Parin Patel

SVMs, KNN, and Random Forest for Handwriting Recognition

Introduction:

The ability to recognize handwriting is an invaluable tool for various organizations and people wanting to convert text into digitized documents. This is especially common in our current digital world, where even chemical laboratories are now working to convert older documents into digital formats on their databases. Not only does it allow for ease-of-access, but this conversion allows documents to be better archived, distributed, and analyzed further.

The level of where organizations are using text analysis varies. For example, Microsoft has developed a service called Project Ink Analysis that converts handwritings and drawings into text. This is especially useful for those who work in fields where diagramming and drawing are often used alongside standard texts, like chemist working in a laboratory or database systems architects. Some organizations, however, are just beginning to shuffle through old journals and logs and convert them to digital records. In the past, most paperwork and notes were written on paper and then filed manually.

Today, while most organizations now scan documents for digital access, often the key topics and important information found in the past documents gets overlooked or misinterpreted due to simple misunderstanding of the written text. To combat this, and provide a more systematic record keeping methodology, organizations have begun using handwriting recognition to convert written text into digital ledgers.

There are numerous prediction methods that have been developed, and knowing when to use each type, based on how it fits the data is key to making a successful prediction. Previously, we looked at naiive Bayed and decision tree models to recognize handwriting .Now, we will use SVM, kNN, and Random Forest. In the end, we will compare the performance between the different types.

Analysis:

Data Pre-Processing

We begin by importing train and test the datasets. After, this initial analysis, we conclude we first need to convert our label variable into a factor. This will then be converted to factors that

can become variables names. The whole goal is to determine the likelihood that the observation belongs to a certain category.

```
> #convert label data type to factor
> digit_train$label <- factor(paste0('X', digit_train$label), levels = c('X0'
, 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9'))
> digit_test$label <- factor(paste0('X', digit_test$label), levels = c('X0',
'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9'))
```

kNN:

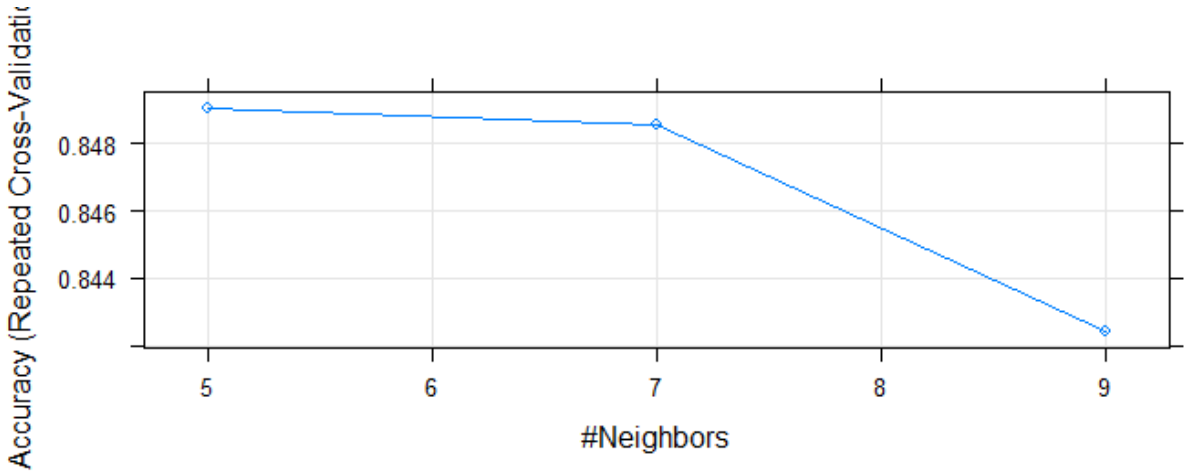
We will start by creating the kNN. We choose to do a three-fold CV . We set the seed to 239 create our first model.

```
> ##kNN : 3-fold CV
> set.seed(239)
> t <- trainControl(method = 'repeatedcv', repeats = 3, classProbs = TRUE)
> Train.kNN <- train(label ~ ., data = digit_train, method = 'knn', preProces
s = c('center', 'scale'), trControl = x, metric = 'ROC', tuneLength = 3)
```

Below are the results of the model. We will discuss this further in the Results section

```
> >Analyze Model
> Train.kNN
k-Nearest Neighbors

1400 samples
 784 predictor
  10 classes: 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9'

Pre-processing: centered (784), scaled (784)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 1261, 1261, 1261, 1259, 1260, 1262, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.8490694  0.8320306
  7  0.8485647  0.8314400
  9  0.8423990  0.8245379

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was k = 5.
```

Additionally, we graph the accuracy below. We can see that as our neighbors increases the accuracy falls. We then go on to predict the class label.



Below, we are predicting our class label from the kNN training data. Additionally, need to subtract by 1 to show this.

```
> ##Testing- kNN
> Pred.kNN <- predict(Train.kNN, digit_test, type = 'prob')
> Pred.kNN <- as.data.frame(Pred.kNN)
> Pred.kNN.df<- data.frame(apply(Pred.kNN, 1, which.max) - 1)
> colnames(Pred.kNN.df) <- 'prediction'

>
```

We then get our results by comparing our actual values to our predicted ones. Results will be down in the confusion matrix. This will be discussed in the next section.

```
> knnResults <- digit_test %>% select(label) %>% mutate(real = str_remove(lab
el, 'X')) %>% bind_cols(Pred.kNN.df) %>% mutate(real = as.factor(real), predi
ction = as.factor(prediction))
```

```
> confusionMatrix(knnResults$real, knnResults$prediction)

> Confusion Matrix and Statistics
>
>           Reference
> Prediction   0   1   2   3   4   5   6   7   8   9
>          0 382   0   0   3   0   2   2   0   1   1
>          1   0 476   1   0   1   0   0   0   0   0
>          2   5   9 412   5   5   1   3   2   3   0
>          3   1   7   8 368   0  12   1   7   4   2
>          4   0  11   3   0 358   0   1   4   0  18
>          5   2   6   5  17   1 319  11   2   6  12
>          6   8   0   6   0   2   3 425   0   0   0
>          7   1  11   2   1   9   0   0 380   2  18
>          8   3  15   4  18   5  31   4   4 307   3
>          9   1   5   2   7  19   1   0  29   2 358
>
> Overall Statistics
>
>                Accuracy : 0.8045
>                  95% CI : (0.7952, 0.8132)
>     No Information Rate : 0.0286
>     P-Value [Acc > NIR] : < 1.2e-16
>
>                   Kappa : 0.7938
>  Mcnemar's Test P-Value : NA
>
> Statistics by Class:
>
>                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
> Sensitivity           0.84864   0.8815   0.8300  0.87828  0.88500  0.86450
> Specificity           0.88762   0.8885   0.8812  0.88888  0.88026  0.88382
> Pos Pred Value        0.87778   0.8858   0.8258  0.88756  0.80633  0.83727
> Neg Pred Value        0.88447   0.8828   0.8817  0.88654  0.88886  0.88681
> Prevalence            0.08828   0.1286   0.1055  0.08876  0.08524  0.08786
> Detection Rate        0.08428   0.1133   0.0881  0.08762  0.08524  0.07585
> Detection Prevalence  0.08643   0.1138   0.1060  0.08762  0.08405  0.08071
> Balanced Accuracy     0.87363   0.8405   0.8606  0.83358  0.84263  0.82416
>                      Class: 6 Class: 7 Class: 8 Class: 9
> Sensitivity            0.8508  0.88785  0.84462  0.86883
> Specificity            0.8848  0.88834  0.87755  0.88258
> Pos Pred Value         0.8572  0.88623  0.77818  0.84434
> Neg Pred Value         0.8841  0.88728  0.88527  0.88570
> Prevalence             0.1064  0.10180  0.07738  0.08810
> Detection Rate         0.1012  0.08048  0.07310  0.08524
> Detection Prevalence   0.1057  0.10085  0.08381  0.10085
> Balanced Accuracy      0.8728  0.83808  0.86108  0.82575
```

SVM:

For our SVM model, we do not need to processes the data. Therefore, we can move onto training.

First, we will run the model using a linear kernel to classify the data. Again ,we choose to keep the 3- fold cross validation. The following process, as you will note, overall is similar to how we ran our kNN model. First we classified the data, then we ran our train model, and then our testing model. Finally we created a dataframe where we were able to clearly compare the results.

```
> svmTrain <- svm(label ~ ., data = digit_train, type = 'C', kernel = 'linear
', cross = 3, probability = TRUE)

> summary(svmTrain)

Call:
svm(formula = label ~ ., data = digit_train, type = "C", kernel = "linear",
    cross = 3, probability = TRUE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.00127551

Number of Support Vectors:  738

 ( 60 61 76 86 58 63 82 71 87 83 )


Number of Classes:  10

Levels:
 X0 X1 X2 X3 X4 X5 X6 X7 X8 X8

3-fold cross-validation on training data:

Total Accuracy: 88.57143
Single Accuracies:
 86.68528 80.14888 88.8651
```

Next, we will run the test set on the model.

```
> svmPred <- predict(svmTrain, digit_test, type = 'C')
> svmPred <- as.data.frame(svmPred)
> colnames(svmPred) <- 'results'
```

Finally ,we will build the dataframe to compare the results. As with our previous model, we will discuss our confusion matrix later in the Results.

```
confusionMatrix(svmResults$real, svmResults$prediction)
> Confusion Matrix and Statistics
>
>           Reference
> Prediction   0   1   2   3   4   5   6   7   8   9
>          0 382   0   2   0   1   4   4   1   1   0
>          1   0 469   0   1   1   1   0   1   5   0
>          2   1   3 408   3   8   1   5   4  12   0
>          3   1   5  10 351   1  22   2   4  10   4
>          4   1   3   1   0 369   0   2   3   1  15
>          5   5   2   4  21   5 319   8   1  12   4
>          6   3   0   4   0   8   4 419   0   6   0
>          7   2   3   6   2   6   1   0 387   2  15
>          8   4  11   8   9   1  16   4   3 333   5
>          9   0   3   2   6  16   2   0  23   6 366
>
> Overall Statistics
>
>                Accuracy : 0.8079
>                  95% CI : (0.7987, 0.8164)
>     No Information Rate : 0.0288
>     P-Value [Acc > NIR] : < 1.2e-16
>
>                   Kappa : 0.7976
>  Mcnemar's Test P-Value : NA
>
> Statistics by Class:
>
>                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
> Sensitivity           0.85844   0.8388  0.81685  0.88313  0.88702  0.86216
> Specificity           0.88657   0.8876  0.88015  0.88450  0.88313  0.88381
> Pos Pred Value        0.86780   0.8812  0.81685  0.85610  0.83418  0.83727
> Neg Pred Value        0.88552   0.8818  0.88015  0.88882  0.88765  0.88665
> Prevalence            0.08738   0.1988  0.10585  0.08357  0.08805  0.08810
> Detection Rate        0.08333   0.1117  0.08714  0.08357  0.08786  0.07585
> Detection Prevalence  0.08643   0.1138  0.10585  0.08762  0.08405  0.08071
> Balanced Accuracy     0.87750   0.8687  0.85350  0.83882  0.84007  0.82288
>                      Class: 6 Class: 7 Class: 8 Class: 9
> Sensitivity           0.84368  0.80632  0.85825  0.88487
> Specificity           0.88334  0.88018  0.88400  0.88470
> Pos Pred Value        0.84368  0.81274  0.84518  0.86321
> Neg Pred Value        0.88334  0.88841  0.88555  0.88861
> Prevalence            0.10571  0.10167  0.08938  0.08938
> Detection Rate        0.08876  0.08214  0.07828  0.08714
> Detection Prevalence  0.10571  0.10085  0.08381  0.10085
> Balanced Accuracy     0.86852  0.84826  0.82112  0.83878
```

Random Forest

Our third model will be created using random forest. The process to create this model was slightly different than kNN and SVM. For random forest, we first created our control parameters for our training data. It should also be noted that we kept our 3-fold cross validation.

```
> rfTrain
Random Forest

1400 samples
 784 predictor
  10 classes: 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X8'
```

```
No pre-processing
Resampling: Cross-Validated (3 fold, repeated 3 times)
Summary of sample sizes: 833, 832, 835, 834, 833, 833, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
    2   0.8171488  0.7838507
   38   0.8008503  0.8888442
  783   0.8623741  0.8468288

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was mtry = 38.
```

Using our results from the controlled training model, we were able to get our first initial assessment of accuracy. We now build the testing model, which we will use to compare to our training set.

```
> #test rf
> rfPred <- predict(rfTrain, digit_test, type = 'prob')
> rfPred <- as.data.frame(rfPred)

 > rfPredictedValues <- data.frame(apply(rfPred, 1, which.max) - 1)
 > colnames(rfPredictedValues) <- 'results'
```

After building our predication model from our training set, we use it to determine the predicted label values, which we use to compare to our actual label values. The difference between the two models will determine our accuracy. We use a confusion matrix to assess the prediction capabilities.

```
> #Rf results and confusion matrix
> rfResults <- testSet %>% select(label) %>% bind_cols(rfPredictedValues) %>%
mutate(real = factor(as.character(str_remove(label, 'X'))), prediction = fact
or(results))

> confusionMatrix(rfResults$real, rfResults$prediction)
```

```
> confusionMatrix(rfResults$real, rfResults$prediction)


> Confusion Matrix and Statistics
>
>           Reference
> Prediction   0   1   2   3   4   5   6   7   8   9
>          0 382   0   1   0   0   1   2   0   3   0
>          1   0 470   2   1   1   2   1   0   0   1
>          2   1   1 424   1   6   0   3   3   4   2
>          3   2   6  12 368   1   8   0   4   6   3
>          4   0   1   2   0 379   0   3   0   0  10
>          5   4   1   1   9   0 359   6   0   1   0
>          6   4   1   0   0   1   2 432   1   3   0
>          7   0   2   6   2   2   0   1 403   2   6
>          8   2   7   0   8   2   5   5   1 359   5
>          9   3   2   0   9   7   4   1  12   5 381
>
> Overall Statistics
>
>                Accuracy : 0.846
>                  95% CI : (0.8387, 0.8526)
>     No Information Rate : 0.0269
>     P-Value [Acc > NIR] : < 1.2e-16
>
>                   Kappa : 0.8399
>  Mcnemar's Test P-Value : NA
>
> Statistics by Class:
>
>                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
> Sensitivity           0.86135   0.8572   0.8464  0.82462  0.84887  0.84226
> Specificity           0.88815   0.8878   0.8844  0.88885  0.88578  0.88424
> Pos Pred Value        0.88272   0.8833   0.8528  0.88756  0.85848  0.84226
> Neg Pred Value        0.88578   0.8844   0.8836  0.88208  0.88474  0.88424
> Prevalence            0.08857   0.1168   0.1067  0.08476  0.08500  0.08071
> Detection Rate        0.08476   0.1118   0.1010  0.08762  0.08024  0.08548
> Detection Prevalence  0.08643   0.1138   0.1060  0.08762  0.08405  0.08071
> Balanced Accuracy     0.87875   0.8775   0.8704  0.85678  0.87283  0.86825
>                      Class: 6 Class: 7 Class: 8 Class: 9
> Sensitivity            0.8515  0.85047  0.83734  0.83382
> Specificity            0.8868  0.88444  0.88083  0.88866
> Pos Pred Value         0.8730  0.85047  0.81117  0.88858
> Neg Pred Value         0.8841  0.88444  0.88368  0.88285
> Prevalence             0.1081  0.10085  0.08118  0.08714
> Detection Rate         0.1028  0.08585  0.08548  0.08071
> Detection Prevalence   0.1057  0.10085  0.08381  0.10085
> Balanced Accuracy      0.8742  0.87246  0.86408  0.86124
```

Results:

From the results of our confusion matrix we are able to first see the accuracy rates between all three models. The first kNN model, we had a 80% accuracy. Our kappa was 79%. In the second SVM model, our accuracy and kappa increased slightly, where it was 80% accurate and 80% kappa. Finally, in our final Random Forest model, we had the highest level of accuracy, which was 84% and a higher kappa of 83%. From our three model, it appears the that the random forest was the most accurate.

Conclusion:

Based on the results of the accuracy measurements, overall the kNN and the SVM were lower than expected. This is likely due to technical problems, there was some issues with RStudio maintaining installed packages and not having enough memory to operate executed functions. This likely led to the decreased accuracy overall since we had to use the smaller, pre-split dataset. Overall, however, these results are better than what we got through the use of Naive Bayes and decision trees in predicting handwriting. Additionally, it is important to note that the sheer size of the original dataset forced us to work with only a small subset of the original data. Since we did not create this, we are not completely sure of how the data was split. We had many problems initially when working with the large Kaggle dataset, and creating our own training and testing models. This is why we chose to stick with the smaller dataset given. Also, when testing kNN on the iris dataset, we used the kNN pre-build function. When doing so, we had a higher level of accuracy. We wonder if, in the future, we can tweak our kNN, SVM, and random forst models for different datasets to compare accuracy to the Naiive Bayes and Decision Trees.

Appendix A: R Code

```
# Load the required packages.
require("caret")
require("e1071")
require("dplyr")
require("rpart")
require("stringr")
require("randomForest")
require("ggplot2")
require("dplyr")

>import datasets
```

```
setwd("C:\\Users\\parin\\Documents\\IST-707\\Hw_6")
#import train
filename='Kaggle-digit-train-sample-small-1400.csv'
digit_train<-read.csv(filename,header = TRUE, stringsAsFactors = TRUE)
#import test
filename1='Kaggle-digit-test-sample1000.csv'
digit_test<-read.csv(filename1,header = TRUE, stringsAsFactors = TRUE)
```

>explore-

```
dim(digit_train)
summary(digit_train[, 1:10])
str(digit_train[, 1:10])
```

>Training and testing the models.

```
#convert label data type to factor
digit_train$label <- factor(paste0('X', digit_train$label), levels = c('X0', 'X1', 'X2', 'X3', 'X4',
'X5', 'X6', 'X7', 'X8', 'X8'))
digit_test$label <- factor(paste0('X', digit_test$label), levels = c('X0', 'X1', 'X2', 'X3', 'X4', 'X5',
'X6', 'X7', 'X8', 'X8'))
```

>kNN : 3-fold CV

```
set.seed(238)
t <- trainControl(method = 'repeatedcv', repeats = 3, classProbs = TRUE)
Train.kNN <- train(label ~ ., data = digit_train, method = 'knn', preProcess = c('center', 'scale'),
trControl = x, metric = 'ROC', tuneLength = 3)
```

>Analyze Model

```
Train.kNN
plot(Train.kNN)
```

>Testing- kNN

```
Pred.kNN <- predict(Train.kNN, digit_test, type = 'prob')
Pred.kNN <- as.data.frame(Pred.kNN)
Pred.kNN.df<- data.frame(apply(Pred.kNN, 1, which.max) - 1)
colnames(Pred.kNN.df) <- 'prediction'

install.packages("tidyverse")
library("tidyverse")
require("stringr")
```

```
knnResults <- digit_test %>% select(label) %>% mutate(real = str_remove(label, 'X')) %>%
bind_cols(Pred.kNN.df) %>% mutate(real = as.factor(real), prediction = as.factor(prediction))

confusionMatrix(knnResults$real, knnResults$prediction)

str(digit_test)
head(digit)

>svm  - training
svmTrain <- svm(label ~ ., data = digit_train, type = 'C', kernel = 'linear', cross = 3, probability =
TRUE)
summary(svmTrain)

svmPred <- predict(svmTrain, digit_test, type = 'C')
svmPred <- as.data.frame(svmPred)
colnames(svmPred) <- 'results'

# Build the data frame to compare the results.
svmResults <- testSet %>% select(label) %>% bind_cols(svmPred) %>% mutate(real =
factor(as.character(str_remove(label, 'X'))), prediction = factor(as.character(str_remove(results,
'X'))))




>Random Forest
x <- trainControl(method = 'repeatedcv', number = 3, repeats = 3)
rfTrain <- train(label ~ ., data = digit_train, method = 'rf', metric = 'Accuracy', trControl = x, type
= 'C')

rfTrain

#test rf
rfPred <- predict(rfTrain, digit_test, type = 'prob')
rfPred <- as.data.frame(rfPred)

rfPredictedValues <- data.frame(apply(rfPred, 1, which.max) - 1)
colnames(rfPredictedValues) <- 'results'

#Rf results and confusion matrix
rfResults <- testSet %>% select(label) %>% bind_cols(rfPredictedValues) %>% mutate(real =
factor(as.character(str_remove(label, 'X'))), prediction = factor(results))
confusionMatrix(rfResults$real, rfResults$prediction)
```