
## 2 0 3 996 17 2 2 1 15 3 5  
## 3 1 3 12 1025 2 21 0 1 21 18  
## 4 1 3 12 2 929 2 5 6 8 20  
## 5 3 1 1 17 0 858 9 0 10 4  
## 6 6 1 9 6 4 11 1015 0 8 0  
## 7 0 5 16 11 0 1 0 1054 2 18  
## 8 10 3 3 14 3 6 5 4 895 7  
## 9 1 2 1 10 31 7 0 18 19 986  
##   
## Overall Statistics  
##   
## Accuracy : 0.947   
## 95% CI : (0.9426, 0.9513)  
## No Information Rate : 0.1159   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9411   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.97826 0.9827 0.93521 0.92426 0.95184 0.92857  
## Specificity 0.99547 0.9957 0.99491 0.99159 0.99381 0.99530  
## Pos Pred Value 0.95837 0.9676 0.95402 0.92844 0.94028 0.95017  
## Neg Pred Value 0.99768 0.9977 0.99270 0.99106 0.99506 0.99312  
## Prevalence 0.09638 0.1159 0.10143 0.10562 0.09295 0.08800  
## Detection Rate 0.09429 0.1139 0.09486 0.09762 0.08848 0.08171  
## Detection Prevalence 0.09838 0.1177 0.09943 0.10514 0.09410 0.08600  
## Balanced Accuracy 0.98686 0.9892 0.96506 0.95792 0.97282 0.96194  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.97036 0.9478 0.91889 0.9258  
## Specificity 0.99524 0.9944 0.99423 0.9906  
## Pos Pred Value 0.95755 0.9521 0.94211 0.9172  
## Neg Pred Value 0.99672 0.9938 0.99173 0.9916  
## Prevalence 0.09962 0.1059 0.09276 0.1014  
## Detection Rate 0.09667 0.1004 0.08524 0.0939  
## Detection Prevalence 0.10095 0.1054 0.09048 0.1024  
## Balanced Accuracy 0.98280 0.9711 0.95656 0.9582

##Random Forest Best

The following tests the sensitivity to the number of trees allowed parameter. The default is 500 and the following code tests for 5 to 15.

prev\_result <- 0  
best\_result <- 0  
best\_number\_trees <-0  
for (trees in 5:15) {  
 if (trees %% 5 == 0) {  
 all\_results <- data.frame(orig=c(), pred=c())  
 for (k in 1:kfolds) {  
 new\_test <- trainset[holdout[[k]], ]  
 new\_train <- trainset[-holdout[[k]], ]  
   
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
   
 test\_model <- randomForest(label ~ ., new\_train, replace=TRUE, na.action=na.pass)  
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
   
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
 }  
 #table(all\_results$orig, all\_results$pred)  
 new\_result <- get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))  
   
 if (new\_result > prev\_result) {  
 prev\_result <- new\_result  
 } else {  
 best\_number\_trees <- trees  
 best\_result <- new\_result  
 break  
 }  
 }  
}   
paste("Best Number of Trees:", best\_number\_trees, "- Best Result:", best\_result, sep=" ")

## [1] "Best Number of Trees: 10 - Best Result: 94.7428571428571"

table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 988 0 0 1 1 3 8 0 10 1  
## 1 0 1196 3 2 4 1 1 6 3 1  
## 2 8 6 996 9 12 2 10 15 6 1  
## 3 1 3 20 1028 3 15 6 9 15 9  
## 4 4 2 2 2 932 0 4 0 2 28  
## 5 10 6 1 20 1 865 10 1 4 6  
## 6 11 2 2 0 5 8 1012 0 6 0  
## 7 2 10 14 1 7 0 1 1057 3 17  
## 8 1 8 4 20 6 11 12 1 887 24  
## 9 3 3 4 14 18 6 0 21 9 987

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 94.74286

A confusion matrix is generated to show accuracy and other important statistics.

#Testing accuracy of naive bayes model with Kaggle train data sub set  
confusionMatrix(all\_results$pred, all\_results$orig)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 988 0 8 1 4 10 11 2 1 3  
## 1 0 1196 6 3 2 6 2 10 8 3  
## 2 0 3 996 20 2 1 2 14 4 4  
## 3 1 2 9 1028 2 20 0 1 20 14  
## 4 1 4 12 3 932 1 5 7 6 18  
## 5 3 1 2 15 0 865 8 0 11 6  
## 6 8 1 10 6 4 10 1012 1 12 0  
## 7 0 6 15 9 0 1 0 1057 1 21  
## 8 10 3 6 15 2 4 6 3 887 9  
## 9 1 1 1 9 28 6 0 17 24 987  
##   
## Overall Statistics  
##   
## Accuracy : 0.9474   
## 95% CI : (0.943, 0.9516)  
## No Information Rate : 0.1159   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9415   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.97628 0.9827 0.93521 0.9270 0.95492 0.93615  
## Specificity 0.99578 0.9957 0.99470 0.9927 0.99402 0.99520  
## Pos Pred Value 0.96109 0.9676 0.95220 0.9371 0.94237 0.94951  
## Neg Pred Value 0.99747 0.9977 0.99270 0.9914 0.99537 0.99385  
## Prevalence 0.09638 0.1159 0.10143 0.1056 0.09295 0.08800  
## Detection Rate 0.09410 0.1139 0.09486 0.0979 0.08876 0.08238  
## Detection Prevalence 0.09790 0.1177 0.09962 0.1045 0.09419 0.08676  
## Balanced Accuracy 0.98603 0.9892 0.96496 0.9598 0.97447 0.96567  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.96750 0.9505 0.91068 0.9268  
## Specificity 0.99450 0.9944 0.99391 0.9908  
## Pos Pred Value 0.95113 0.9523 0.93862 0.9190  
## Neg Pred Value 0.99640 0.9941 0.99089 0.9917  
## Prevalence 0.09962 0.1059 0.09276 0.1014  
## Detection Rate 0.09638 0.1007 0.08448 0.0940  
## Detection Prevalence 0.10133 0.1057 0.09000 0.1023  
## Balanced Accuracy 0.98100 0.9724 0.95229 0.9588

##Random Forest Tuning

The Random Forest Original Model is updated with parameter in an attempt to achieve a higher prediction or accuracy rate.

Additionally, the files previously spit into 60% training and 40% test are used instead of k-fold or cross validation for this tuning exercise.

The additional parameters are specifiying the type is classification and setting the importance and proximity parameters to true.

Setting the type to classification is correct but lowered the accuracy slightly to %94.12 and will not be used

Adding the parameter of importance gave a huge jump in accuracy to %97.92857.

The addition of the proximity parameter set to true brought the accuracy rate to %98.02381.

all\_results <- data.frame(orig=c(), pred=c())  
  
 new\_test\_no\_label <- test[-c(1)]  
 new\_test\_just\_label <- test[c(1)]  
   
 test\_model <- randomForest(label ~ ., new\_train, na.action=na.pass, importance = TRUE, proximity = TRUE)  
   
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
   
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
  
table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 412 0 0 0 0 2 0 0 0 1  
## 1 0 474 2 1 0 0 0 1 0 0  
## 2 0 2 421 0 0 0 4 1 0 0  
## 3 0 0 1 421 0 3 0 4 2 4  
## 4 1 1 0 0 394 0 1 0 2 3  
## 5 0 0 0 3 0 358 0 0 0 1  
## 6 1 0 0 0 0 1 419 0 2 0  
## 7 0 1 1 0 1 0 0 433 0 2  
## 8 0 2 0 7 0 1 1 1 382 2  
## 9 0 1 0 3 0 0 0 2 2 415

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 98.30952

## Result

Each model achieved the following accuracy rates.

*Decision Tree - 0.5652*  
Naive Bayes 0 K-folds - 0.5274  
*Naive Bayes 3 K-folds - 0.507* kNN - 0.9388  
*SVM - 0.1159*  
SVM with Binarized Data - 0.9114  
*Random Forest - 0.9486* Random Forest for Best Tree - 0.9479 *Random Forest w/type classification - 0.9412* Random Forest w/importance set to true - 0.9793 \*Random Forest w/importance and proximity set to true - .9802

Random Forest testing for the best tree between 5, 10, & 15 is 15. The accuracy is only slightly less than when unspecified, meaning the default of 500 trees was used.

The winner is Random Forest, and the model will be used to create a file of 28000 sample records to submit to Kaggle.

all\_results <- data.frame(orig=c(), pred=c())  
  
 new\_test <- DigitTotalDF[1:28000,]  
# new\_train <- DigitTotalDF[28001:42000,]  
   
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
   
 # test\_model <- randomForest(label ~ ., new\_train, na.action=na.pass, importance = TRUE, proximity = TRUE)   
   
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
   
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
  
  
table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 2674 0 3 1 4 5 15 1 17 1  
## 1 0 3052 17 9 3 5 8 3 6 4  
## 2 11 15 2679 16 21 2 24 33 17 6  
## 3 6 9 43 2716 6 52 3 27 28 19  
## 4 3 7 6 0 2631 0 18 11 10 70  
## 5 17 10 4 51 3 2370 27 4 14 21  
## 6 16 4 2 1 6 20 2718 0 12 0  
## 7 7 20 34 4 17 0 0 2787 10 45  
## 8 7 18 13 51 15 21 14 5 2513 37  
## 9 14 8 7 41 52 4 2 20 16 2601

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 95.50357

A confusion matrix is generated to show accuracy and other important statistics.

#Testing accuracy of naive bayes model with Kaggle train data sub set  
confusionMatrix(all\_results$pred, all\_results$orig)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 2674 0 11 6 3 17 16 7 7 14  
## 1 0 3052 15 9 7 10 4 20 18 8  
## 2 3 17 2679 43 6 4 2 34 13 7  
## 3 1 9 16 2716 0 51 1 4 51 41  
## 4 4 3 21 6 2631 3 6 17 15 52  
## 5 5 5 2 52 0 2370 20 0 21 4  
## 6 15 8 24 3 18 27 2718 0 14 2  
## 7 1 3 33 27 11 4 0 2787 5 20  
## 8 17 6 17 28 10 14 12 10 2513 16  
## 9 1 4 6 19 70 21 0 45 37 2601  
##   
## Overall Statistics  
##   
## Accuracy : 0.955   
## 95% CI : (0.9525, 0.9574)  
## No Information Rate : 0.111   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.95   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.98273 0.9823 0.94865 0.9337 0.95464 0.94010  
## Specificity 0.99680 0.9963 0.99488 0.9931 0.99497 0.99572  
## Pos Pred Value 0.97060 0.9710 0.95406 0.9398 0.95395 0.95603  
## Neg Pred Value 0.99814 0.9978 0.99424 0.9923 0.99505 0.99408  
## Prevalence 0.09718 0.1110 0.10086 0.1039 0.09843 0.09004  
## Detection Rate 0.09550 0.1090 0.09568 0.0970 0.09396 0.08464  
## Detection Prevalence 0.09839 0.1123 0.10029 0.1032 0.09850 0.08854  
## Balanced Accuracy 0.98976 0.9893 0.97177 0.9634 0.97481 0.96791  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.97805 0.95315 0.93281 0.94069  
## Specificity 0.99560 0.99585 0.99486 0.99196  
## Pos Pred Value 0.96076 0.96403 0.95081 0.92760  
## Neg Pred Value 0.99758 0.99454 0.99286 0.99349  
## Prevalence 0.09925 0.10443 0.09621 0.09875  
## Detection Rate 0.09707 0.09954 0.08975 0.09289  
## Detection Prevalence 0.10104 0.10325 0.09439 0.10014  
## Balanced Accuracy 0.98682 0.97450 0.96384 0.96632

A CSV file is created for submission to Kaggle.

## Conclusion

With such a large data set, it would have been more efficient to create a smaller sample size in the beginning so that the models could be tuned and run faster.

Two models showed promise, and they were the kNN and Random Forest. These should be pursued for additional tuning.

A sample file was created using the tuned Random Forest and submitted to Kaggle. The final accuracy rate was 95% regardless if the model from the tuning segment was used or it was rerun with a larger sample of 14k, the balance of the entire file minus the 28k records to be predicted and submitted to Kaggle. The Kaggle score was 0.10042, placing 2182 under the user name amaguin.