CPTR330 – Final Project – Format

Ivan Guillen & Nicholas Zimmerman

June 6, 2021

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

Final Project – Kaggle Challenge

This project attempts the Kaggle challenge Digit Recognizer (https://www.kaggle.com/c/digit-recognizer). Digit Recognizer is a computer vision problem. Digits 0-9 are provided as handwritten gray-scale 28x28 pixel images and need to be correctly identified by the machine learning algorithm. We will apply KNN, decision tree, and SVM algorithms which are well-established approaches for solving this computer vision task. An ensemble will then combine at least two of the algorithms to produce the best possible algorithm. It is expected that data preparation be difficult because features are not explicitly identified in the dataset. Additional research will need to be done to find methods to extract features to be used by the algorithms.

Data Preparation

Step 1 - Collect Data

For this analysis, we will utilize a dataset of handwritten images sourced from MNIST ("Modified National Institute of Standards and Technology") and provided through the Kaggle.com Digit Recognizer challenge. The training set contains 42000 examples and the test set contains 28000 examples. Features are not provided and must be extracted from the 28x28 pixel gray-scale data for each image.

Step 2 - Exploring And Preparing The Data

First, we will read in both the training and testing sets provided by Kaggle.

```
train <- read.csv("train.csv")
test <- read.csv("test.csv")
```

Although using the `str` function on the data would result in a very long output, we can see the first few rows and columns to get a general idea.

```
train[1:5, 1:2]
```

```
##   label pixel0
## 1     1      0
## 2     0      0
## 3     1      0
## 4     4      0
## 5     0      0
```

We know that we have 42000 examples in the training data. Additionally, we know (from the Kaggle data overview) that the training set has 785 variables. The first variable, `label`, is the target value. Based on the remaining 784 variables per row, each representing a single pixel, the goal is to predict this target value. The testing set does not contain this first target value, as it is our goal to create a model that predicts this value.

Next, we can see what types of variables we are dealing with.

```
str(train$label)
```

```
##  int [1:42000] 1 0 1 4 0 0 7 3 5 3 ...
```

```
str(train$pixel0)
```

```
##  int [1:42000] 0 0 0 0 0 0 0 0 0 0 ...
```

Using the `str` function, we can see that all variables are integer values (The second row is representative of all 784 pixels, as they are the same type of variable).

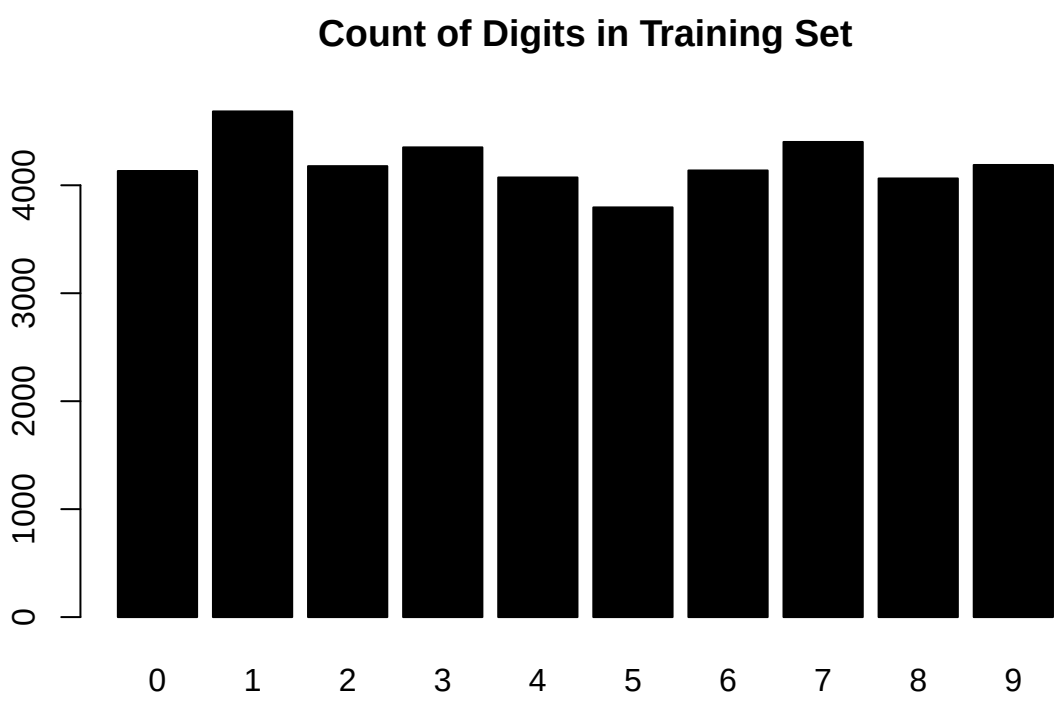
We will need to convert the target values into factor variables, which is required by the algorithms we will be using.

```
train$label <- as.factor(train$label)
str(train$label)
```

```
##  Factor w/ 10 levels "0","1","2","3",..: 2 1 2 5 1 1 8 4 6 4 ...
```

Next, we can begin to visualize the target value using a bar plot (`barplot`).

```
barplot(table(train[, 1]), col = "black", main = "Count of Digits in Training Set")
```



Using the bar plot, we can observe that the most common label is 1, and the least common is 5. All labels seem to have about 4000 observations.

Next, we can actually convert each row into a 28x28 matrix to further visualize the target variable. We can do this by first creating a copy of the training data without the labels. Then, we use the `matrix` function on each row in this new copy using the `lapply` function. Finally, we will print out an example of a matrix.

As we can see, creating this 28x28 pixel matrix allows us to see the general shape of the number 4, which is the fourth row in the training set.

Finally, we can actually produce images of the data by creating a scaled copy of the training data and producing a 10x12 matrix of examples for each label (0-9).

```
{r} # var <- t(train[,-1]) # row_matrices <- lapply(1:42000,
function(x) matrix(var[,x], nrow = 28)) #  # #Used to plot
image. Commented out for formatting. Will attach image to pdf.
# par(mfrow = c(1,1)) # t(row_matrices[[4]]) #
```

Commented out to help with knitting. Will attach image to pdf.

```
{r} # pixels.train <- train[,-1]/255 #  # train.scaled <- cbind('label'
= train$label, pixels.train) #  # par(mfrow=c(10, 12), pty='s',
mai=c(0.2, 0, 0, 0.1)) #  # for (lab in 0:9) { #   samp <-
train.scaled %>% #     filter(label == lab) #   for (i in 1:12)
{ #     img <- matrix(as.numeric(samp[i, -1]), 28, 28, byrow =
TRUE) #     image(t(img)[,28:1], axes = FALSE, col = grey(seq(1,
0, length = 256))) #     box(lty = 'solid') #   } # } #
```

From these images, we can clearly see the differences even between the same label. For example, in the 1 row, some of the images are slanted. Some images are neat, while some are a bit messier. Some images have thin handwriting, while others have thicker handwriting. This will almost certainly have some effect on the overall accuracy of the models we create. However, we can also see that, for the most part, the digits are relatively centered. This will be a great help.

First Algorithm

First, we will use k-Nearest Neighbors.

The k Nearest-Neighbor algorithm is a machine learning classification algorithm. The algorithm itself requires three things: an existing data set, a distance metric (which can potentially different for different types of data) and a value k, which determines how many neighbors to look at when classifying unknown records. It is classified as a lazy learner, which means that it does not explicitly build a classification model. The algorithm classifies an unknown record by computing the distance from the unknown record to k nearest known records. It then (usually) takes a majority vote using the class labels from those nearest neighbors and classifies the unknown record. Some advantages of kNN are that it is relatively simple to use and understand. It also has a fast training phase. However, some disadvantages are that (as stated above) it does not produce a model and that it requires a careful and appropriate selection of a k value. Additionally, if some features (data set variables) have a much larger range of values in comparison to other features, distance measurements will be dominated by the features with larger ranges.

We decided to begin with kNN because of its simplicity. Our training data only contains numerical data and a factor target variable, which kNN should have no trouble processing. If anything, the creation of the kNN model will take quite a while, simply due to the sheer number of both examples and predictor variables. While kNN typically works best with scaled values, we will omit this step to begin with, as most of the columns contain similar ranges of numeric data.

First, we will create a smaller training dataset to directly see accuracy values here in RStudio. This smaller dataset will contain 5000 examples, which will further be divided into a training set (4000 examples) and a testing set (1000 values). We will then create run the kNN algorithm on this small training set and predict the values in the small testing set. Following this, we will run the kNN algorithm on the entire training data, split using the same train to test ratio used in the Kaggle Challenge.

Step 3 - Training A Model On The Data

First, we will create the training and testing data needed for each model.

```
# Set random seed
set.seed(330)

# Get 5000 random indices from the entire training set
small.test.train.indices <- sample(nrow(train), 5000, replace = FALSE)
```

```
# Split into training and testing for small dataset
small_train <- train[small.test.train.indices[1:4000], ]
small_test <- train[small.test.train.indices[4001:5000], ]

# Split into training and testing using entire dataset
fullSplit_train <- train[1:25200, ]
fullSplit_test <- train[25201:42000, ]
```

Next, we will run the `knn` function from the `FNN` library on our two sets of training and testing data. We will begin with an experimental k-value of 5. The reason for using FNN (Fast KNN) is because of the large number of predictor variables. We are able to more efficiently process the data with very little difference in accuracy.

```
library(class)
library(FNN)
```

```
##
## Attaching package: 'FNN'
```

```
## The following objects are masked from 'package:class':
##
##     knn, knn.cv
```

```
knn.pred <- FNN::knn(small_train[, -1], small_test[, -1], cl = small_train$label,
    k = 5)
knn.full.pred <- FNN::knn(fullSplit_train[, -1], fullSplit_test[, -1], cl = fullSplit_train$label,
    k = 5)
```

Step 4 - Evaluating Model Performance

Now, using the factor vectors produced by the KNN function, we can view how accurate our predictions are using confusion matrices.

```
confusionMatrix(knn.pred, small_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 102   0   3   1   0   0   2   0   3   1
##          1   0 106   3   2   5   3   1   2   3   1
##          2   0   2  80   0   0   0   1   0   0   0
##          3   0   0   1 104   0   3   0   0   7   1
##          4   0   0   1   0  96   0   0   0   1   3
##          5   1   0   0   1   0  84   1   0   4   0
##          6   0   0   1   0   1   1  89   0   1   0
##          7   0   0   1   0   0   0   0 102   1   1
##          8   0   0   3   0   0   0   0   0  68   0
##          9   0   0   0   0   3   0   0   2   1  96
##
```

```
## Overall Statistics
##
##                Accuracy : 0.927
##                  95% CI : (0.9091, 0.9423)
##     No Information Rate : 0.108
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9188
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9903   0.9815   0.8602   0.9630   0.9143   0.9231
## Specificity            0.9889   0.9776   0.9967   0.9865   0.9944   0.9923
## Pos Pred Value         0.9107   0.8413   0.9639   0.8966   0.9505   0.9231
## Neg Pred Value         0.9989   0.9977   0.9858   0.9955   0.9900   0.9923
## Prevalence             0.1030   0.1080   0.0930   0.1080   0.1050   0.0910
## Detection Rate         0.1020   0.1060   0.0800   0.1040   0.0960   0.0840
## Detection Prevalence   0.1120   0.1260   0.0830   0.1160   0.1010   0.0910
## Balanced Accuracy      0.9896   0.9795   0.9285   0.9748   0.9543   0.9577
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9468   0.9623   0.7640   0.9320
## Specificity            0.9956   0.9966   0.9967   0.9933
## Pos Pred Value         0.9570   0.9714   0.9577   0.9412
## Neg Pred Value         0.9945   0.9955   0.9774   0.9922
## Prevalence             0.0940   0.1060   0.0890   0.1030
## Detection Rate         0.0890   0.1020   0.0680   0.0960
## Detection Prevalence   0.0930   0.1050   0.0710   0.1020
## Balanced Accuracy      0.9712   0.9795   0.8804   0.9627
```

From the confusion matrix above, we can observe that the predictions of our small test were fairly accurate. We achieved an accuracy of 92.7 by simply running the FNN function on our unmodified data. From the matrix itself, the label that was misclassified the most times was the number 8. It was misclassified as a 3 seven times. This is reasonable, as the two digits have a similar form. Next, we can create a confusion matrix for our large data test.

```
confusionMatrix(knn.full.pred, fullSplit_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0 1666    0   19    2    0    7   12    1    6   11
##          1    0 1861   21    7   17    5    3   19   33    7
##          2    1    3 1555   12    0    0    1    7    3    2
##          3    0    0    3 1656    0   25    0    0   26   12
##          4    0    1    1    0 1542    2    1    5    9   20
##          5    2    0    5   17    0 1440    7    0   28    5
##          6    5    1    3    0    6   26 1623    0    7    0
##          7    0    2   30   12    1    2    0 1729    3   31
##          8    0    0    1   12    0    2    0    0 1513    3
```

```
##          9    0    0    1   10   38    8    0   18   21 1604
##
## Overall Statistics
##
##                Accuracy : 0.9636
##                  95% CI : (0.9607, 0.9664)
##     No Information Rate : 0.1112
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9596
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.99522   0.9963  0.94875  0.95833  0.96135  0.94924
## Specificity           0.99617   0.9925  0.99809  0.99562  0.99743  0.99581
## Pos Pred Value        0.96636   0.9432  0.98169  0.96167  0.97533  0.95745
## Neg Pred Value        0.99947   0.9995  0.99448  0.99522  0.99593  0.99497
## Prevalence            0.09964   0.1112  0.09756  0.10286  0.09548  0.09030
## Detection Rate        0.09917   0.1108  0.09256  0.09857  0.09179  0.08571
## Detection Prevalence  0.10262   0.1174  0.09429  0.10250  0.09411  0.08952
## Balanced Accuracy     0.99569   0.9944  0.97342  0.97698  0.97939  0.97253
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.98543   0.9719  0.91753  0.94631
## Specificity           0.99683   0.9946  0.99881  0.99364
## Pos Pred Value        0.97127   0.9552  0.98824  0.94353
## Neg Pred Value        0.99841   0.9967  0.99109  0.99397
## Prevalence            0.09804   0.1059  0.09815  0.10089
## Detection Rate        0.09661   0.1029  0.09006  0.09548
## Detection Prevalence  0.09946   0.1077  0.09113  0.10119
## Balanced Accuracy     0.99113   0.9833  0.95817  0.96998
```

Our knn predictions were even more accurate this time! We achieved an accuracy of 96.36, using the same FNN parameters from the small test. It stands to reason that the FNN predictions for the entire testing data might be even more accurate, or at least the same. This time, it seems that 4 was the most misclassified digit. It was misclassified as a 9 thirty-eight times.

Step 5 - Improving Model Performance

Our initial plan to identify potentially better k-values was to use the `knnEval` function from the `chemometrics` in order to see CV error for a variety of k-values. However, when dealing with large datasets as we are now, it is difficult to produce a useful graph. As a rule of thumb, the optimal value of k is typically found by taking the square root of the total number of rows. We can try this method and see if it produces better results.

```
small.k <- ceiling(sqrt(5000))
large.k <- ceiling(sqrt(42000))
```

We can pass in these values for the k parameter.

```
knn.pred.k <- FNN::knn(small_train[, -1], small_test[, -1], cl = small_train$label,
    k = small.k)
knn.full.pred.k <- FNN::knn(fullSplit_train[, -1], fullSplit_test[, -1], cl = fullSplit_train$label,
    k = large.k)
```

Let's try printing out our results again using confusion matrices.

```
confusionMatrix(knn.pred.k, small_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 101   0   3   0   0   0   3   0   3   1
##          1   0 107  21  12   6   9   5   6  11   3
##          2   0   1  61   1   0   0   0   0   1   0
##          3   0   0   1  89   0   8   0   0   7   1
##          4   0   0   2   0  85   0   0   0   1   1
##          5   1   0   0   1   0  67   2   0   1   0
##          6   0   0   1   1   2   4  84   0   3   0
##          7   0   0   2   1   0   1   0  98   2   3
##          8   0   0   2   1   0   0   0   0  56   0
##          9   1   0   0   2  12   2   0   2   4  94
##
## Overall Statistics
##
##                Accuracy : 0.842
##                  95% CI : (0.8179, 0.8641)
##     No Information Rate : 0.108
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8241
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9806   0.9907   0.6559   0.8241   0.8095   0.7363
## Specificity            0.9889   0.9182   0.9967   0.9809   0.9955   0.9945
## Pos Pred Value         0.9099   0.5944   0.9531   0.8396   0.9551   0.9306
## Neg Pred Value         0.9978   0.9988   0.9658   0.9787   0.9780   0.9741
## Prevalence             0.1030   0.1080   0.0930   0.1080   0.1050   0.0910
## Detection Rate         0.1010   0.1070   0.0610   0.0890   0.0850   0.0670
## Detection Prevalence   0.1110   0.1800   0.0640   0.1060   0.0890   0.0720
## Balanced Accuracy      0.9847   0.9545   0.8263   0.9025   0.9025   0.8654
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.8936   0.9245   0.6292   0.9126
## Specificity            0.9879   0.9899   0.9967   0.9744
## Pos Pred Value         0.8842   0.9159   0.9492   0.8034
## Neg Pred Value         0.9890   0.9910   0.9649   0.9898
## Prevalence             0.0940   0.1060   0.0890   0.1030
## Detection Rate         0.0840   0.0980   0.0560   0.0940
```

```
## Detection Prevalence   0.0950   0.1070   0.0590   0.1170
## Balanced Accuracy      0.9407   0.9572   0.8130   0.9435
```

```
confusionMatrix(knn.full.pred.k, fullSplit_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0 1635    0   34    4    1   18   24    1   14    9
##          1    3 1863  145   62   57   46   33   76  137   29
##          2    1    2 1331   10    0    0    0    3    3    2
##          3    0    1   15 1548    0   54    0    0   69   24
##          4    0    0   11    1 1403    8    9    5   10   19
##          5   10    0    5   21    0 1303    8    1   40    2
##          6   22    1   15   10   14   45 1572    0   12    2
##          7    1    0   48   22    2    4    1 1641    7   66
##          8    1    0   25   18    0    3    0    0 1290    3
##          9    1    1   10   32  127   36    0   52   67 1539
##
## Overall Statistics
##
##                Accuracy : 0.9003
##                  95% CI : (0.8957, 0.9048)
##     No Information Rate : 0.1112
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8891
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.97670   0.9973  0.81208  0.89583  0.87469  0.85893
## Specificity           0.99306   0.9606  0.99861  0.98919  0.99585  0.99431
## Pos Pred Value        0.93966   0.7601  0.98447  0.90473  0.95703  0.93741
## Neg Pred Value        0.99741   0.9997  0.98006  0.98807  0.98689  0.98611
## Prevalence            0.09964   0.1112  0.09756  0.10286  0.09548  0.09030
## Detection Rate        0.09732   0.1109  0.07923  0.09214  0.08351  0.07756
## Detection Prevalence  0.10357   0.1459  0.08048  0.10185  0.08726  0.08274
## Balanced Accuracy     0.98488   0.9790  0.90535  0.94251  0.93527  0.92662
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.95446  0.92243  0.78229  0.90796
## Specificity           0.99201  0.98995  0.99670  0.97842
## Pos Pred Value        0.92853  0.91574  0.96269  0.82520
## Neg Pred Value        0.99504  0.99080  0.97678  0.98955
## Prevalence            0.09804  0.10589  0.09815  0.10089
## Detection Rate        0.09357  0.09768  0.07679  0.09161
## Detection Prevalence  0.10077  0.10667  0.07976  0.11101
## Balanced Accuracy     0.97324  0.95619  0.88950  0.94319
```

Unfortunately, our predictions using the rule of thumb method from the textbook were not as accurate as

our initial predictions. We got an accuracy of 84.2 for the small test and 90.03 for the large test. Still, it seems that our larger dataset was affected less by the change in k-value.

The book suggests that as the size of a dataset increases, the value of k becomes less important. Keeping this in mind, we would like to try one more k-value (3). If the statement from the book is true, then decreasing the value for k should not significantly alter the accuracy for the larger dataset. Let's test this.

```
knn.pred.3 <- FNN::knn(small_train[, -1], small_test[, -1], cl = small_train$label,
    k = 3)
knn.full.pred.3 <- FNN::knn(fullSplit_train[, -1], fullSplit_test[, -1], cl = fullSplit_train$label,
    k = 3)
```

Finally, let's print out the accuracies one final time.

```
confusionMatrix(knn.pred.3, small_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 102   0   2   1   0   1   1   0   2   1
##          1   0 106   3   1   4   3   1   2   2   1
##          2   0   2  82   0   0   0   1   0   3   0
##          3   0   0   1 104   0   6   0   0   5   0
##          4   0   0   1   0  93   0   0   0   1   4
##          5   1   0   0   0   0  80   1   0   4   1
##          6   0   0   0   0   1   1  90   0   0   0
##          7   0   0   3   0   0   0   0 101   1   1
##          8   0   0   1   0   0   0   0   0  70   0
##          9   0   0   0   2   7   0   0   3   1  95
##
## Overall Statistics
##
##                Accuracy : 0.923
##                  95% CI : (0.9047, 0.9388)
##     No Information Rate : 0.108
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9144
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9903   0.9815   0.8817   0.9630   0.8857   0.8791
## Specificity            0.9911   0.9809   0.9934   0.9865   0.9933   0.9923
## Pos Pred Value         0.9273   0.8618   0.9318   0.8966   0.9394   0.9195
## Neg Pred Value         0.9989   0.9977   0.9879   0.9955   0.9867   0.9880
## Prevalence             0.1030   0.1080   0.0930   0.1080   0.1050   0.0910
## Detection Rate         0.1020   0.1060   0.0820   0.1040   0.0930   0.0800
## Detection Prevalence   0.1100   0.1230   0.0880   0.1160   0.0990   0.0870
## Balanced Accuracy      0.9907   0.9812   0.9376   0.9748   0.9395   0.9357
##                      Class: 6 Class: 7 Class: 8 Class: 9
```

```
## Sensitivity            0.9574   0.9528   0.7865   0.9223
## Specificity            0.9978   0.9944   0.9989   0.9855
## Pos Pred Value         0.9783   0.9528   0.9859   0.8796
## Neg Pred Value         0.9956   0.9944   0.9795   0.9910
## Prevalence             0.0940   0.1060   0.0890   0.1030
## Detection Rate         0.0900   0.1010   0.0700   0.0950
## Detection Prevalence   0.0920   0.1060   0.0710   0.1080
## Balanced Accuracy      0.9776   0.9736   0.8927   0.9539
```

```
confusionMatrix(knn.full.pred.3, fullSplit_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0 1669    0   17    2    0    8   10    2    9   11
##          1    0 1862   14    5   16    3    4   17   30    6
##          2    1    2 1563   15    0    0    0    5    9    2
##          3    0    2    5 1660    0   32    0    1   27   15
##          4    0    1    1    0 1535    3    1    4    8   20
##          5    1    0    2   16    0 1434    6    0   25    6
##          6    3    0    3    0    7   26 1626    0    7    0
##          7    0    1   31   11    2    0    0 1727    4   30
##          8    0    0    1   11    0    3    0    0 1508    3
##          9    0    0    2    8   44    8    0   23   22 1602
##
## Overall Statistics
##
##                Accuracy : 0.9635
##                  95% CI : (0.9605, 0.9662)
##     No Information Rate : 0.1112
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9594
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.99701   0.9968  0.95363  0.96065  0.95698  0.94529
## Specificity           0.99610   0.9936  0.99776  0.99456  0.99750  0.99634
## Pos Pred Value        0.96586   0.9515  0.97871  0.95293  0.97584  0.96242
## Neg Pred Value        0.99967   0.9996  0.99500  0.99548  0.99547  0.99458
## Prevalence            0.09964   0.1112  0.09756  0.10286  0.09548  0.09030
## Detection Rate        0.09935   0.1108  0.09304  0.09881  0.09137  0.08536
## Detection Prevalence  0.10286   0.1165  0.09506  0.10369  0.09363  0.08869
## Balanced Accuracy     0.99656   0.9952  0.97569  0.97760  0.97724  0.97081
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.98725   0.9708  0.91449  0.94513
## Specificity           0.99696   0.9947  0.99881  0.99292
## Pos Pred Value        0.97249   0.9563  0.98820  0.93739
## Neg Pred Value        0.99861   0.9965  0.99077  0.99384
## Prevalence            0.09804   0.1059  0.09815  0.10089
```

```
## Detection Rate         0.09679   0.1028  0.08976  0.09536
## Detection Prevalence   0.09952   0.1075  0.09083  0.10173
## Balanced Accuracy      0.99211   0.9828  0.95665  0.96902
```

Using a slightly lower k-value did not alter the initial predictions significantly. We achieved very similar accuracies (92.3 and 96.35) for the small and large data. If anything, the accuracy of both decreased by a minuscule amount. Therefore, we will be utilizing a k value of 5 when we run the FNN function on the entire testing data.

Second Algorithm

Decision trees are classifier models "that utilize a tree structure to model the relationships among the features and the potential outcomes" (Lantz 126). The tree itself is made up of a root node (the beginning of the model), decision nodes (typically Boolean tests), branches (which indicate potential outcomes of a decision), and leaf nodes (which indicate that a classification can be made). They are built using a divide and conquer tactic, in which data is recursively split into smaller subsets until the data in each subset is homogeneous. Decision trees themselves can be classified as greedy learners, because they "use data on a first-come, first-served basis" (Lantz 156). Decision trees also have the potential to overfit data by becoming very large and complex. As such, decision trees can be pruned (reduced in size and complexity such that the model is a better predictor of unseen data).

Step 3 - Training A Model On The Data

To train our decision tree models, we will use the `C5.0` function from the `C50` package. While we might typically produce a summary of the model, we will refrain from doing so in this case (simply due to the large number of predictors and length of summary). We will once again use the same small and large datasets we created above. As a reminder, the small dataset holds 5000 examples, with 4000 going to training and 1000 going to testing. The large dataset holds all 42000 examples, with 25200 going to training and 16800 going to testing. This is the same ratio as the entire training (42000) and entire testing (28000).

```
library(C50)

small.tree.model <- C5.0(small_train[, -1], small_train[, 1])
large.tree.model <- C5.0(fullSplit_train[, -1], fullSplit_train[, 1])
```

Step 4 - Evaluating Model Performance

We can again create predictions for our models and use confusion matrices to view how our models did.

```
small.tree.pred <- predict(small.tree.model, small_test)
large.tree.pred <- predict(large.tree.model, fullSplit_test)
```

```
confusionMatrix(small.tree.pred, small_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0  85   0   3   3   1   1   5   1   3   1
##          1   0 101   2   4   0   2   1   1   4   3
```

```
##          2  11   2  66   7   0   5   1   3   3   2
##          3   3   0   2  75   3  16   2   1   3   2
##          4   1   1   5   1  77   1   2   1   2  10
##          5   0   0   2   7   5  55   4   3   3   5
##          6   2   1   3   1   4   2  74   0   2   1
##          7   0   0   2   3   0   0   1  85   0   3
##          8   1   3   3   4   6   6   3   1  66   1
##          9   0   0   5   3   9   3   1  10   3  75
##
## Overall Statistics
##
##                Accuracy : 0.759
##                  95% CI : (0.7313, 0.7852)
##     No Information Rate : 0.108
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7321
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.8252   0.9352   0.7097   0.6944   0.7333   0.6044
## Specificity            0.9799   0.9809   0.9625   0.9641   0.9732   0.9681
## Pos Pred Value         0.8252   0.8559   0.6600   0.7009   0.7624   0.6548
## Neg Pred Value         0.9799   0.9921   0.9700   0.9630   0.9689   0.9607
## Prevalence             0.1030   0.1080   0.0930   0.1080   0.1050   0.0910
## Detection Rate         0.0850   0.1010   0.0660   0.0750   0.0770   0.0550
## Detection Prevalence   0.1030   0.1180   0.1000   0.1070   0.1010   0.0840
## Balanced Accuracy      0.9026   0.9581   0.8361   0.8293   0.8533   0.7862
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.7872   0.8019   0.7416   0.7282
## Specificity            0.9823   0.9899   0.9693   0.9621
## Pos Pred Value         0.8222   0.9043   0.7021   0.6881
## Neg Pred Value         0.9780   0.9768   0.9746   0.9686
## Prevalence             0.0940   0.1060   0.0890   0.1030
## Detection Rate         0.0740   0.0850   0.0660   0.0750
## Detection Prevalence   0.0900   0.0940   0.0940   0.1090
## Balanced Accuracy      0.8848   0.8959   0.8554   0.8451
```

```
confusionMatrix(large.tree.pred, fullSplit_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0 1563    0   33    8   10   14   28   20   15   15
##          1    4 1780   23   19    8   15   10    3   50   13
##          2   13   21 1385   77   26   18   28   38   35   15
##          3   14   16   49 1381    6  101    7   19   65   28
##          4    9    4   20   16 1395   25   29   22   43   81
##          5   18    4   12   92   25 1232   32    9   48   32
##          6   23    8   40    9   25   32 1484    1   24    5
```

```
##          7    8   11   28   25   12    4    4 1570   11   57
##          8   15   23   38   61   28   45   22    6 1302   37
##          9    7    1   11   40   69   31    3   91   56 1412
##
## Overall Statistics
##
##                Accuracy : 0.8633
##                  95% CI : (0.858, 0.8685)
##     No Information Rate : 0.1112
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8481
##
##  Mcnemar's Test P-Value : 2.344e-06
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.93369   0.9529  0.84503   0.7992  0.86970  0.81213
## Specificity           0.99055   0.9903  0.98213   0.9798  0.98361  0.98220
## Pos Pred Value        0.91618   0.9247  0.83635   0.8191  0.84854  0.81915
## Neg Pred Value        0.99265   0.9941  0.98323   0.9770  0.98621  0.98137
## Prevalence            0.09964   0.1112  0.09756   0.1029  0.09548  0.09030
## Detection Rate        0.09304   0.1060  0.08244   0.0822  0.08304  0.07333
## Detection Prevalence  0.10155   0.1146  0.09857   0.1004  0.09786  0.08952
## Balanced Accuracy     0.96212   0.9716  0.91358   0.8895  0.92666  0.89717
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.90103  0.88252  0.78957  0.83304
## Specificity           0.98898  0.98935  0.98185  0.97954
## Pos Pred Value        0.89885  0.90751  0.82562  0.82045
## Neg Pred Value        0.98924  0.98613  0.97721  0.98123
## Prevalence            0.09804  0.10589  0.09815  0.10089
## Detection Rate        0.08833  0.09345  0.07750  0.08405
## Detection Prevalence  0.09827  0.10298  0.09387  0.10244
## Balanced Accuracy     0.94501  0.93593  0.88571  0.90629
```

Immediately, we can see the accuracy of our models dropped in comparison to the previous KNN predictions. We got an accuracy of 75.9 for the small data and 86.33 for the large data. While this isn't too bad at all, it is definitely a downgrade. Something else to note is that, like our previous predictions, the accuracy for the large dataset is higher than the accuracy for the small data. This could simply be due to the fact that there is less data to train from, which leads to a worse model.

Step 5 - Improving Model Performance

We can attempt to improve our models using a method called adaptive boosting. This is a process in which many decision trees are built and the trees vote on the best label for each example. The `C5.0` function takes as an optional parameter `trials`, which is used to determine the number of decision tree models built. According to the textbook, 10 trials has become the "de facto standard", so this is the first value we will try.

```
small.tree.model.boost.10 <- C5.0(small_train[, -1], small_train[, 1], trials = 10)
large.tree.model.boost.10 <- C5.0(fullSplit_train[, -1], fullSplit_train[, 1], trials = 10)
```

We can now create predictions and confusion matrices for our boosted models.

```
small.tree.pred.boost.10 <- predict(small.tree.model.boost.10, small_test)
large.tree.pred.boost.10 <- predict(large.tree.model.boost.10, fullSplit_test)
```

```
confusionMatrix(small.tree.pred.boost.10, small_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 101   0   2   0   0   2   0   1   1   2
##          1   0 106   1   2   0   0   0   1   1   2
##          2   0   2  81   3   0   0   0   1   1   0
##          3   0   0   1  92   0   4   0   1   0   1
##          4   0   0   4   1  95   2   0   1   1   5
##          5   0   0   0   3   1  81   2   0   0   1
##          6   1   0   2   0   3   1  91   0   1   0
##          7   0   0   1   3   0   0   0  95   1   1
##          8   1   0   1   2   1   0   1   1  82   1
##          9   0   0   0   2   5   1   0   5   1  90
##
## Overall Statistics
##
##                Accuracy : 0.914
##                  95% CI : (0.8949, 0.9306)
##     No Information Rate : 0.108
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9044
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9806   0.9815   0.8710   0.8519   0.9048   0.8901
## Specificity            0.9911   0.9922   0.9923   0.9922   0.9844   0.9923
## Pos Pred Value         0.9266   0.9381   0.9205   0.9293   0.8716   0.9205
## Neg Pred Value         0.9978   0.9977   0.9868   0.9822   0.9888   0.9890
## Prevalence             0.1030   0.1080   0.0930   0.1080   0.1050   0.0910
## Detection Rate         0.1010   0.1060   0.0810   0.0920   0.0950   0.0810
## Detection Prevalence   0.1090   0.1130   0.0880   0.0990   0.1090   0.0880
## Balanced Accuracy      0.9858   0.9868   0.9316   0.9220   0.9446   0.9412
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9681   0.8962   0.9213   0.8738
## Specificity            0.9912   0.9933   0.9912   0.9844
## Pos Pred Value         0.9192   0.9406   0.9111   0.8654
## Neg Pred Value         0.9967   0.9878   0.9923   0.9855
## Prevalence             0.0940   0.1060   0.0890   0.1030
## Detection Rate         0.0910   0.0950   0.0820   0.0900
## Detection Prevalence   0.0990   0.1010   0.0900   0.1040
## Balanced Accuracy      0.9796   0.9448   0.9563   0.9291
```

```
confusionMatrix(large.tree.pred.boost.10, fullSplit_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0 1641    0   12    2    5   15   17    6    8    7
##          1    0 1821    3    4    2    3    5    3   26    4
##          2    6   17 1560   33    8    2    9   23   13    6
##          3    1    3   22 1606    1   37    2    7   12   25
##          4    1    5    4    1 1515    4   12   17   11   43
##          5    4    1    1   32    5 1395   14    4   15   15
##          6   10    8    6    1   13   26 1573    0    7    0
##          7    1    4    8   17    4    1    0 1656    7   34
##          8    9    9   19   20   12   16   15    3 1531   13
##          9    1    0    4   12   39   18    0   60   19 1548
##
## Overall Statistics
##
##                Accuracy : 0.9432
##                  95% CI : (0.9396, 0.9467)
##     No Information Rate : 0.1112
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9369
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.98029   0.9748  0.95180   0.9294  0.94451  0.91958
## Specificity           0.99524   0.9967  0.99228   0.9927  0.99355  0.99405
## Pos Pred Value        0.95797   0.9733  0.93023   0.9359  0.93924  0.93876
## Neg Pred Value        0.99781   0.9969  0.99478   0.9919  0.99414  0.99203
## Prevalence            0.09964   0.1112  0.09756   0.1029  0.09548  0.09030
## Detection Rate        0.09768   0.1084  0.09286   0.0956  0.09018  0.08304
## Detection Prevalence  0.10196   0.1114  0.09982   0.1021  0.09601  0.08845
## Balanced Accuracy     0.98776   0.9857  0.97204   0.9610  0.96903  0.95681
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.95507  0.93086  0.92844  0.91327
## Specificity           0.99531  0.99494  0.99234  0.98987
## Pos Pred Value        0.95681  0.95612  0.92957  0.91005
## Neg Pred Value        0.99512  0.99184  0.99221  0.99026
## Prevalence            0.09804  0.10589  0.09815  0.10089
## Detection Rate        0.09363  0.09857  0.09113  0.09214
## Detection Prevalence  0.09786  0.10310  0.09804  0.10125
## Balanced Accuracy     0.97519  0.96290  0.96039  0.95157
```

Right off the bat, we can see a drastic accuracy improvement through the utilization of adaptive boosting. We achieved an accuracy of 91.4 for the small data and 94.32 for the large data. The jump in accuracy was less for the large data. Therefore, we would like to see if simply increasing the number of trials will have any further significant effect on the models.

```
small.tree.model.boost.20 <- C5.0(small_train[, -1], small_train[, 1], trials = 20)
large.tree.model.boost.20 <- C5.0(fullSplit_train[, -1], fullSplit_train[, 1], trials = 20)
```

Finally, we can create predictions and confusion matrices for these models with 20 trials.

```
small.tree.pred.boost.20 <- predict(small.tree.model.boost.20, small_test)
large.tree.pred.boost.20 <- predict(large.tree.model.boost.20, fullSplit_test)
```

```
confusionMatrix(small.tree.pred.boost.20, small_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 102   0   1   0   0   1   0   1   1   1
##          1   0 105   0   0   0   0   0   1   1   2
##          2   0   3  85   2   0   0   0   1   1   0
##          3   0   0   2  98   0   7   0   1   1   1
##          4   0   0   1   0  94   1   0   2   1   3
##          5   0   0   0   3   1  78   2   0   0   0
##          6   0   0   1   0   3   1  90   0   1   0
##          7   0   0   1   2   0   0   0  93   1   1
##          8   1   0   2   1   1   0   2   2  81   1
##          9   0   0   0   2   6   3   0   5   1  94
##
## Overall Statistics
##
##                Accuracy : 0.92
##                  95% CI : (0.9014, 0.9361)
##     No Information Rate : 0.108
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9111
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9903   0.9722   0.9140   0.9074   0.8952   0.8571
## Specificity            0.9944   0.9955   0.9923   0.9865   0.9911   0.9934
## Pos Pred Value         0.9533   0.9633   0.9239   0.8909   0.9216   0.9286
## Neg Pred Value         0.9989   0.9966   0.9912   0.9888   0.9878   0.9858
## Prevalence             0.1030   0.1080   0.0930   0.1080   0.1050   0.0910
## Detection Rate         0.1020   0.1050   0.0850   0.0980   0.0940   0.0780
## Detection Prevalence   0.1070   0.1090   0.0920   0.1100   0.1020   0.0840
## Balanced Accuracy      0.9924   0.9839   0.9531   0.9470   0.9431   0.9253
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9574   0.8774   0.9101   0.9126
## Specificity            0.9934   0.9944   0.9890   0.9810
## Pos Pred Value         0.9375   0.9490   0.8901   0.8468
## Neg Pred Value         0.9956   0.9856   0.9912   0.9899
```

```
## Prevalence             0.0940   0.1060   0.0890   0.1030
## Detection Rate         0.0900   0.0930   0.0810   0.0940
## Detection Prevalence   0.0960   0.0980   0.0910   0.1110
## Balanced Accuracy      0.9754   0.9359   0.9496   0.9468
```

```
confusionMatrix(large.tree.pred.boost.20, fullSplit_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0 1642    0   10    2    7   17   14    6    7    9
##          1    0 1827    1    2    2    1    2    4   26    3
##          2    3   15 1583   32    6    1    1   22    6    5
##          3    1    3    8 1620    1   28    0    4   10   22
##          4    3    6    5    2 1530    1    6   14    9   41
##          5    2    1    3   23    2 1410   14    5   13   11
##          6    7    8    5    2   11   24 1597    0    8    0
##          7    2    2    8   19    2    0    0 1670    5   29
##          8   14    5   14   16   13   18   13    4 1549   12
##          9    0    1    2   10   30   17    0   50   16 1563
##
## Overall Statistics
##
##                Accuracy : 0.9518
##                  95% CI : (0.9485, 0.955)
##     No Information Rate : 0.1112
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9465
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.98088   0.9781  0.96583  0.93750  0.95387  0.92947
## Specificity           0.99524   0.9973  0.99400  0.99489  0.99427  0.99516
## Pos Pred Value        0.95799   0.9781  0.94564  0.95463  0.94620  0.95013
## Neg Pred Value        0.99788   0.9973  0.99630  0.99285  0.99513  0.99301
## Prevalence            0.09964   0.1112  0.09756  0.10286  0.09548  0.09030
## Detection Rate        0.09774   0.1087  0.09423  0.09643  0.09107  0.08393
## Detection Prevalence  0.10202   0.1112  0.09964  0.10101  0.09625  0.08833
## Balanced Accuracy     0.98806   0.9877  0.97992  0.96620  0.97407  0.96231
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.96964   0.9387  0.93936  0.92212
## Specificity           0.99571   0.9955  0.99281  0.99166
## Pos Pred Value        0.96089   0.9614  0.93426  0.92540
## Neg Pred Value        0.99670   0.9928  0.99340  0.99126
## Prevalence            0.09804   0.1059  0.09815  0.10089
## Detection Rate        0.09506   0.0994  0.09220  0.09304
## Detection Prevalence  0.09893   0.1034  0.09869  0.10054
## Balanced Accuracy     0.98268   0.9671  0.96608  0.95689
```

By increasing the number of trials, we were able to increase the accuracy, though not by much. We reached accuracies of 92 for the small data and 95.18 for the large data. While the increase is there, the time it took to produce these models doesn't seem worth the small increase in accuracy, especially when we have achieved higher accuracies using FNN. Regardless, these decision tree models performed very well.

Ensemble Approach

For our ensemble approach, we are choosing to use to go with a random forest (also known a decision tree forest). This approach "combines the base principles of bagging with random feature selection to add additional diversity to the decision tree models" (Lantz 367). We chose to go with random forests for a couple of reasons. First, we previously looked at decision trees. As random forests are essentially many decision trees, it will be interesting to see how the two compare in terms of accuracy. Additionally, random forests are less likely to overfit the training data, which will be helpful for our Kaggle score. Finally, random forests work well with large datasets, which we have.

Step 3 - Training A Model On The Data

To create our randomForest model, we will use the `randomForest` function from the `randomForest` package. The function takes in an `ntree` parameter, which determines how many decision trees are created. We will start off with a small forest (ntree = 10). Again, we will use the function on both the small and large datasets we created above.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
small.random.forest.model.10 <- randomForest(label ~ ., data = small_train, ntree = 10)
forest.model.10 <- randomForest(label ~ ., data = fullSplit_train, ntree = 10)
```

Step 4 - Evaluating Model Performance

Now, we can create predictions and confusion matrices like we did above.

```
small.forest.preds.10 <- predict(small.random.forest.model.10, small_test)
forest.preds.10 <- predict(forest.model.10, fullSplit_test)
```

```
confusionMatrix(small.forest.preds.10, small_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 100   0   5   0   0   1   2   2   1   2
##          1   0 106   3   3   0   1   1   2   1   1
##          2   1   2  78   4   0   0   0   0   2   0
##          3   0   0   2  90   2   6   0   2   9   1
##          4   0   0   0   0  93   1   0   3   2   6
##          5   2   0   0   5   0  74   5   0   0   2
##          6   0   0   3   2   4   4  86   0   0   1
##          7   0   0   0   2   0   1   0  89   0   5
##          8   0   0   2   1   0   2   0   1  71   1
##          9   0   0   0   1   6   1   0   7   3  84
##
## Overall Statistics
##
##                Accuracy : 0.871
##                  95% CI : (0.8486, 0.8912)
##     No Information Rate : 0.108
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8565
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9709   0.9815   0.8387   0.8333   0.8857   0.8132
## Specificity            0.9855   0.9865   0.9901   0.9753   0.9866   0.9846
## Pos Pred Value         0.8850   0.8983   0.8966   0.8036   0.8857   0.8409
## Neg Pred Value         0.9966   0.9977   0.9836   0.9797   0.9866   0.9814
## Prevalence             0.1030   0.1080   0.0930   0.1080   0.1050   0.0910
## Detection Rate         0.1000   0.1060   0.0780   0.0900   0.0930   0.0740
## Detection Prevalence   0.1130   0.1180   0.0870   0.1120   0.1050   0.0880
## Balanced Accuracy      0.9782   0.9840   0.9144   0.9043   0.9362   0.8989
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9149   0.8396   0.7978   0.8155
## Specificity            0.9845   0.9911   0.9923   0.9799
## Pos Pred Value         0.8600   0.9175   0.9103   0.8235
## Neg Pred Value         0.9911   0.9812   0.9805   0.9788
## Prevalence             0.0940   0.1060   0.0890   0.1030
## Detection Rate         0.0860   0.0890   0.0710   0.0840
## Detection Prevalence   0.1000   0.0970   0.0780   0.1020
## Balanced Accuracy      0.9497   0.9153   0.8950   0.8977
```

```
confusionMatrix(forest.preds.10, fullSplit_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0 1633    0   12    3    7   16   18    5    6    9
```

```
##          1    1 1832    8   10    3    4    5    8   20    6
##          2    1   12 1538   38   11    3   13   21   24    9
##          3    3    4   11 1582    3   62    0    3   38   22
##          4    5    5    8    2 1503   14   12   19   18   60
##          5    6    2    9   29    1 1354   16    2   28    7
##          6   10    4   11    4   15   24 1569    0   18    3
##          7    3    3   13   18    4    1    1 1694    4   37
##          8   11    6   27   24    6   23   13    5 1465   18
##          9    1    0    2   18   51   16    0   22   28 1524
##
## Overall Statistics
##
##                Accuracy : 0.9342
##                  95% CI : (0.9303, 0.9379)
##     No Information Rate : 0.1112
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9268
##
##  Mcnemar's Test P-Value : 1.035e-12
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.97551   0.9807  0.93838  0.91551  0.93703  0.89255
## Specificity           0.99498   0.9956  0.99129  0.99031  0.99059  0.99346
## Pos Pred Value        0.95553   0.9657  0.92096  0.91551  0.91312  0.93122
## Neg Pred Value        0.99728   0.9976  0.99332  0.99031  0.99334  0.98938
## Prevalence            0.09964   0.1112  0.09756  0.10286  0.09548  0.09030
## Detection Rate        0.09720   0.1090  0.09155  0.09417  0.08946  0.08060
## Detection Prevalence  0.10173   0.1129  0.09940  0.10286  0.09798  0.08655
## Balanced Accuracy     0.98524   0.9882  0.96484  0.95291  0.96381  0.94300
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.95264   0.9522  0.88842  0.89912
## Specificity           0.99413   0.9944  0.99122  0.99086
## Pos Pred Value        0.94632   0.9528  0.91677  0.91697
## Neg Pred Value        0.99485   0.9943  0.98790  0.98870
## Prevalence            0.09804   0.1059  0.09815  0.10089
## Detection Rate        0.09339   0.1008  0.08720  0.09071
## Detection Prevalence  0.09869   0.1058  0.09512  0.09893
## Balanced Accuracy     0.97338   0.9733  0.93982  0.94499
```

Overall, the initial random forest models performed much better than the initial decision tree models. This makes sense, as random forests create multiple decision trees and use them to attempt to find the correct label. We got accuracies of 87.3 for the small data and 93.21 for the large data. Not too shabby.

Step 5 - Improving Model Performance

To improve our random forest models, we will increase the number of trees that are created. The default value is 500, so this is what we'll use.

```
small.random.forest.model <- randomForest(label ~ ., data = small_train, ntree = 500)
forest.model <- randomForest(label ~ ., data = fullSplit_train, ntree = 500)
```

Finally, we can create predictions and confusion matrices.

```
small.forest.preds <- predict(small.random.forest.model, small_test)
forest.preds <- predict(forest.model, fullSplit_test)
```

```
confusionMatrix(small.forest.preds, small_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 102   0   1   0   0   1   1   0   1   1
##          1   0 106   1   1   0   1   1   1   1   2
##          2   0   1  84   1   0   0   0   1   1   0
##          3   0   1   3 102   0   3   0   1   3   1
##          4   0   0   2   0 100   0   0   1   1   2
##          5   0   0   0   0   0  82   2   0   1   0
##          6   0   0   1   1   3   2  89   0   1   0
##          7   0   0   0   2   0   0   0  99   1   0
##          8   1   0   1   1   0   1   1   1  79   0
##          9   0   0   0   0   2   1   0   2   0  97
##
## Overall Statistics
##
##                Accuracy : 0.94
##                  95% CI : (0.9234, 0.9539)
##     No Information Rate : 0.108
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9333
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9903   0.9815   0.9032   0.9444   0.9524   0.9011
## Specificity            0.9944   0.9910   0.9956   0.9865   0.9933   0.9967
## Pos Pred Value         0.9533   0.9298   0.9545   0.8947   0.9434   0.9647
## Neg Pred Value         0.9989   0.9977   0.9901   0.9932   0.9944   0.9902
## Prevalence             0.1030   0.1080   0.0930   0.1080   0.1050   0.0910
## Detection Rate         0.1020   0.1060   0.0840   0.1020   0.1000   0.0820
## Detection Prevalence   0.1070   0.1140   0.0880   0.1140   0.1060   0.0850
## Balanced Accuracy      0.9924   0.9863   0.9494   0.9655   0.9728   0.9489
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9468   0.9340   0.8876   0.9417
## Specificity            0.9912   0.9966   0.9934   0.9944
## Pos Pred Value         0.9175   0.9706   0.9294   0.9510
## Neg Pred Value         0.9945   0.9922   0.9891   0.9933
## Prevalence             0.0940   0.1060   0.0890   0.1030
## Detection Rate         0.0890   0.0990   0.0790   0.0970
## Detection Prevalence   0.0970   0.1020   0.0850   0.1020
## Balanced Accuracy      0.9690   0.9653   0.9405   0.9681
```

```
confusionMatrix(forest.preds, fullSplit_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0 1652    0   10    2    2    6    9    2    1    9
##          1    0 1843    2    3    3    0    4    5   13    5
##          2    1    8 1583   25    2    2    1   18    7    7
##          3    1    7   10 1639    0   19    0    1   14   29
##          4    0    3    4    2 1549    5    5    4    6   22
##          5    1    1    4   15    0 1457   10    0   10    5
##          6    8    1    4    2    7   10 1612    0    6    0
##          7    0    2    8   14    1    2    0 1719    3   22
##          8   11    3   13   20    8   10    6    5 1567   12
##          9    0    0    1    6   32    6    0   25   22 1584
##
## Overall Statistics
##
##                Accuracy : 0.9646
##                  95% CI : (0.9617, 0.9673)
##     No Information Rate : 0.1112
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9606
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.98686   0.9866  0.96583  0.94850  0.96571  0.96045
## Specificity           0.99729   0.9977  0.99532  0.99463  0.99664  0.99699
## Pos Pred Value        0.97578   0.9814  0.95707  0.95291  0.96812  0.96939
## Neg Pred Value        0.99854   0.9983  0.99630  0.99410  0.99638  0.99608
## Prevalence            0.09964   0.1112  0.09756  0.10286  0.09548  0.09030
## Detection Rate        0.09833   0.1097  0.09423  0.09756  0.09220  0.08673
## Detection Prevalence  0.10077   0.1118  0.09845  0.10238  0.09524  0.08946
## Balanced Accuracy     0.99207   0.9921  0.98057  0.97156  0.98118  0.97872
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.97875   0.9663  0.95027  0.93451
## Specificity           0.99749   0.9965  0.99419  0.99391
## Pos Pred Value        0.97697   0.9706  0.94683  0.94511
## Neg Pred Value        0.99769   0.9960  0.99459  0.99266
## Prevalence            0.09804   0.1059  0.09815  0.10089
## Detection Rate        0.09595   0.1023  0.09327  0.09429
## Detection Prevalence  0.09821   0.1054  0.09851  0.09976
## Balanced Accuracy     0.98812   0.9814  0.97223  0.96421
```

Out of all the models we have created thus far, these random forest models have performed the best. The "large" random forest model beat out FNN, in terms of accuracy, by only 0.0004. However, it is important to note that FNN took much less time to run and produced almost identical results.

Analysis of Approaches

Analysis of Kaggle Results

```
final.preds.knn <- FNN::knn(train[, -1], test, cl = train$label, k = 5)
final.preds.tree <- C5.0(train[, -1], train[, 1], trials = 20)
final.preds.forest <- randomForest(label ~ ., data = train, ntree = 500)
```

```
final.preds.tree.values <- predict(final.preds.tree, test)
final.preds.forest.values <- predict(final.preds.forest, test)
```

```
knn.final <- data.frame(ImageId = seq(1, 28000), Label = final.preds.knn)
tree.final <- data.frame(ImageId = seq(1, 28000), Label = final.preds.tree.values)
forest.final <- data.frame(ImageId = seq(1, 28000), Label = final.preds.forest.values)
```

```
write.csv(knn.final, file = "knn_submit.csv", row.names = F)
write.csv(tree.final, file = "tree_submit.csv", row.names = F)
write.csv(forest.final, file = "forest_submit.csv", row.names = F)
```

Conclusions