HW 6 & 7

Victoria Haley

2023-05-15

HW 6 & 7

Section 1

Introduction

As both the capabilities and utilization of machine learning grows every day, so does the need for the precision and accuracy of the models used to make predictions. Considering that there are many different classification algorithms that can be used, how does one decide which, if any, model is best used for a specific task?

One popular use of machine learning algorithms is handwriting recognition. Handwriting styles, whether it be letters or numbers, are like fingerprints and vary widely among people, so the a computer being able to determine what is written is an incredible feat. As such, this report will attempt to build supervised learning models using both Decision Tree and Naive Bayes methods in order to compare which method is more appropriate for this classification task.

In order to work towards this goal, I will analyze the data set from Kaggle's Digit Recognizer competition. The data set consists of csv files of both training and testing files of "gray-scale images of hand-drawn digits, from zero through nine." Before any analysis can be done, the data must be loaded and processed, and explored a bit so that it can be understood a bit more. Then, samples will be taken from each training and testing set to use in the models to train the algorithm to recognize digits so that it can make predictions on the testing data.

The analysis will first use Decision Tree and Naive Bayes models on the data, and their respective results will be compared to determine which of the algorithms is more appropriate for this task. Afterwards, those results will be compared with Support Vector Machines, k Nearest Neighbor, and Random Forest.

Pre-processing

```
#load the data
library(readr)
digit_train <-
  read_csv("/Users/victoriahaley/Desktop/IST 707/Kaggle-digit-train-sample-small-1400.csv")
```

```
## Rows: 1400 Columns: 785
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (785): label, pixel0, pixel1, pixel2, pixel3, pixel4, pixel5, pixel6, pi...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
digit_test <-
  read_csv("/Users/victoriahaley/Desktop/IST 707/Kaggle-digit-test-sample1000.csv")
```

```
## Rows: 999 Columns: 785
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (1): label
## dbl (784): pixel0, pixel1, pixel2, pixel3, pixel4, pixel5, pixel6, pixel7, p...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
#view structure of training data
#update the label variable from numeric to label so that the digits are factors
digit_train$label <- as.factor(digit_train$label)
```

```
#load packages
library(DataExplorer)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
```

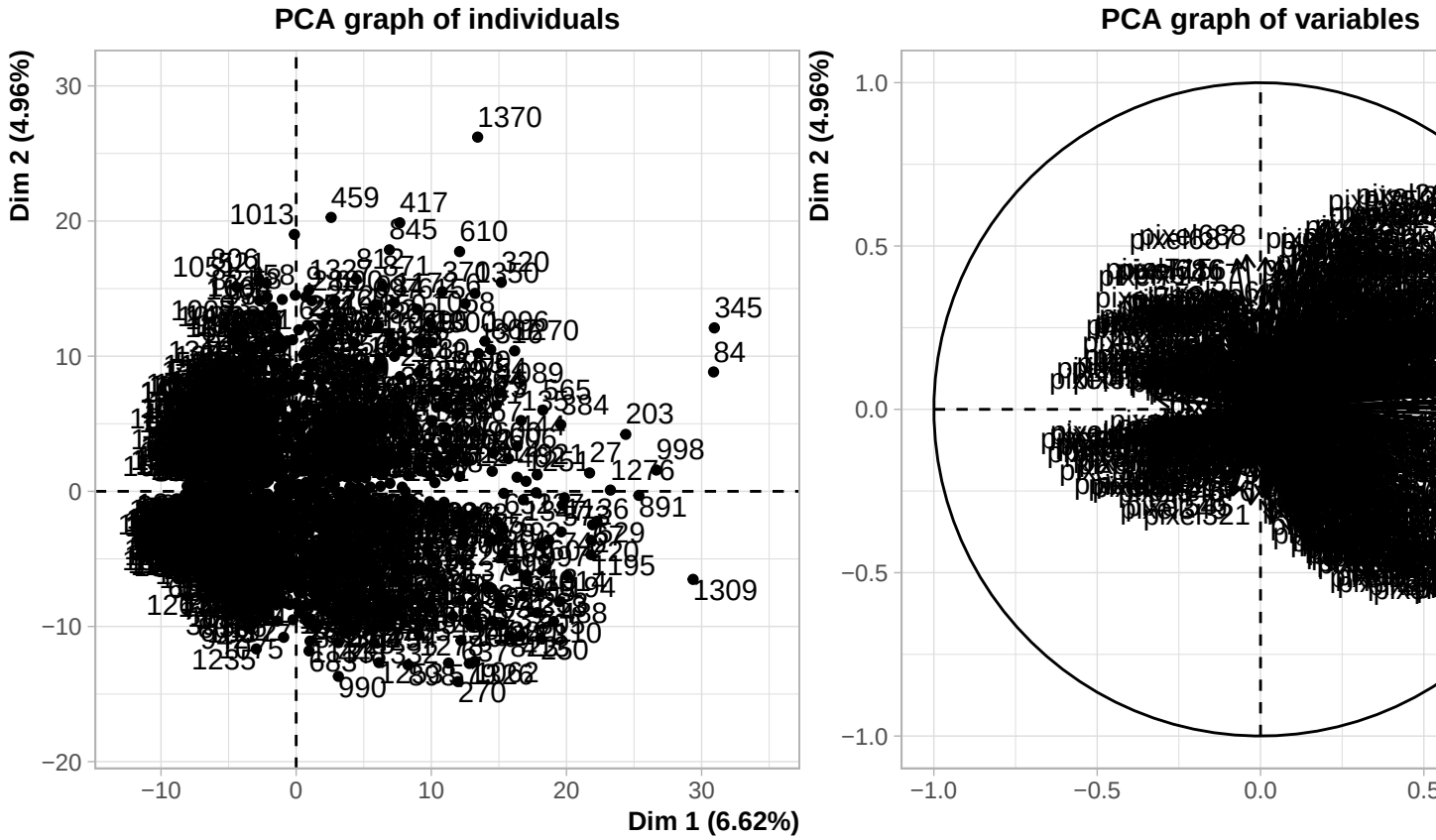
```
## v ggplot2 3.3.6     v purrr   0.3.5
## v tibble  3.1.8     v stringr 1.4.1
## v tidyr   1.2.1     v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(ggplot2)
library(SmartEDA)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(FactoMineR)
```

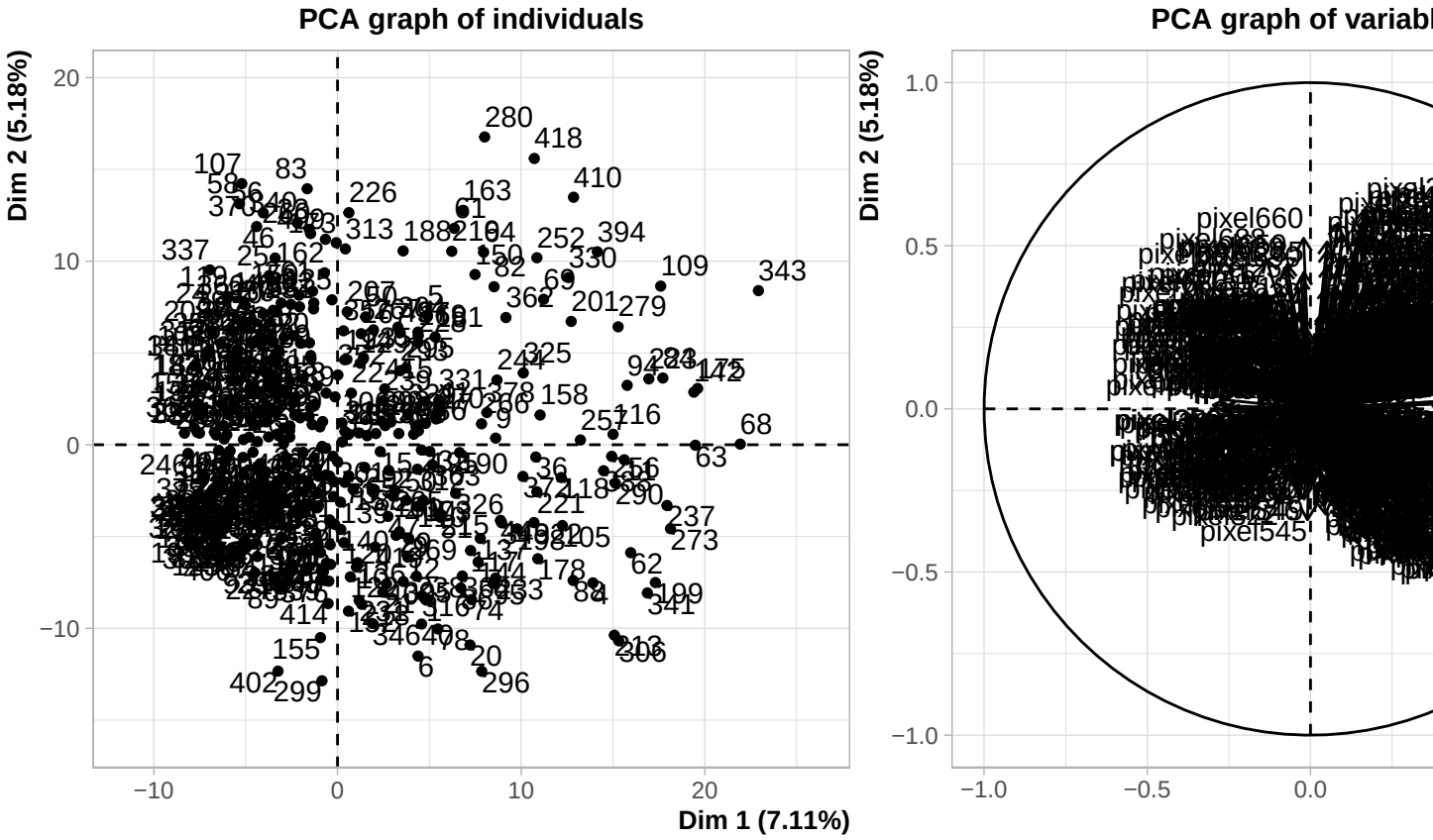
```
#PCA with entire training data set
train_pca = PCA(digit_train[,-1])
```




```
#Let's reduce the dataframe further to see if we can find more helpful data
set.seed(330)
percent <- 0.3
train_split <- sample(nrow(digit_train), nrow(digit_train)*percent)
digit_train_df <- digit_train[train_split,]
dim(digit_train_df)
```

```
## [1] 420 785
```

```
#PCA with new training data set
train_df_pca = PCA(digit_train_df[,-1])
```



Experimental Design

```
#feature selection/dim reduction
#feature selection

#create k-folds
#number of observations
N <- nrow(digit_train_df)
#number of desired splits
kfolds <- 3
#generate indices of holdout observations
holdout <- split(sample(1:N), 1:kfolds)
head(holdout)
```

```
## $`1`
##   [1]  70 157 194  26 317  10 244 348 249 109 205 135 311  91  74   6 394 100
##  [19] 121 310  45 176  17 224 248 413 211  75 179  52  27 127 125 138 349 393
##  [37]  50 146 230  36  29 133 107 412 257 229  57  85  80  98 378  59  65  23
##  [55] 204 255 379  48 329 182 234 417 210 245  79 119 324  82 314  58 233 323
##  [73] 174 242 185 371 410 104   1 243  86 148 318 342 263 363  19 246 212 110
##  [91] 361  34 169 392 297  40 299 106 188 227 331 159 365 102  33 238 396   3
## [109] 301 160 338 184 213 189  56 333 400 252 261 322 356 180 222 373 416 134
## [127] 241 362  76  20 199 296 155 271 231 395 149 147 203  47
##
## $`2`
##   [1] 267 285  54 387  16  62 368 381 228 308 420 153 111  81 313 101 366 278
##  [19] 335 357 209 126 202 164 113 130 137 328 385 408 302 276  18 399  94 131
```

```
##  [37] 359 195 168 240 304 187 272 275 384  93   2  28 165 279 305 122 143 419
##  [55] 112 300 150 326 390 386  14  71 332 346 129  95 235 383 312 367 353 258
##  [73] 141  55 128  22 316 190 264 124 268 374  31 351  37 330 307 315 283 375
##  [91] 183 239  38  43 377   9 406 254  44 158 334   5  15 167 136  99  72 200
## [109] 181 339 265 217 260   8  63 411 380 117 398 414  90 197 178  30 156 142
## [127]  87 289 191 355  73  84 259 208  24  69 407 360 389  11
##
## $`3`
##   [1] 214 218 139 347 345 166 118 321 288 220  41 114 284 319  49 409 247 172
##  [19] 340 269 237 369 354  96 382 266 161  53 256 216 350  97 273  21 337 402
##  [37] 370  66  42 103 270  77 281  92 327  68  89 343 225 219 277  83 250 162
##  [55] 320 154 201 292 132 352 193 294 415  88 376 175 223 163  51 198  39 177
##  [73]  13 344 221 372 170 192 306 151  46 173 232 388 120 140 108 262 391 309
##  [91] 364 290 405 282  61 186 418  35 274 253 303 105  32 404 336 116 298 401
## [109]  64  67 115 152 123 325 286 145   4 341 206 403 226 236 358  25  12 144
## [127]  78 291 196 280   7 397 171 293  60 215 287 295 251 207
```

```
#first build training and testing data sets for each fold
AllResults <- list()
AllLabels <- list()
for (k in 1:kfolds){
  digitDF_test <- digit_train_df[holdout[[k]],]
  digitDF_train =digit_train_df[-holdout[[k]],]
}

head(digitDF_test)
```

Cross-validation

```
## # A tibble: 6 x 785
##   label pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9
##   <fct>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 5          0      0      0      0      0      0      0      0      0      0
## 2 3          0      0      0      0      0      0      0      0      0      0
## 3 2          0      0      0      0      0      0      0      0      0      0
## 4 5          0      0      0      0      0      0      0      0      0      0
## 5 1          0      0      0      0      0      0      0      0      0      0
## 6 2          0      0      0      0      0      0      0      0      0      0
## # ... with 774 more variables: pixel10 <dbl>, pixel11 <dbl>, pixel12 <dbl>,
## #   pixel13 <dbl>, pixel14 <dbl>, pixel15 <dbl>, pixel16 <dbl>, pixel17 <dbl>,
## #   pixel18 <dbl>, pixel19 <dbl>, pixel20 <dbl>, pixel21 <dbl>, pixel22 <dbl>,
## #   pixel23 <dbl>, pixel24 <dbl>, pixel25 <dbl>, pixel26 <dbl>, pixel27 <dbl>,
## #   pixel28 <dbl>, pixel29 <dbl>, pixel30 <dbl>, pixel31 <dbl>, pixel32 <dbl>,
## #   pixel33 <dbl>, pixel34 <dbl>, pixel35 <dbl>, pixel36 <dbl>, pixel37 <dbl>,
## #   pixel38 <dbl>, pixel39 <dbl>, pixel40 <dbl>, pixel41 <dbl>, ...
```

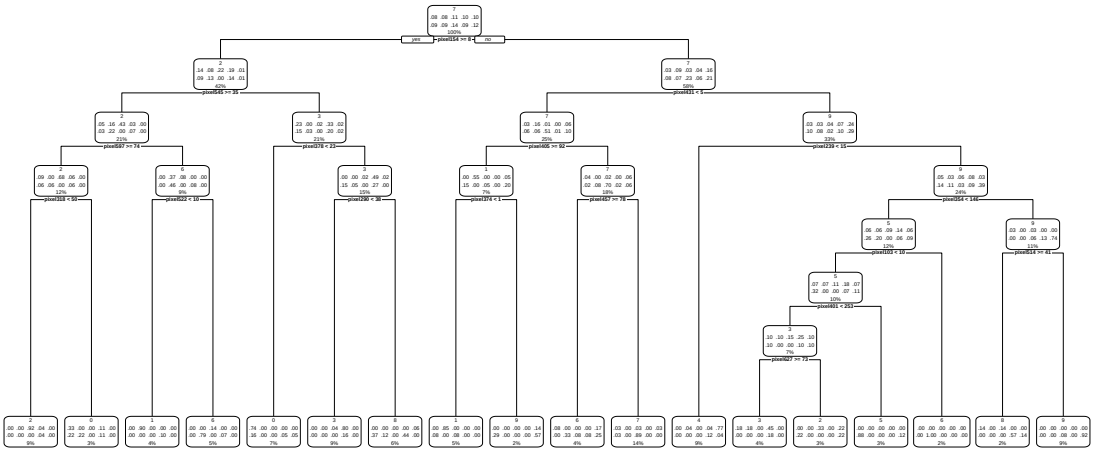
```
digitDF_test_noLabel <- digitDF_test[-c(1)]
digitDF_test_justLabel <- digitDF_test[c(1)]

digitDF_test$label<- as.factor(digitDF_test$label)
```

Section 2: Decision Tree

Using features of the data to make a tree-like structure, with nodes that represent decisions and "leaves" that represent predictions of what the handwritten digit is based on the decision made off of the features in a node.

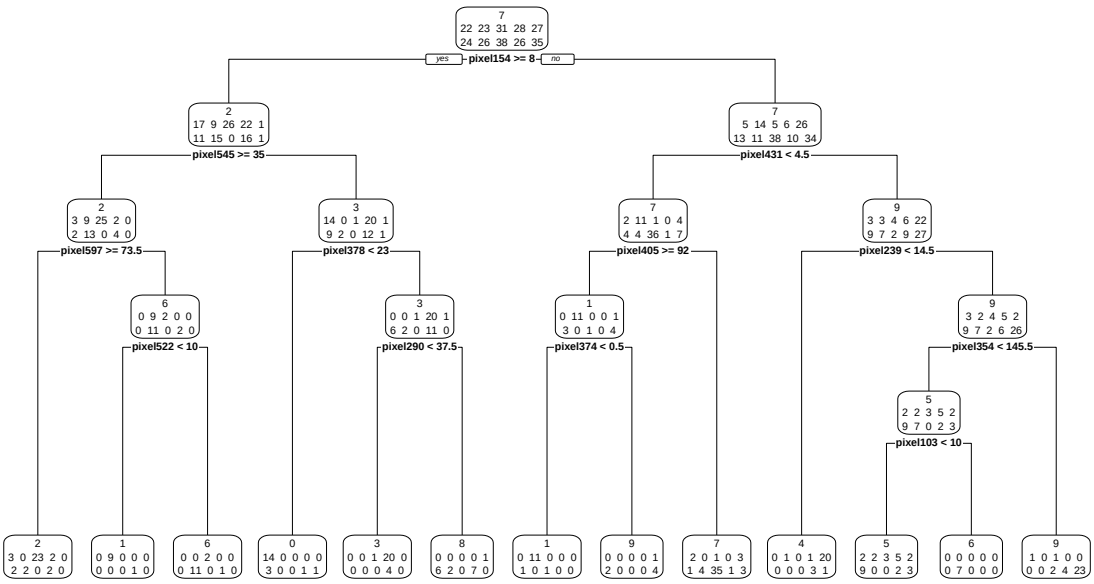
```
#build initial decision tree
library(rpart)
library(rpart.plot)
train_tree <- rpart(label ~., digitDF_train, method = "class")
rpart.plot(train_tree)
```

```
## Warning: All boxes will be white (the box.palette argument will be ignored) because
## the number of classes in the response 10 is greater than length(box.palette) 6.
## To silence this warning use box.palette=0 or trace=-1.
```



```
#identify best cp value to use
best_cp <- train_tree$cptable[which.min(train_tree$cptable[,"xerror"]), "CP"]
#prune intial tree based on best cp value
train_tree2 <- prune(train_tree, cp=best_cp)
#plot
rpart.plot(train_tree2, faclen = 0, extra = 1, roundint = F, digits = 5)
```

```
## Warning: All boxes will be white (the box.palette argument will be ignored) because
## the number of classes in the response 10 is greater than length(box.palette) 6.
## To silence this warning use box.palette=0 or trace=-1.
```




```
#prediction time
predicted_digit <- predict(train_tree2, newdata = digitDF_test, type="class")
predicted_digit
```

```
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
##   4   3   2   0   4   2   0   8   7   1   6   4   0   8   6   4   5   1   9   2
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
##   0   9   2   7   2   8   3   4   0   1   9   6   2   9   7   4   7   2   8   1
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
##   7   1   7   4   1   7   5   2   1   6   7   9   2   9   7   6   7   0   7   3
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
##   1   4   7   7   6   7   7   3   2   5   6   7   5   4   2   6   1   1   2   7
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
##   7   5   0   2   3   3   7   4   7   5   5   0   5   1   7   4   4   9   4   2
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
##   1   7   0   5   0   7   3   9   0   7   4   9   8   3   1   8   2   2   9   8
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
##   9   8   7   1   2   2   2   3   4   3   1   8   4   4   4   0   9   4   9   9
## Levels: 0 1 2 3 4 5 6 7 8 9
```

```
library(caret)
```

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

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

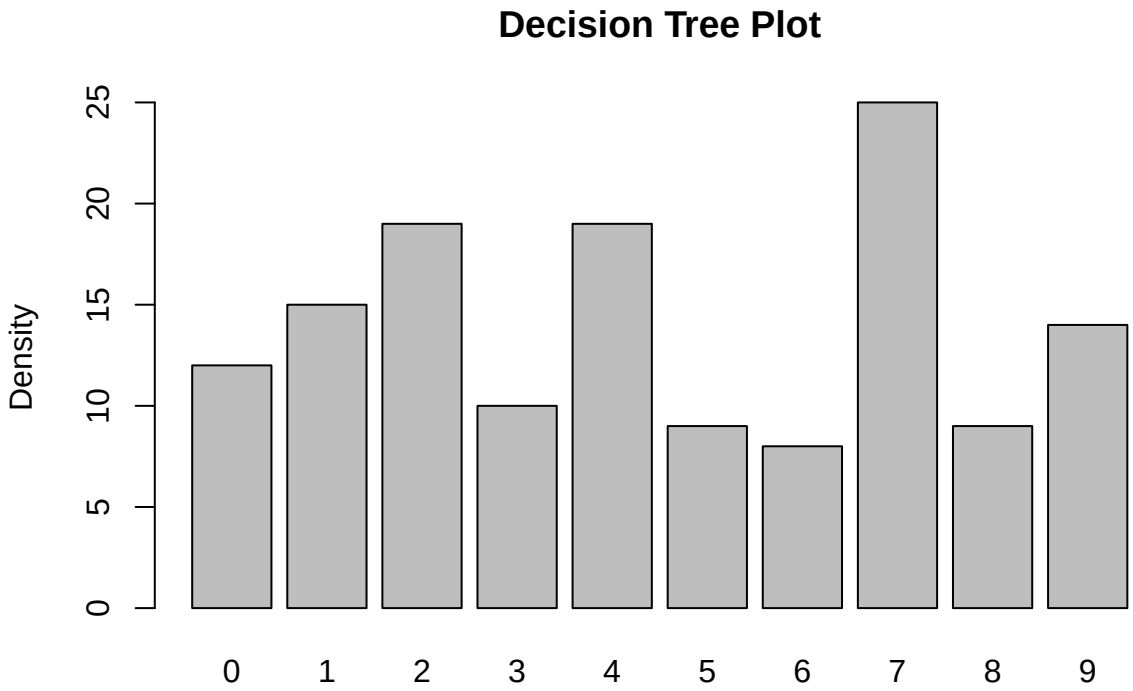
```
confusionMatrix(digitDF_test$label,predicted_digit)
```

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

```
##          1  0 12  0  0  2  1  0  3  0  0
##          2  0  2 10  1  0  0  4  1  0  0
##          3  1  1  2  5  1  2  0  0  0  0
##          4  0  0  0  0  9  0  0  1  0  2
##          5  2  0  1  1  3  2  0  0  5  0
##          6  0  0  1  1  4  1  4  2  0  0
##          7  0  0  0  0  0  0  0 11  0  0
##          8  0  0  1  2  0  1  0  1  3  3
##          9  1  0  0  0  0  1  0  1  1  8
##
## Overall Statistics
##
##                Accuracy : 0.5143
##                  95% CI : (0.4284, 0.5996)
##     No Information Rate : 0.1786
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4603
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.66667  0.80000  0.52632  0.50000  0.47368  0.22222
## Specificity           0.91406  0.95200  0.93388  0.94615  0.97521  0.90840
## Pos Pred Value        0.42105  0.66667  0.55556  0.41667  0.75000  0.14286
## Neg Pred Value        0.96694  0.97541  0.92623  0.96094  0.92188  0.94444
## Prevalence            0.08571  0.10714  0.13571  0.07143  0.13571  0.06429
## Detection Rate        0.05714  0.08571  0.07143  0.03571  0.06429  0.01429
## Detection Prevalence  0.13571  0.12857  0.12857  0.08571  0.08571  0.10000
## Balanced Accuracy     0.79036  0.87600  0.73010  0.72308  0.72445  0.56531
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.50000  0.44000  0.33333  0.57143
## Specificity           0.93182  1.00000  0.93893  0.96825
## Pos Pred Value        0.30769  1.00000  0.27273  0.66667
## Neg Pred Value        0.96850  0.89147  0.95349  0.95312
## Prevalence            0.05714  0.17857  0.06429  0.10000
## Detection Rate        0.02857  0.07857  0.02143  0.05714
## Detection Prevalence  0.09286  0.07857  0.07857  0.08571
## Balanced Accuracy     0.71591  0.72000  0.63613  0.76984
```

```
#accumulate results from each fold
AllResults <- c(AllResults, predicted_digit)
AllLabels <- c(AllLabels, predicted_digit)
```

```
plot(predicted_digit,ylab="Density", main="Decision Tree Plot")
```




```
# end CV - present results for all folds
table(unlist(AllResults), unlist(AllLabels))
```

```
##
##       1  2  3  4  5  6  7  8  9 10
##   1  12  0  0  0  0  0  0  0  0  0
##   2   0 15  0  0  0  0  0  0  0  0
##   3   0  0 19  0  0  0  0  0  0  0
##   4   0  0  0 10  0  0  0  0  0  0
##   5   0  0  0  0 19  0  0  0  0  0
##   6   0  0  0  0  0  9  0  0  0  0
##   7   0  0  0  0  0  0  8  0  0  0
##   8   0  0  0  0  0  0  0 25  0  0
##   9   0  0  0  0  0  0  0  0  9  0
##   10  0  0  0  0  0  0  0  0  0 14
```

Section 3: Naive Bayes

Similarly to the Decision Tree model, the Naive Bayes model will attempt to predict what a handwritten digit is. However, this model uses probabilities of one event happening given some circumstance, rather than using a flow-chart like method. For example, what is the likelihood of a digit being "7" given that pixel value of pixel #330 is 92?

```
library(e1071)
#training model
NB_training <- naiveBayes(label ~ ., data=digitDF_train, na.action = na.pass)
summary(NB_training)
```

```
##           Length Class  Mode
## apriori    10    table  numeric
## tables    784    -none- list
## levels     10    -none- character
## isnumeric 784    -none- logical
```

```
## call        4    -none- call
```

```
#NB model prediction
NB_prediction <- predict(NB_training, digitDF_test_noLabel)
NB_prediction
```

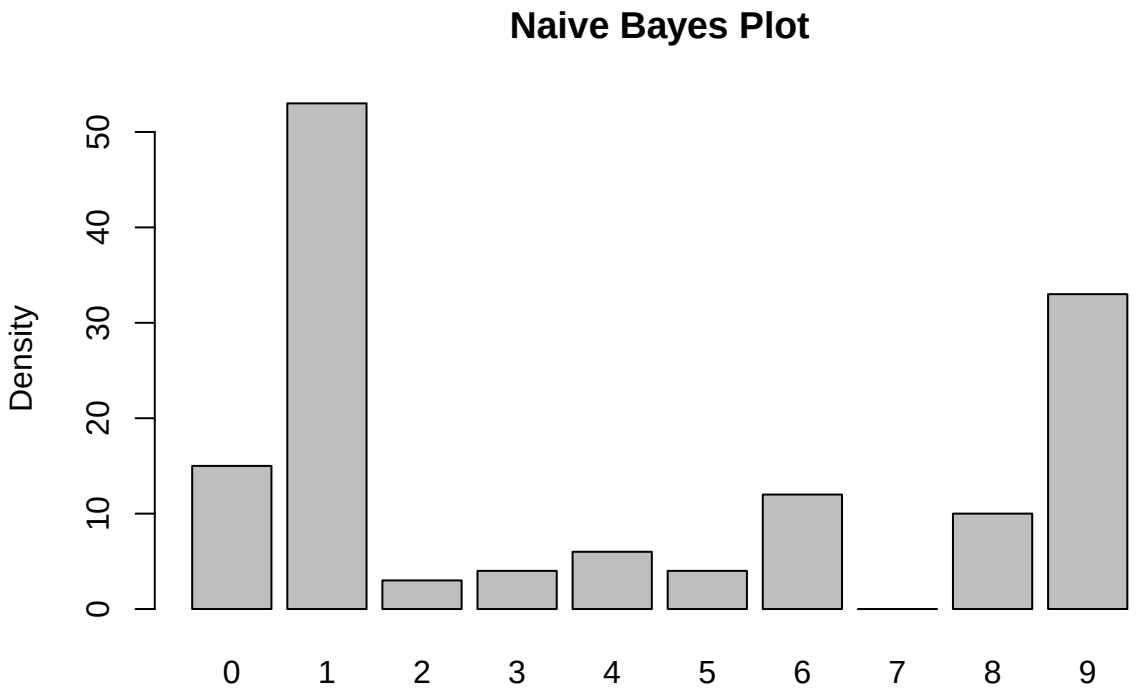
```
##   [1] 1 8 8 5 1 1 0 1 1 1 8 9 8 9 6 9 1 1 9 1 0 9 1 1 1 8 1 9 0 1 9 1 0 8 9 1 9
##  [38] 1 9 6 6 1 9 9 1 0 1 0 1 2 1 4 1 9 9 6 6 1 1 8 1 1 1 0 1 0 0 2 8 1 1 1 9 9
##  [75] 1 1 1 1 5 9 9 4 6 0 1 3 9 1 9 3 8 0 2 1 1 1 4 9 1 1 1 1 9 1 9 0 3 9 0 9 4
## [112] 1 1 5 1 3 0 0 9 1 9 9 9 5 6 1 1 6 9 6 1 9 9 6 4 6 9 6 4 8
## Levels: 0 1 2 3 4 5 6 7 8 9
```

```
#confusion matrix to see how accurate the model is
confusionMatrix(NB_prediction,digitDF_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 13  0  0  0  0  1  0  0  0  1
##          1  2 18 11  2  3  4  5  2  6  0
##          2  0  0  1  1  0  0  0  0  1  0
##          3  0  0  0  3  0  1  0  0  0  0
##          4  0  0  0  0  4  0  0  0  1  1
##          5  0  0  1  2  0  1  0  0  0  0
##          6  1  0  2  0  0  2  7  0  0  0
##          7  0  0  0  0  0  0  0  0  0  0
##          8  3  0  2  2  0  1  0  0  2  0
##          9  0  0  1  2  5  4  1  9  1 10
##
## Overall Statistics
##
##                Accuracy : 0.4214
##                  95% CI : (0.3385, 0.5077)
##     No Information Rate : 0.1357
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3509
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.68421   1.0000 0.055556  0.25000  0.33333 0.071429
## Specificity           0.98347   0.7131 0.983607  0.99219  0.98438 0.976190
## Pos Pred Value        0.86667   0.3396 0.333333  0.75000  0.66667 0.250000
## Neg Pred Value        0.95200   1.0000 0.875912  0.93382  0.94030 0.904412
## Prevalence            0.13571   0.1286 0.128571  0.08571  0.08571 0.100000
## Detection Rate        0.09286   0.1286 0.007143  0.02143  0.02857 0.007143
## Detection Prevalence  0.10714   0.3786 0.021429  0.02857  0.04286 0.028571
## Balanced Accuracy     0.83384   0.8566 0.519581  0.62109  0.65885 0.523810
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.53846  0.00000  0.18182  0.83333
## Specificity           0.96063  1.00000  0.93798  0.82031
## Pos Pred Value        0.58333      NaN  0.20000  0.30303
```

```
## Neg Pred Value         0.95312  0.92143  0.93077  0.98131
## Prevalence             0.09286  0.07857  0.07857  0.08571
## Detection Rate         0.05000  0.00000  0.01429  0.07143
## Detection Prevalence   0.08571  0.00000  0.07143  0.23571
## Balanced Accuracy      0.74955  0.50000  0.55990  0.82682
```

```
plot(NB_prediction, ylab= "Density", main = "Naive Bayes Plot")
```



Section 4: Decision Tree and Naive Bayes Algorithm Performance Comparison

Looking at the results of each model above, both algorithms were able to make some pretty good predictions on the handwritten digits. Using a confusion matrix that looks at True/False positives and True/False Negatives, the summary of the statistics of each models are:

Decision Tree:

- p-value: <2.2e-16, which is statistically significant assuming an alpha level of 0.05
- Accuracy: 0.5714 or 57%
- 95% confidence interval: (0.4851, 0.6542). Meaning that the accuracy of the model is between 48% and 65%

Naive Bayes:

- p-value: <2.2e-16, which is also statistically significant assuming an alpha level of 0.05
- Accuracy: 0.4214 or 42%
- 95% confidence interval (0.3385, 0.5077). Meaning that the accuracy of the model is between 34% and 51%.

In addition to the differences in accuracy of each model, it's also important to note that each model ran at different lengths of time. More specifically, the Naive Bayes algorithm ran slightly faster than the Decision Tree. The difference in speed of the models can likely be explained by the different computational methods of the algorithms. Seeing that Naive Bayes is a statistical model that uses a probability equation to make

predictions, it makes sense that a computer takes less time to plug in numbers and do the math rather than having to make decisions based on features.

Based on the results of this analysis, the Decision Tree model was more accurate at recognizing the handwritten digits. Although this may the case in this specific task, it is not fair to say that one algorithm is better than the other. Each algorithm has pros and cons of their own, and may or may not be more appropriate for achieving goals. Ultimately, choosing the appropriate algorithm will depend on the goal of the task and the data being used.

SVMs, kNN, and Random Forest for Handwriting Recognition

In addition to Decision Tree and Naive Bayes models, Support Vector Machines (SVM), k Nearest Neighbors (kNN), and Random Forests can be used to make predictions. Using the same handwritten digit dataset that was pre-processed above, the following sections will use SVM, kNN, and Random Forest algorithms for handwriting recognition and those predictions will be compared with the Decision Tree and Naive Bayes results above.

Section 5: Support Vector Machine

Support Vector Machines (SVMs) are another type of machine learning algorithm used to make classifications. SVMs are highly flexible, and make predictions based on probabilities. In this section, the digit data will be used to train and test an SVM to see how well it can recognize handwritten data.

```
#make the sampling predictable
set.seed(333)
#randomly sample training dataset
SVMdigit_list <- createDataPartition(y=digit_train$label, p=.6, list = FALSE)
#create train and test sets
SVMtrain <- digit_train[SVMdigit_list,]
SVMtest <- digit_train[-SVMdigit_list,]
SVMtrain$label <- as.factor(SVMtrain$label)
SVMtest$label <- as.factor(SVMtest$label)
```

```
#create SVM model
SVM_model <- train(label ~ ., data=SVMtrain, method = "svmRadial", trControl=trainControl(method="none")
```

```
## Warning in preProcess.default(method = c("center", "scale"), x =
## structure(c(0, : These variables have zero variances: pixel0, pixel1, pixel2,
## pixel3, pixel4, pixel5, pixel6, pixel7, pixel8, pixel9, pixel10, pixel11,
## pixel12, pixel13, pixel14, pixel15, pixel16, pixel17, pixel18, pixel19, pixel20,
## pixel21, pixel22, pixel23, pixel24, pixel25, pixel26, pixel27, pixel28, pixel29,
## pixel30, pixel31, pixel32, pixel33, pixel34, pixel35, pixel36, pixel46, pixel47,
## pixel48, pixel49, pixel50, pixel51, pixel52, pixel53, pixel54, pixel55, pixel56,
## pixel57, pixel58, pixel59, pixel60, pixel61, pixel81, pixel82, pixel83, pixel84,
## pixel85, pixel86, pixel87, pixel88, pixel109, pixel110, pixel111, pixel112,
## pixel113, pixel114, pixel115, pixel138, pixel139, pixel140, pixel141, pixel142,
## pixel143, pixel144, pixel166, pixel167, pixel168, pixel169, pixel170, pixel195,
## pixel196, pixel197, pixel198, pixel223, pixel224, pixel226, pixel251, pixel252,
## pixel253, pixel279, pixel280, pixel281, pixel307, pixel308, pixel309, pixel335,
## pixel336, pixel337, pixel362, pixel363, pixel364, pixel365, pixel366, pixel390,
## pixel391, pixel392, pixel393, pixel394, pixel418, pixel419, pixel420, pixel421,
## pixel422, pixel448, pixel449, pixel450, pixel476, pixel477, pixel478, pixel504,
## pixel505, pixel530, pixel531, pixel532, pixel533, pixel559, pixel560, pixel561,
## pixel587, pixel588, pixel589, pixel615, pixel616, pixel617, pixel643, pixel644,
## pixel645, pixel669, pixel670, pixel671, pixel672, pixel673, pixel674, pixel697,
```

```
## pixel698, pixel699, pixel700, pixel701, pixel702, pixel703, pixel725, pixel726,
## pixel727, pixel728, pixel729, pixel730, pixel731, pixel732, pixel753, pixel754,
## pixel755, pixel756, pixel757, pixel758, pixel759, pixel760, pixel761, pixel762,
## pixel763, pixel764, pixel778, pixel779, pixel780, pixel781, pixel782, pixel783
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut
## = 10, : These variables have zero variances: pixel0, pixel1, pixel2, pixel3,
## pixel4, pixel5, pixel6, pixel7, pixel8, pixel9, pixel10, pixel11, pixel12,
## pixel13, pixel14, pixel15, pixel16, pixel17, pixel18, pixel19, pixel20, pixel21,
## pixel22, pixel23, pixel24, pixel25, pixel26, pixel27, pixel28, pixel29, pixel30,
## pixel31, pixel32, pixel33, pixel34, pixel35, pixel36, pixel46, pixel47, pixel48,
## pixel49, pixel50, pixel51, pixel52, pixel53, pixel54, pixel55, pixel56, pixel57,
## pixel58, pixel59, pixel60, pixel61, pixel81, pixel82, pixel83, pixel84, pixel85,
## pixel86, pixel87, pixel88, pixel109, pixel110, pixel111, pixel112, pixel113,
## pixel114, pixel115, pixel138, pixel139, pixel140, pixel141, pixel142, pixel143,
## pixel144, pixel166, pixel167, pixel168, pixel169, pixel170, pixel195, pixel196,
## pixel197, pixel198, pixel223, pixel224, pixel226, pixel251, pixel252, pixel253,
## pixel279, pixel280, pixel281, pixel307, pixel308, pixel309, pixel335, pixel336,
## pixel337, pixel362, pixel363, pixel364, pixel365, pixel366, pixel390, pixel391,
## pixel392, pixel393, pixel394, pixel418, pixel419, pixel420, pixel421, pixel422,
## pixel448, pixel449, pixel450, pixel476, pixel477, pixel478, pixel504, pixel505,
## pixel530, pixel531, pixel532, pixel533, pixel559, pixel560, pixel561, pixel587,
## pixel588, pixel589, pixel615, pixel616, pixel617, pixel643, pixel644, pixel645,
## pixel669, pixel670, pixel671, pixel672, pixel673, pixel674, pixel697, pixel698,
## pixel699, pixel700, pixel701, pixel702, pixel703, pixel725, pixel726, pixel727,
## pixel728, pixel729, pixel730, pixel731, pixel732, pixel753, pixel754, pixel755,
## pixel756, pixel757, pixel758, pixel759, pixel760, pixel761, pixel762, pixel763,
## pixel764, pixel778, pixel779, pixel780, pixel781, pixel782, pixel783
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
SVM_model
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 845 samples
## 784 predictors
##  10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
##
## Pre-processing: centered (784), scaled (784)
## Resampling: None
```

The model has 842 samples (number rows/observations in the SVMtrain object), and was centered on 784 predictors, which is one less than the initial data set because the first attribute (label) is the target. The model also has no resampling, meaning that there is no cross validation.

```
#time to make predictions
SVM_modelPredict <- predict(SVM_model, newdata = SVMtest)
confusionMatrix(SVM_modelPredict, SVMtest$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 49  0  0  0  0  0  0  0  0  0
##          1  0 61  1  2  0  1  2  1  3  1
```

```
##          2  3  2 47  7  6  9  9 10  3  2
##          3  0  0  2 41  0  7  0  0  4  1
##          4  0  1  0  0 41  2  0  0  0  4
##          5  0  1  0  1  0 26  1  0  6  0
##          6  2  0  0  0  0  1 45  0  1  0
##          7  0  0  0  1  0  2  0 38  0  1
##          8  1  1  2  3  1  2  1  0 37  0
##          9  1  0  0  1  4  0  0  5  0 48
##
## Overall Statistics
##
##                Accuracy : 0.7802
##                  95% CI : (0.7434, 0.814)
##     No Information Rate : 0.1189
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7556
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.87500   0.9242  0.90385  0.73214  0.78846  0.52000
## Specificity           1.00000   0.9775  0.89861  0.97194  0.98608  0.98218
## Pos Pred Value        1.00000   0.8472  0.47959  0.74545  0.85417  0.74286
## Neg Pred Value        0.98617   0.9896  0.98906  0.97000  0.97830  0.95385
## Prevalence            0.10090   0.1189  0.09369  0.10090  0.09369  0.09009
## Detection Rate        0.08829   0.1099  0.08468  0.07387  0.07387  0.04685
## Detection Prevalence  0.08829   0.1297  0.17658  0.09910  0.08649  0.06306
## Balanced Accuracy     0.93750   0.9509  0.90123  0.85204  0.88727  0.75109
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.77586  0.70370  0.68519  0.84211
## Specificity           0.99195  0.99202  0.97804  0.97791
## Pos Pred Value        0.91837  0.90476  0.77083  0.81356
## Neg Pred Value        0.97431  0.96881  0.96647  0.98185
## Prevalence            0.10450  0.09730  0.09730  0.10270
## Detection Rate        0.08108  0.06847  0.06667  0.08649
## Detection Prevalence  0.08829  0.07568  0.08649  0.10631
## Balanced Accuracy     0.88391  0.84786  0.83161  0.91001
```

This model that tested a subset of the entire dataset was 77% accurate, and with a p value of <0.05 the results are statistically significant.

Section 6: K Nearest Neighbor

The next algorithm that will be explored is k Nearest Neighbor (kNN). Unlike other machine learning algorithms, kNN models make classifications and predictions when new examples are given. The model stores the training data and does not do any calculations until it makes its predictions using classifications based on the majority-voted category label in the k-nearest training example. Like done before, this model will also use the data from above.

```
#load packages
library(e1071)
library(class)
```

```r
library(caTools)

#make the sampling predictable
set.seed(333)
#randomly sample training dataset
kNNdigit_list <- createDataPartition(y=digit_train$label, p=.6, list = FALSE)
#create train and test sets
kNNtrain <- digit_train[kNNdigit_list,]
kNNtest <- digit_train[-kNNdigit_list,]
kNNtrain$label <- as.factor(kNNtrain$label)
kNNtest$label <- as.factor(kNNtest$label)

#kNN model with standard k = 1
kNNdigit <- knn(train = kNNtrain, test = kNNtest, cl = kNNtrain$label, k=1)
confusionMatrix(kNNdigit, kNNtest$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 52  0  0  1  0  0  2  0  1  0
##          1  0 65  3  0  1  1  2  0  3  0
##          2  0  0 44  0  1  0  0  0  0  0
##          3  0  0  0 47  0  4  0  0  4  0
##          4  1  0  0  0 43  0  0  0  1  5
##          5  1  0  0  4  0 39  0  1  2  0
##          6  1  0  0  0  1  2 54  0  1  0
##          7  0  1  4  2  1  1  0 46  0  6
##          8  0  0  0  1  0  1  0  2 42  0
##          9  1  0  1  1  5  2  0  5  0 46
##
## Overall Statistics
##
##                Accuracy : 0.8613
##                  95% CI : (0.8297, 0.8889)
##     No Information Rate : 0.1189
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8457
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.92857   0.9848  0.84615  0.83929  0.82692  0.78000
## Specificity           0.99198   0.9796  0.99801  0.98397  0.98608  0.98416
## Pos Pred Value        0.92857   0.8667  0.97778  0.85455  0.86000  0.82979
## Neg Pred Value        0.99198   0.9979  0.98431  0.98200  0.98218  0.97835
## Prevalence            0.10090   0.1189  0.09369  0.10090  0.09369  0.09009
## Detection Rate        0.09369   0.1171  0.07928  0.08468  0.07748  0.07027
## Detection Prevalence  0.10090   0.1351  0.08108  0.09910  0.09009  0.08468
## Balanced Accuracy     0.96028   0.9822  0.92208  0.91163  0.90650  0.88208
##                      Class: 6 Class: 7 Class: 8 Class: 9
```

```
## Sensitivity            0.9310  0.85185  0.77778  0.80702
## Specificity            0.9899  0.97006  0.99202  0.96988
## Pos Pred Value         0.9153  0.75410  0.91304  0.75410
## Neg Pred Value         0.9919  0.98381  0.97642  0.97773
## Prevalence             0.1045  0.09730  0.09730  0.10270
## Detection Rate         0.0973  0.08288  0.07568  0.08288
## Detection Prevalence   0.1063  0.10991  0.08288  0.10991
## Balanced Accuracy      0.9605  0.91096  0.88490  0.88845
```

Right off the bat, at 88% this model is more accurate than all of the others so far. This model also has a very small p-value. Now, let's try adjusting the k value in the model to see which k value is the most accurate.

```
#kNN model with standard k = 3
kNNdigit <- knn(train = kNNtrain, test = kNNtest, cl = kNNtrain$label, k=3)
confusionMatrix(kNNdigit, kNNtest$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 55  0  0  0  0  0  1  0  0  0
##          1  0 65  3  1  3  2  2  3  4  0
##          2  0  0 44  0  0  0  0  0  0  0
##          3  0  0  2 47  0  4  0  0  3  0
##          4  0  0  0  0 43  0  0  0  2  2
##          5  0  0  0  2  0 37  0  1  2  0
##          6  1  0  0  0  2  1 55  0  1  0
##          7  0  1  3  1  1  1  0 45  0  4
##          8  0  0  0  3  0  2  0  0 41  0
##          9  0  0  0  2  3  3  0  5  1 51
##
## Overall Statistics
##
##                Accuracy : 0.8703
##                  95% CI : (0.8394, 0.8971)
##     No Information Rate : 0.1189
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8556
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9821   0.9848  0.84615  0.83929  0.82692  0.74000
## Specificity            0.9980   0.9632  1.00000  0.98196  0.99205  0.99010
## Pos Pred Value         0.9821   0.7831  1.00000  0.83929  0.91489  0.88095
## Neg Pred Value         0.9980   0.9979  0.98434  0.98196  0.98228  0.97466
## Prevalence             0.1009   0.1189  0.09369  0.10090  0.09369  0.09009
## Detection Rate         0.0991   0.1171  0.07928  0.08468  0.07748  0.06667
## Detection Prevalence   0.1009   0.1495  0.07928  0.10090  0.08468  0.07568
## Balanced Accuracy      0.9901   0.9740  0.92308  0.91062  0.90949  0.86505
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9483  0.83333  0.75926  0.89474
```

```
## Specificity            0.9899  0.97804  0.99002  0.97189
## Pos Pred Value         0.9167  0.80357  0.89130  0.78462
## Neg Pred Value         0.9939  0.98196  0.97446  0.98776
## Prevalence             0.1045  0.09730  0.09730  0.10270
## Detection Rate         0.0991  0.08108  0.07387  0.09189
## Detection Prevalence   0.1081  0.10090  0.08288  0.11712
## Balanced Accuracy      0.9691  0.90569  0.87464  0.93331
```

At 87%, k = 3 is less accurate than k = 1.

```
#kNN model with standard k = 5
kNNdigit <- knn(train = kNNtrain, test = kNNtest, cl = kNNtrain$label, k=5)
confusionMatrix(kNNdigit, kNNtest$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 54  0  0  0  0  0  1  0  1  0
##          1  0 65  6  0  4  1  3  3  5  0
##          2  0  0 38  0  0  0  0  0  0  0
##          3  0  0  2 48  0  2  0  0  3  0
##          4  0  0  1  0 41  1  0  0  0  2
##          5  0  0  0  3  0 40  0  0  2  0
##          6  2  0  0  0  1  2 54  0  1  0
##          7  0  1  4  1  0  1  0 46  0  4
##          8  0  0  1  2  0  2  0  0 41  0
##          9  0  0  0  2  6  1  0  5  1 51
##
## Overall Statistics
##
##                Accuracy : 0.8613
##                  95% CI : (0.8297, 0.8889)
##     No Information Rate : 0.1189
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8456
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9643   0.9848  0.73077  0.85714  0.78846  0.80000
## Specificity            0.9960   0.9550  1.00000  0.98597  0.99205  0.99010
## Pos Pred Value         0.9643   0.7471  1.00000  0.87273  0.91111  0.88889
## Neg Pred Value         0.9960   0.9979  0.97292  0.98400  0.97843  0.98039
## Prevalence             0.1009   0.1189  0.09369  0.10090  0.09369  0.09009
## Detection Rate         0.0973   0.1171  0.06847  0.08649  0.07387  0.07207
## Detection Prevalence   0.1009   0.1568  0.06847  0.09910  0.08108  0.08108
## Balanced Accuracy      0.9801   0.9699  0.86538  0.92156  0.89025  0.89505
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9310  0.85185  0.75926  0.89474
## Specificity            0.9879  0.97804  0.99002  0.96988
## Pos Pred Value         0.9000  0.80702  0.89130  0.77273
```

```
## Neg Pred Value         0.9919  0.98394  0.97446  0.98773
## Prevalence             0.1045  0.09730  0.09730  0.10270
## Detection Rate         0.0973  0.08288  0.07387  0.09189
## Detection Prevalence   0.1081  0.10270  0.08288  0.11892
## Balanced Accuracy      0.9595  0.91495  0.87464  0.93231
```

At k = 5, the model also achieves an accuracy rate of 87%. However, when increasing the decimal points, it is slightly lower than when k = 3.

```
#kNN model with standard k = 2
kNNdigit <- knn(train = kNNtrain, test = kNNtest, cl = kNNtrain$label, k=2)
confusionMatrix(kNNdigit, kNNtest$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 54  0  0  0  0  0  2  0  1  0
##          1  0 64  2  1  3  2  2  0  4  0
##          2  0  0 46  2  1  0  0  0  0  0
##          3  0  0  0 46  2  6  0  0  4  0
##          4  0  0  0  0 39  0  0  0  1  2
##          5  0  0  0  3  0 36  0  1  2  0
##          6  1  0  0  0  2  2 54  0  2  0
##          7  0  2  4  1  0  1  0 45  1  5
##          8  0  0  0  1  0  1  0  2 38  0
##          9  1  0  0  2  5  2  0  6  1 50
##
## Overall Statistics
##
##                Accuracy : 0.8505
##                  95% CI : (0.818, 0.8791)
##     No Information Rate : 0.1189
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8336
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9643   0.9697  0.88462  0.82143  0.75000  0.72000
## Specificity            0.9940   0.9714  0.99404  0.97595  0.99404  0.98812
## Pos Pred Value         0.9474   0.8205  0.93878  0.79310  0.92857  0.85714
## Neg Pred Value         0.9960   0.9958  0.98814  0.97988  0.97466  0.97271
## Prevalence             0.1009   0.1189  0.09369  0.10090  0.09369  0.09009
## Detection Rate         0.0973   0.1153  0.08288  0.08288  0.07027  0.06486
## Detection Prevalence   0.1027   0.1405  0.08829  0.10450  0.07568  0.07568
## Balanced Accuracy      0.9791   0.9705  0.93933  0.89869  0.87202  0.85406
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9310  0.83333  0.70370  0.87719
## Specificity            0.9859  0.97206  0.99202  0.96586
## Pos Pred Value         0.8852  0.76271  0.90476  0.74627
## Neg Pred Value         0.9919  0.98185  0.96881  0.98566
```

```
## Prevalence             0.1045  0.09730  0.09730  0.10270
## Detection Rate         0.0973  0.08108  0.06847  0.09009
## Detection Prevalence   0.1099  0.10631  0.07568  0.12072
## Balanced Accuracy      0.9585  0.90269  0.84786  0.92153
```

At 86%, k = 2 gives the lowest accuracy level so far. As such, it seems as though k = 1 gives us the best results.

Section 7: Random Forest

Finally, the last algorithm that will be tested in this report is the Random Forest model. Random Forests work by creating decision trees from subsets of data, then creating a "forest" of those trees in order to make a decision based on majority vote. Again, the same data from above will be used in this algorithm.

```
#load packages
library(randomForest)
```

```
## randomForest 4.7-1.1
```

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

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

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

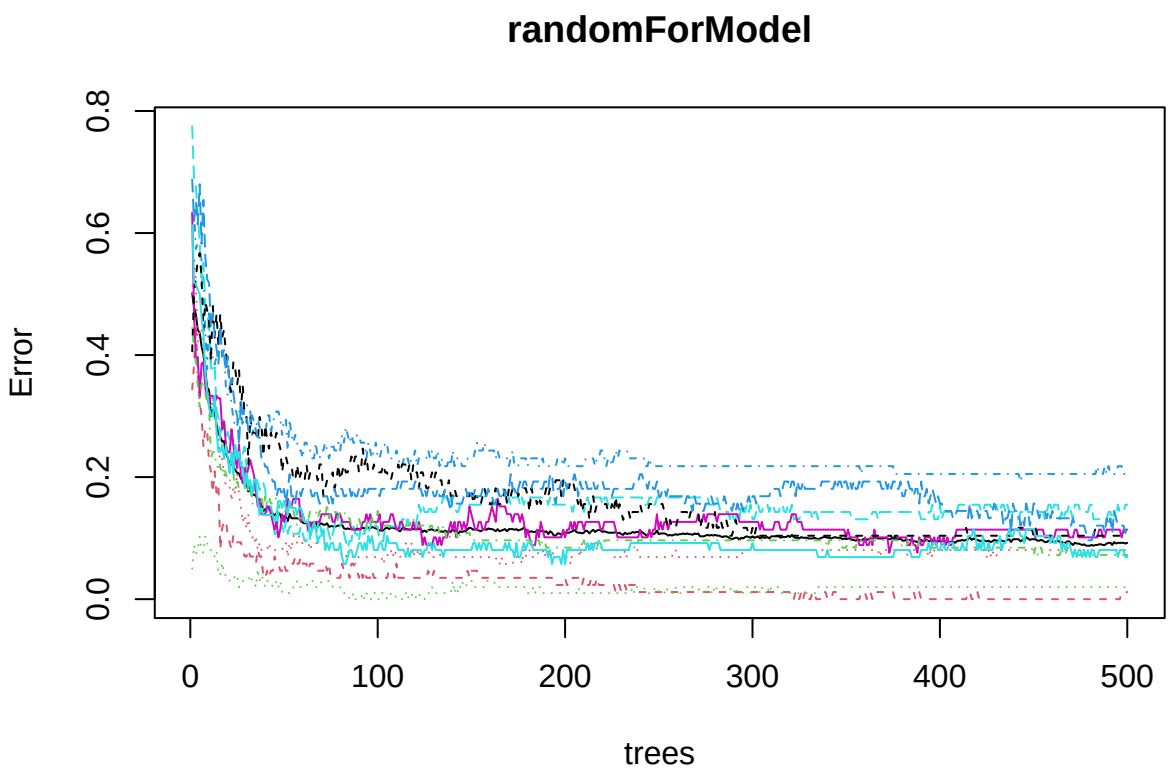
```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
#make the sampling predictable
set.seed(333)
#randomly sample training dataset
RanFdigit_list <- createDataPartition(y=digit_train$label, p=.6, list = FALSE)
#create train and test sets
RanFtrain <- digit_train[RanFdigit_list,]
RanFtest <- digit_train[-RanFdigit_list,]
RanFtrain$label <- as.factor(RanFtrain$label)
RanFtest$label <- as.factor(RanFtest$label)

#create random forest model
randomForModel <- randomForest(formula = label ~., data=RanFtrain)
randomForModel
```

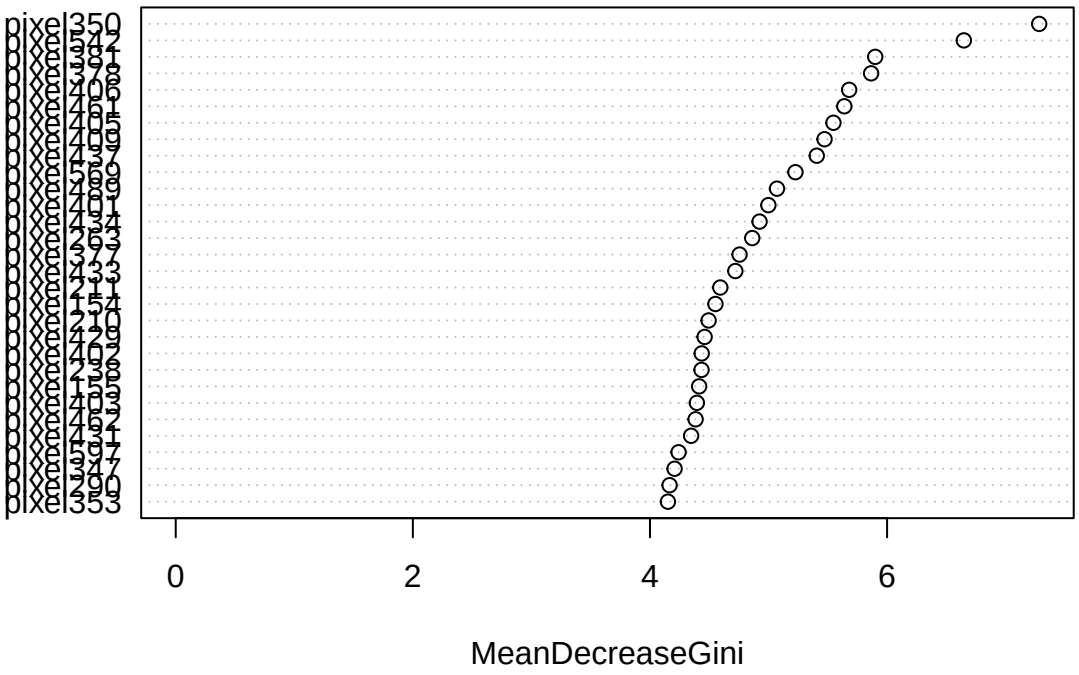
```
##
## Call:
##  randomForest(formula = label ~ ., data = RanFtrain)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 28
##
##         OOB estimate of  error rate: 9.23%
## Confusion matrix:
##    0  1  2  3  4  5  6  7  8  9 class.error
## 0 85  0  0  0  0  1  0  0  0  0  0.01162791
## 1  0 99  2  0  0  0  0  0  0  0  0.01980198
```

```
## 2  0  1 62  2  2  0  5  1  5  0  0.20512821
## 3  1  0  2 71  0  4  1  0  5  0  0.15476190
## 4  0  0  0  0 70  0  1  0  0  8  0.11392405
## 5  1  1  0  4  2 69  0  0  0  0  0.10389610
## 6  2  2  1  0  1  1 80  0  0  0  0.08045977
## 7  0  3  0  0  0  0  0 77  1  2  0.07228916
## 8  0  1  0  3  0  1  1  1 73  3  0.12048193
## 9  1  0  0  0  3  0  0  2  0 81  0.06896552
```

```
#visuals
plot(randomForModel)
```




```
#visualize importance of each predictor
varImpPlot(randomForModel)
```

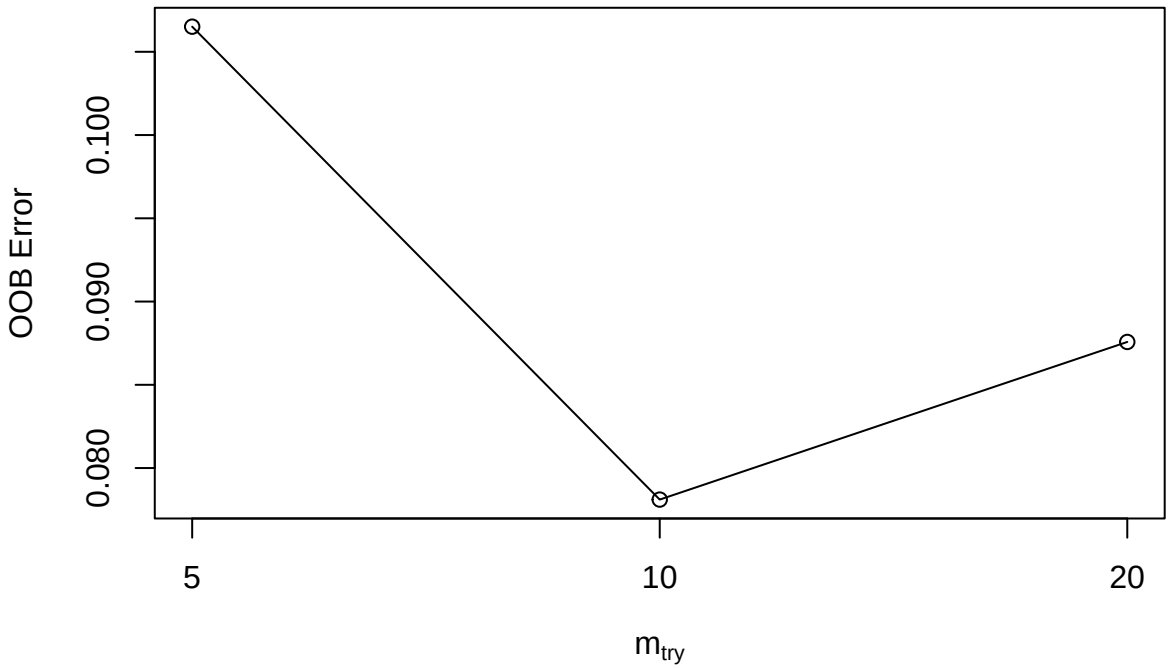


Although hard to read, it can be seen that most attributes have the same importance on the prediction. However, there are a couple that clearly stand out from the rest.

Like with the decision tree model, the random forest should be pruned so that it can be as accurate as possible.

```
#tuning
tuneRF(x=RanFtrain[,-1], y=RanFtrain$label, ntreeTry = 500, mtryStart = 10, stepFactor = 2, improve = 0
```

```
## -0.3636364 0.01
## -0.1212121 0.01
```




```
##        mtry   OOBError
## 5.OOB     5 0.10650888
## 10.OOB   10 0.07810651
## 20.OOB   20 0.08757396
```

With these parameters, the lowest out-of-bag (OOB) estimated errors was achieved when 10 randomly chosen predictors at each split when making the trees. The OOB is really small at 10, so we'll use this to make predictions.

```
#create new forest with tuned parameters
tunedRF <- randomForest(formula = label ~., data=RanFtrain, ntree=500, mtry=10)
#predictions using tuned forest
RFpredict <- predict(tunedRF, newdata = RanFtest)
confusionMatrix(RFpredict, RanFtest$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 53  0  0  0  0  0  1  0  0  0
##          1  0 65  1  1  0  1  2  1  3  1
##          2  0  0 45  0  0  0  0  0  0  0
##          3  0  0  1 49  0  2  0  0  3  1
##          4  0  0  0  1 45  1  1  1  1  4
##          5  0  0  0  1  0 44  0  0  0  0
##          6  2  0  0  1  2  1 54  0  0  0
##          7  0  1  1  1  0  0  0 47  0  1
##          8  1  0  3  1  1  0  0  1 45  0
##          9  0  0  1  1  4  1  0  4  2 50
##
## Overall Statistics
##
##                Accuracy : 0.8955
##                  95% CI : (0.867, 0.9197)
```

```
##     No Information Rate : 0.1189
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8837
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9464   0.9848  0.86538  0.87500  0.86538  0.88000
## Specificity            0.9980   0.9796  1.00000  0.98597  0.98211  0.99802
## Pos Pred Value         0.9815   0.8667  1.00000  0.87500  0.83333  0.97778
## Neg Pred Value         0.9940   0.9979  0.98627  0.98597  0.98603  0.98824
## Prevalence             0.1009   0.1189  0.09369  0.10090  0.09369  0.09009
## Detection Rate         0.0955   0.1171  0.08108  0.08829  0.08108  0.07928
## Detection Prevalence   0.0973   0.1351  0.08108  0.10090  0.09730  0.08108
## Balanced Accuracy      0.9722   0.9822  0.93269  0.93049  0.92375  0.93901
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9310  0.87037  0.83333  0.87719
## Specificity            0.9879  0.99202  0.98603  0.97390
## Pos Pred Value         0.9000  0.92157  0.86538  0.79365
## Neg Pred Value         0.9919  0.98611  0.98211  0.98577
## Prevalence             0.1045  0.09730  0.09730  0.10270
## Detection Rate         0.0973  0.08468  0.08108  0.09009
## Detection Prevalence   0.1081  0.09189  0.09369  0.11351
## Balanced Accuracy      0.9595  0.93119  0.90968  0.92554
```

Wow! After tuning the model, this random forest made predictions at nearly 90% accuracy.

Comparisons and Conclusion

As mentioned earlier, there is no "best overall" model in general. Each model will work best in different situations. With that being said, after using the models above, it is clear that the SVM, kNN, and Random Forest models were all better at recognizing the digits than the Decision Tree or Naive Bayes models. What this tells me is that this dataset is not linearly separable, which makes it hard for the Naive Bayes model to set decision boundaries. Seeing that the SVM, kNN, and Random Forest models are able to be flexible and work with complex data is what leads me to believe that any of those would be a more appropriate machine learning technique to use rather than a Decision Tree or Naive Bayes. Further, I would argue that the Random Forest model was the best at recognizing handwritten digits than all of the other models used in this report. At nearly 90%, the Random Forest model was about twice as accurate as either the Decision Tree or Naive Bayes model. This is due to the fact that variance is highly diminished since the model makes decisions based on decisions made in all the trees in the forest. At the end of the day, the results of this analysis show that higher accuracy can be achieved when there is as little variation as possible.