Assignment 06

Section No. 1: Introduction

Briefly describe the classification problem and general data preprocessing. Note that some data preprocessing steps maybe specific to a particular algorithm. Report those steps under each algorithm section.

For this assignment, we are viewing a dataset with digits from 0 to 9 of all handwritten images. We will use both the decision tree and naïve bayes to see which model gives us the best prediction.

#import the following packages
library(dplyr)
library(caret)
library(rpart)
library(e1071)
library(stringr)

#import data
digittrain <- read.csv("/Volumes/STORE N GO/00 - Graduate School/00 - SYR In Session/00 - Fall 2019/IST 707 - Data Analytics Wed9PM/Deliverables/HWs/HW6/digittrain.csv")
digittest <- read.csv("/Volumes/STORE N GO/00 - Graduate School/00 - SYR In Session/00 - Fall 2019/IST 707 - Data Analytics Wed9PM/Deliverables/HWs/HW6/digittest.csv")

dim(digittrain)

We observe that the training dataset has 42,000 observations and 785 variables.

str(digittrain[, 1:10])

```
> str(digittrain[, 1:10])
'data.frame':	42000 obs. of  10 variables:
 $ label : Factor w/ 10 levels "0","1","2","3",..: 2 1 2 5 1 1 8 4 6 4 ...
 $ pixel0: int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel1: int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel2: int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel3: int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel4: int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel5: int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel6: int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel7: int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel8: int  0 0 0 0 0 0 0 0 0 0 ...
```

summary(digittrain[, 1:10])

dim(digittest)

We observe that the test dataset has 28,000 observations and 784 variables.

str(digittest[, 1:10])
summary(digittest[, 1:10])

The dataset is 0 through 9 handwritten images & 784-pixel variables. Our labels are "Pixels" and our "data type are all integers

We will be working with a sample of the data (due to the size) for building our model. As described in the "Task Description", we will be sampling 10% of our training and testing data to train and test to be used for our classifier.'''

#we set the seed for randomness
set.seed(150)

# selection of 10% of the training & test data
splitrain <- sample(nrow(digittrain), nrow(digittrain) * .1)
splitest <- sample(nrow(digittest), nrow(digittest) * .1)

# row selection to build our set
subtrain <- digittrain[splitrain,]
subtest <- digittest[splitest,]

#specify that the row names are "NULL", that there are none
row.names(digittrain) <- NULL
row.names(digittest) <- NULL

We will also attempt to view the heading of our datasets.

Section No. 2: Decision Tree

Build a decision tree model. Tune the parameters, such as the pruning options, and report the 3-fold CV accuracy.

```
# Decision tree model
digittreetrain <- rpart(label ~ ., data = subtrain, method = 'class',
              control = rpart.control(cp = 0), minsplit = 100, maxdepth = 10)
```

```
> digittreetrain
n= 4200

node), split, n, loss, yval, (yprob)
      * denotes terminal node

   1) root 4200 3745 1 (0.096 0.11 0.1 0.1 0.1 0.095 0.095 0.1 0.098 0.095)
     2) pixel409< 0.5 1378  970 1 (0.24 0.3 0.078 0.061 0.022 0.13 0.052 0.028 0.08 0.015)
       4) pixel462< 3 512  184 0 (0.64 0.0059 0.059 0.066 0.0039 0.09 0.053 0.043 0.021 0.018)
         8) pixel455>=9.5 319   40 0 (0.87 0 0.019 0.0094 0.0063 0.025 0.034 0.0063 0.0063 0.019)
          16) pixel351< 161.5 295   23 0 (0.92 0 0.017 0.0034 0.0068 0.014 0.01 0.0068 0 0.02)
            32) pixel517< 242.5 280   13 0 (0.95 0 0.0036 0.0036 0.0036 0.0071 0.0036 0.0071 0 0.018)
              64) pixel628>=18.5 250    3 0 (0.99 0 0.004 0.004 0 0.004 0 0 0 0) *
              65) pixel628< 18.5 30   10 0 (0.67 0 0 0 0.033 0.033 0.033 0.067 0 0.17)
               130) pixel442>=25 23    4 0 (0.83 0 0 0 0 0 0.043 0.043 0 0.087) *
               131) pixel442< 25 7    4 9 (0.14 0 0 0 0.14 0.14 0 0.14 0 0.43) *
            33) pixel517>=242.5 15   10 0 (0.33 0 0.27 0 0.067 0.13 0.13 0 0 0.067) *
          17) pixel351>=161.5 24   16 6 (0.29 0 0.042 0.083 0 0.17 0.33 0 0.083 0)
            34) pixel515< 55 16    9 0 (0.44 0 0.062 0.12 0 0.25 0 0 0.12 0) *
            35) pixel515>=55 8    0 6 (0 0 0 0 0 0 1 0 0 0) *
         9) pixel455< 9.5 193  144 0 (0.25 0.016 0.12 0.16 0 0.2 0.083 0.1 0.047 0.016)
          18) pixel324< 13.5 88   42 0 (0.52 0.023 0.11 0.034 0 0.14 0.1 0.068 0 0)
            36) pixel517< 16.5 68   22 0 (0.68 0.015 0.029 0.044 0 0.1 0.044 0.088 0 0)
              72) pixel405< 38.5 56   10 0 (0.82 0 0.018 0 0 0.018 0.036 0.11 0 0) *
              73) pixel405>=38.5 12    6 5 (0 0.083 0.083 0.25 0 0.5 0.083 0 0 0) *
            37) pixel517>=16.5 20   12 2 (0 0.05 0.4 0 0 0.25 0.3 0 0 0)
              74) pixel239< 11 9    1 2 (0 0 0.89 0 0 0.11 0 0 0 0) *
              75) pixel239>=11 11    5 6 (0 0.091 0 0 0 0.36 0.55 0 0 0) *
          19) pixel324>=13.5 105   77 3 (0.029 0.0095 0.13 0.27 0 0.25 0.067 0.13 0.086 0.029)
            38) pixel514>=49 36   22 2 (0.028 0.028 0.39 0.028 0 0.083 0.19 0 0.25 0)
              76) pixel436< 148 26   12 2 (0.038 0.038 0.54 0.038 0 0.12 0.23 0 0 0)
               152) pixel355< 6 15    1 2 (0 0.067 0.93 0 0 0 0 0 0 0) *
               153) pixel355>=6 11    5 6 (0.091 0 0 0.091 0 0.27 0.55 0 0 0) *
              77) pixel436>=148 10    1 8 (0 0 0 0 0 0 0.1 0 0.9 0) *
            39) pixel514< 49 69   42 3 (0.029 0 0 0.39 0 0.33 0 0.2 0 0.043)
              78) pixel287< 175 52   26 3 (0.038 0 0 0.5 0 0.4 0 0.019 0 0.038)
               156) pixel262< 11.5 30    7 3 (0.033 0 0 0.77 0 0.13 0 0.033 0 0.033)
                 312) pixel152>=7 20    0 3 (0 0 0 1 0 0 0 0 0 0) *
                 313) pixel152< 7 10    6 5 (0.1 0 0 0.3 0 0.4 0 0.1 0 0.1) *
               157) pixel262>=11.5 22    5 5 (0.045 0 0 0.14 0 0.77 0 0 0 0.045) *
              79) pixel287>=175 17    4 7 (0 0 0 0.059 0 0.12 0 0.76 0 0.059) *
       5) pixel462>=3 866  461 1 (0.0046 0.47 0.089 0.058 0.033 0.15 0.051 0.02 0.11 0.014)
        10) pixel347< 1.5 570  180 1 (0 0.68 0.11 0.044 0.023 0.046 0.014 0.028 0.039 0.0088)
          20) pixel551< 6 476   90 1 (0 0.81 0.036 0.025 0.0084 0.042 0.0063 0.029 0.038 0.0042)
            40) pixel350>=39.5 430   49 1 (0 0.89 0.012 0.023 0.0047 0.012 0.007 0.016 0.04 0)
              80) pixel300< 75 412   32 1 (0 0.92 0.012 0.015 0.0049 0.0097 0.0073 0.017 0.012 0)
               160) pixel494< 48 402   22 1 (0 0.95 0.005 0.012 0.0025 0.01 0.0075 0.005 0.012 0)
                 320) pixel206< 48.5 392   15 1 (0 0.96 0 0.0026 0 0.01 0.0077 0.0051 0.013 0) *
                 321) pixel206>=48.5 10    6 3 (0 0.3 0.2 0.4 0.1 0 0 0 0 0) *
               161) pixel494>=48 10    5 7 (0 0 0.3 0.1 0.1 0 0 0.5 0 0) *
              81) pixel300>=75 18    6 8 (0 0.056 0 0.22 0 0.056 0 0 0.67 0) *
            41) pixel350< 39.5 46   31 5 (0 0.11 0.26 0.043 0.043 0.33 0 0.15 0.022 0.043)
```

```
# Testing the accuracy of the tree on the training set
trainacc <- data.frame(predict(digittreetrain, subtrain))

#choose max likelihood
trainacc <- as.data.frame(names(trainacc[apply(trainacc, 1, which.max)]))
colnames(trainacc) <- 'prediction'
trainacc$number <- substr(trainacc$prediction, 2, 2)
```

trainacc <- subtrain %>% bind_cols(trainacc) %>% select(label, number) %>% mutate(label = as.factor(label), number = as.factor(round(as.numeric(number), 0)))

Confusion Matrix

# Now let's build the Confusion matrix so we can examine the accuracy percentage
confusionMatrix(trainacc$label, trainacc$number)

#our training set gives us an 86% (approximate) accuracy.

```
> confusionMatrix(trainacc$label, trainacc$number)
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2   3   4   5   6   7   8   9
         0 373   0   3   8   6   4   2   1   5   3
         1   0 415   4   9   9   3   6   7   2   0
         2  12   4 360  12  12   4   5   5   3   7
         3   4   1   7 338  10  26   5   6  18  12
         4   6   2   5  11 386   3   3   2   6  17
         5  10   6   3  22   7 318  11   6   5  12
         6   6   3   4   3  13   5 362   0   2   1
         7   7   3  10   3  21   4   1 372   4  13
         8   5   6   9   8  14   8   5   1 346  11
         9   6   0   6   5  26   8   2   7   4 334

Overall Statistics

               Accuracy : 0.8581
                 95% CI : (0.8472, 0.8685)
    No Information Rate : 0.12
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8423

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.86946  0.94318  0.87591  0.80668   0.7659  0.83029  0.90050
Specificity           0.99151  0.98936  0.98311  0.97646   0.9851  0.97852  0.99026
Pos Pred Value        0.92099  0.91209  0.84906  0.79157   0.8753  0.79500  0.90727
Neg Pred Value        0.98524  0.99332  0.98649  0.97853   0.9686  0.98289  0.98948
Prevalence            0.10214  0.10476  0.09786  0.09976   0.1200  0.09119  0.09571
Detection Rate        0.08881  0.09881  0.08571  0.08048   0.0919  0.07571  0.08619
Detection Prevalence  0.09643  0.10833  0.10095  0.10167   0.1050  0.09524  0.09500
Balanced Accuracy     0.93049  0.96627  0.92951  0.89157   0.8755  0.90440  0.94538
                     Class: 7 Class: 8 Class: 9
Sensitivity           0.91400  0.87595  0.81463
Specificity           0.98260  0.98239  0.98311
Pos Pred Value        0.84932  0.83777  0.83920
Neg Pred Value        0.99070  0.98706  0.98001
Prevalence            0.09690  0.09405  0.09762
Detection Rate        0.08857  0.08238  0.07952
Detection Prevalence  0.10429  0.09833  0.09476
Balanced Accuracy     0.94830  0.92917  0.89887
```

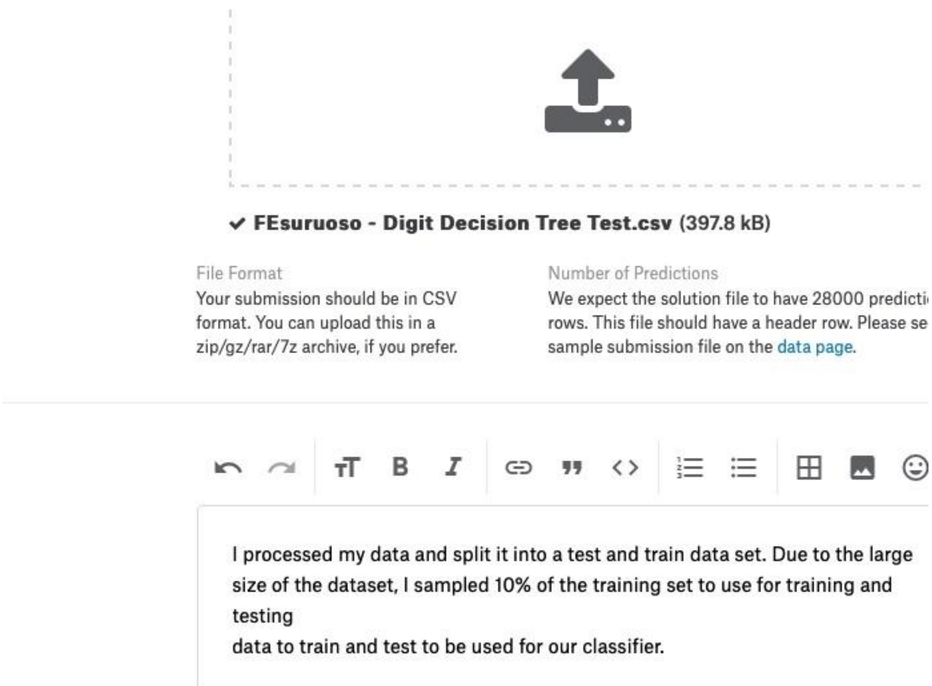
# now we run the same prediction on the test data
testacc <- data.frame(predict(digittreetrain, subtest))
testacc <- as.data.frame(names(testacc[apply(testacc, 1, which.max)]))
colnames(testacc) <- 'prediction'
testacc$number <- substr(testacc$prediction, 2, 2)

Seems about even but we cannot test the results unless we run the decision tree on the entire test set and submit it to Kaggle.

```
finaltest <- data.frame(predict(digittreetrain, digittest))
finaltest <- as.data.frame(names(finaltest[apply(finaltest, 1, which.max)]))
colnames(finaltest) <- 'ImageId'
finaltest$Label <- substr(finaltest$ImageId, 2, 2)
finaltest$ImageId <- 1:nrow(finaltest)

#Now we can export the model (file) and view Kaggle results
write.csv(finaltest, file="/Volumes/STORE N GO/00 - Graduate School/00 - SYR In Session/00 - Fall 2019/IST 707 - Data Analytics Wed9PM/Deliverables/HWs/HW6/FEsuruoso - Digit Decision Tree Test.csv")
```

Sample of our Kaggle submission. We recorded an 81% Kaggle score (approximate).



Section No. 3: Naïve Bayes

Instruction: Build a naïve Bayes model. Tune the parameters, such as the discretization options, to compare results.

#build classifier
digittrainnb <- naiveBayes(as.factor(label) ~ ., data = subtrain)

nbtrainacc <- predict(digittrainnb, subtrain, type = 'class')
confusionMatrix(nbtrainacc, as.factor(subtrain$label))

```
> confusionMatrix(nbtrainacc, as.factor(subtrain$label))
Confusion Matrix and Statistics

          (No matches)
Prediction   0   1   2   3   4   5   6   7   8   9
         0 391   0  83  71  32 104  11   8  10  15
         1   1 443  22  27  13  32  18  24  99  35
         2   0   0  75   2   0   1   0   1   3   0
         3   0   0  54 180   1  12   0   1   7   0
         4   0   0   4   1  75   0   2   1   2   1
         5   0   0   0   0   0  14   0   0   0   0
         6   7   5  99  13  65  15 360   2   3   9
         7   0   0   1   6  16   5   0 367   0  73
         8   6   7  85 121 166 207   8  23 284  59
         9   0   0   1   6  73  10   0  11   5 206

Overall Statistics

               Accuracy : 0.5702
                 95% CI : (0.5551, 0.5853)
    No Information Rate : 0.1083
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5225

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity           0.96543   0.9736  0.17689  0.42155  0.17007 0.035000  0.90226  0.83790  0.68765  0.51759
Specificity           0.91199   0.9276  0.99815  0.98012  0.99707 1.000000  0.94265  0.97315  0.81991  0.97212
Pos Pred Value        0.53931   0.6204  0.91463  0.70588  0.87209 1.000000  0.62284  0.78419  0.29400  0.66026
Neg Pred Value        0.99597   0.9966  0.91525  0.93739  0.91104 0.907788  0.98923  0.98098  0.96011  0.95062
Prevalence            0.09643   0.1083  0.10095  0.10167  0.10500 0.095238  0.09500  0.10429  0.09833  0.09476
Detection Rate        0.09310   0.1055  0.01786  0.04286  0.01786 0.003333  0.08571  0.08738  0.06762  0.04905
Detection Prevalence  0.17262   0.1700  0.01952  0.06071  0.02048 0.003333  0.13762  0.11143  0.23000  0.07429
Balanced Accuracy     0.93871   0.9506  0.58752  0.70083  0.58357 0.517500  0.92245  0.90553  0.75378  0.74485
```

Here we only see a 57% (approximate) test accuracy. This is not as good as the previous model.

nbtestacc <- predict(digittrainnb, digittest, type = 'class')
nbtestacc <- data.frame(nbtestacc)
colnames(nbtestacc)[1] <- 'Label'
nbtestacc$ImageId <- 1:nrow(nbtestacc)
nbtestacc <- nbtestacc %>% select(ImageId, Label)

# We will now submit our Naive Bayes algorithm to kaggle. First we need to export to csv
write.csv(nbtestacc, file="/Volumes/STORE N GO/00 - Graduate School/00 - SYR In Session/00 - Fall 2019/IST 707 - Data Analytics Wed9PM/Deliverables/HWs/HW6/FEsuruoso - Naive Bayes Classifier.csv", row.names = FALSE)

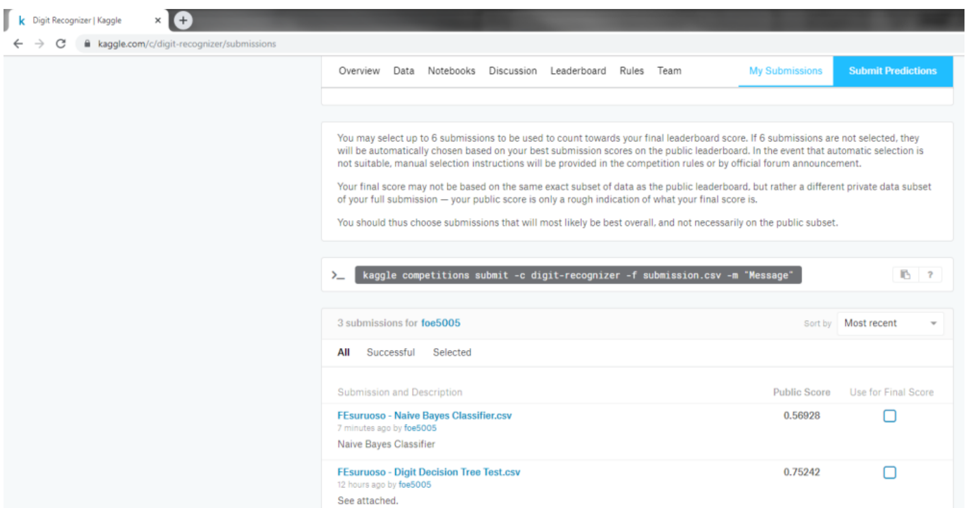
Section No. 4: Algorithm Performance Comparison

Report the test accuracy for the naïve Bayes and decision tree models. Discuss whether overfitting occurs in these models.

I was not pleased with the results obtained by the Naïve Bayes algorithm. I received a 57% in my confidence matrix and a57% on my Kaggle submission. I did not perform any pruning measures like I did with the previous model, so this might have a lot to do with it. There also weren't any additional parameters set.

One of the most significant measures of how good our models did was the Confusion Matrix accuracy. While we had around 80% for the first model, our second model only gave us around 57%. We see that the first model also properly identified our models better. It is also worthy to note that had we used a bigger portion of our data (we only used 10%) and tuned better, we might have had a better model.

Section No. 5: Decision Tree & Naïve Bayes Kaggle Test Result



As shown above, I scored approximately 75% on my decision tree submission and 57% on my Naïve Bayes submission. Both scores could be improved greatly by the following:

- Increasing the sample size in our models from 10% to a larger sample size (maybe 20% or 40%)
- Additional tuning that addresses model fitting
- Address specific model issues (i.e smaller dataset for Naïve Bayes and larger dataset for decision tree)