Dan Tully

IST 707 HW6/7

03/13/21

## Introduction

### General:

Technology today is advancing extremely quickly. For a world that has gone digital, there is still plenty of handwritten material around. In the United States, handwriting still rules in certain areas. Mobile technology use is up throughout the country but not necessarily as much as laptop and desktop usage. That means companies still require customers to fill in forms and those handwritten documents would have to be manually entered or perhaps scanned into computers.

MNIST (“Modified National Institute of Standards and Technology”) is the de facto “hello world” dataset of computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms.(*Digit Recognizer | Kaggle*, n.d.).

With this data set provided, we will examine the digit recognizer using Naïve Bayes, Decision Tree, kNN, Random Forest and SVM algorithms. The models will be evaluated for accuracy and performance.

## Analysis and Models

### About the Data:

The data is loaded. The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive. The training data set, (train.csv), has 785 columns. The first column, called “label”, is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image. (*Digit Recognizer | Kaggle*, n.d.).

Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i \* 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixelx is located on row i and column j of a 28 x 28 matrix, (indexing by zero). (*Digit Recognizer | Kaggle*, n.d.).

#load the data  
setwd("C:\\Users\\copla\\OneDrive\\Documents\\my\_school\\SU\\IST707\\")  
train <- read.csv("21069 Kaggle\_train.csv", header=TRUE)  
test <- read.csv("21069 Kaggle\_test.csv", header=TRUE)  
  
#how many rows  
nrow(train)

## [1] 42000

nrow(test)

## [1] 28000

my\_kable(train)

Fig. 1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| label | pixel0 | pixel1 | pixel2 | pixel3 | pixel4 | pixel5 | pixel6 | pixel7 | pixel8 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

tc <- tc + 1  
my\_kable(test)

Fig. 2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| pixel0 | pixel1 | pixel2 | pixel3 | pixel4 | pixel5 | pixel6 | pixel7 | pixel8 | pixel9 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

tc <- tc + 1  
str(train, list.len=10)

## 'data.frame': 42000 obs. of 785 variables:  
## $ label : int 1 0 1 4 0 0 7 3 5 3 ...  
## $ 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 ...  
## [list output truncated]

str(test, list.len=10)

## 'data.frame': 28000 obs. of 784 variables:  
## $ 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 ...  
## $ pixel9 : int 0 0 0 0 0 0 0 0 0 0 ...  
## [list output truncated]

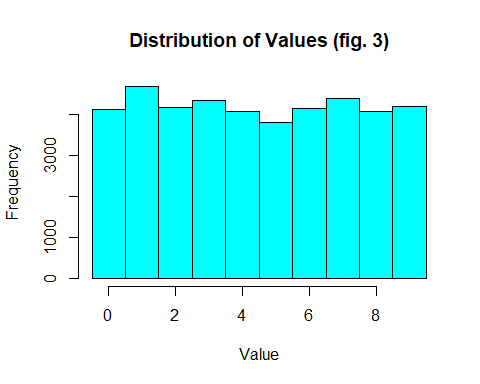
dim(train)

## [1] 42000 785

dim(test)

## [1] 28000 784

Reviewing the data below you can see how many train examples we have per label. These are the Levels: 0 1 2 3 4 5 6 7 8 9.

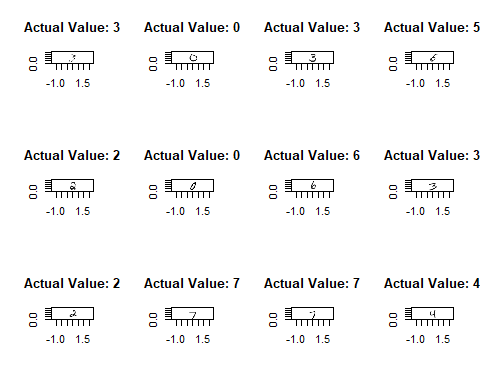


Inspect and visualizing some of the source data.

knitr::opts\_chunk$set(warning = FALSE, message = FALSE)  
#plot  
flip <- function(matrix){  
 apply(matrix, 2, rev)  
}  
#random sampling  
plotdigit <- function(datarow, rm1=F){  
 title <- datarow[1]  
 if(rm1){  
 datarow <- datarow[-1]  
 }  
  
datarow <- as.numeric(datarow)  
x <- rep(0:27)/27  
y <- rep(0:27)/27  
z <- matrix(datarow, ncol=28, byrow=T)  
rotate <- function(x) t(apply(x,2,rev))  
z <- rotate(z)  
image(x, y, z, main=paste("Actual Value:", title), col=gray.colors(255, start=1, end=0), asp=1,  
 xlim=c(0,1), useRaster=T, xlab='', ylab='')  
}  
paste("Sample Digital Values (fig. ",tc,")", sep = "")

## [1] "Sample Digital Values (fig. 4)"

tc <- tc + 1  
par(mfrow=c(3,4))  
set.seed(1)  
rows <- sample(1:42000, size=12)  
for(i in rows){  
 plotdigit(train[i,],rm1=T)  
}



Sample Digital Values

Reducing sample size to aid in using some of the algorithms. Initial Cleaning.

## creating samples for training and testing datasets  
train\_sam <- train[seq(1,nrow(train),10),]  
test\_sam <- test[seq(1,nrow(test),10),]  
# removing pixels that have 0  
train\_clean <- train\_sam[,colSums(train\_sam !=0)>0]  
#Removing pixels w/low variances  
all\_var <- data.frame(apply(train\_clean[-1],2,var))  
colnames(all\_var) <- "Variances"  
  
#sorting variances  
all\_var <- all\_var[order(all\_var$Variances), , drop=FALSE]  
num\_labels <- c(1:661)  
numbered\_var <- cbind(all\_var, num\_labels)  
paste("Variances (fig. ",tc,")", sep = "")

## [1] "Variances (fig. 5)"

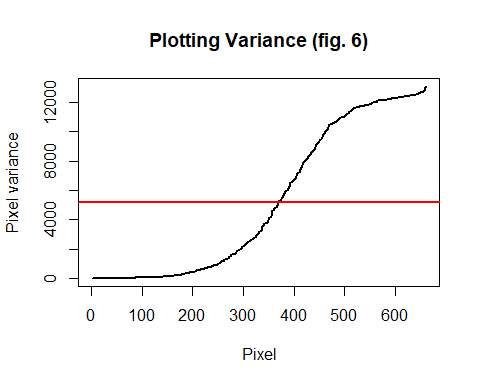
summary(all\_var)

## Variances   
## Min. : 0.001   
## 1st Qu.: 191.852   
## Median : 3157.256   
## Mean : 5181.810   
## 3rd Qu.:11005.109   
## Max. :13102.379

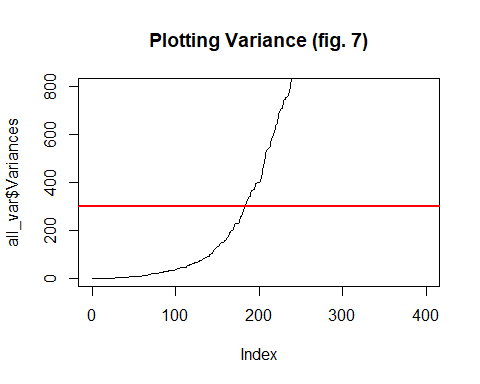
tc=tc+1

Plotting all variances by pixels (Fig. 6) and by index (Fig. 7).

plot(all\_var$Variances, type = "l",main = paste("Plotting Variance (fig. ",tc,")", sep = ""),  
 xlab="Pixel", ylab = "Pixel variance", lwd=2)  
abline(h=5181, col="red", lwd=2)



tc=tc + 1  
  
plot(all\_var$Variances, type = "l",main = paste("Plotting Variance (fig. ",tc,")", sep = ""),  
 xlim=c(0,400), ylim = c(0,800))  
abline(h=300, col="red", lwd=2)



tc=tc + 1  
  
good\_var <- subset(all\_var, all\_var$Variances >=300, "Variances")  
good\_var\_pixels <- row.names(good\_var)  
train\_clean <- train\_clean[,c("label", good\_var\_pixels)]

Normalize the data utilizing the *min\_max\_func* (Fig. 8).

min\_max\_func <- function(x) {  
 a <- (x-min(x))  
 b <- (max(x)-min(x))  
 return(a/b)  
}  
  
clean\_train\_nolabel <- train\_clean[,-1]  
clean\_train\_nolabel\_normalize <- as.data.frame(lapply(clean\_train\_nolabel, min\_max\_func))  
train\_clean <- cbind(label=train\_clean$label, clean\_train\_nolabel\_normalize)  
my\_kable(train\_clean)

Fig. 8

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| label | pixel720 | pixel75 | pixel535 | pixel67 | pixel677 | pixel193 | pixel648 | pixel666 | pixel693 |
| 1 | 0 | 0 | 0 | 0 | 0.0000000 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0.0431373 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0.0000000 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0.0000000 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0.0000000 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0.0000000 | 0 | 0 | 0 | 0 |

tc=tc + 1

### Models:

### Decision tree

A decision tree is very useful in helping you forecast potential outcomes for each option.

train\_clean$label <- as.factor(train\_clean$label)  
set.seed(123)  
idx <- sample(1:nrow(train\_clean), size=0.8 \* nrow(train\_clean))  
train\_set <- train\_clean[idx,]  
test\_set <- train\_clean[-idx,]  
train\_set\_numonly <- train\_set[,-1]  
train\_set\_labels <- train\_set[,1]  
test\_set\_numonly <- test\_set[,-1]  
test\_set\_labels <- test\_set[,1]

m1=J48(label~., data = train\_set)  
paste("Decision Tree Evaluation (fig. ",tc,")", sep = "")

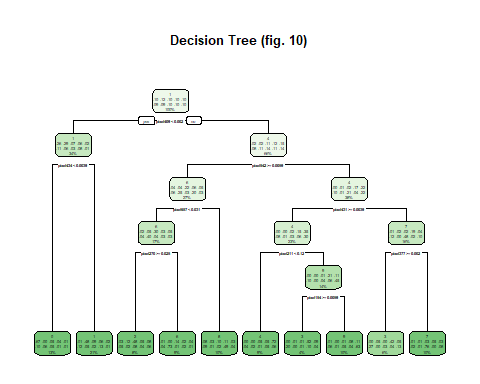
## [1] "Decision Tree Evaluation (fig. 9)"

tc=tc + 1  
e1 <- evaluate\_Weka\_classifier(m1,  
 numFolds = 3,  
 seed = 1, class = TRUE)  
e1

## === 3 Fold Cross Validation ===  
##   
## === Summary ===  
##   
## Correctly Classified Instances 2515 74.8512 %  
## Incorrectly Classified Instances 845 25.1488 %  
## Kappa statistic 0.7203  
## Mean absolute error 0.0536  
## Root mean squared error 0.2133  
## Relative absolute error 29.7972 %  
## Root relative squared error 71.1071 %  
## Total Number of Instances 3360   
##   
## === Detailed Accuracy By Class ===  
##   
## TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class  
## 0.874 0.025 0.800 0.874 0.835 0.816 0.928 0.708 0  
## 0.913 0.019 0.865 0.913 0.888 0.874 0.960 0.852 1  
## 0.747 0.029 0.733 0.747 0.740 0.712 0.862 0.626 2  
## 0.658 0.035 0.678 0.658 0.668 0.631 0.827 0.517 3  
## 0.712 0.037 0.688 0.712 0.700 0.665 0.852 0.548 4  
## 0.629 0.032 0.668 0.629 0.648 0.614 0.812 0.523 5  
## 0.803 0.018 0.819 0.803 0.811 0.792 0.921 0.741 6  
## 0.754 0.023 0.788 0.754 0.770 0.745 0.880 0.646 7  
## 0.698 0.031 0.713 0.698 0.705 0.673 0.845 0.528 8  
## 0.660 0.031 0.695 0.660 0.677 0.644 0.832 0.516 9  
## Weighted Avg. 0.749 0.028 0.747 0.749 0.747 0.720 0.874 0.624   
##   
## === Confusion Matrix ===  
##   
## a b c d e f g h i j <-- classified as  
## 304 2 8 4 4 9 4 2 9 2 | a = 0  
## 0 358 7 5 2 6 1 4 9 0 | b = 1  
## 8 8 239 11 4 9 14 10 14 3 | c = 2  
## 10 6 26 221 6 30 2 10 16 9 | d = 3  
## 8 6 8 12 247 11 8 4 8 35 | e = 4  
## 15 8 3 37 20 195 11 2 15 4 | f = 5  
## 13 3 11 1 16 4 244 1 8 3 | g = 6  
## 4 6 9 9 15 3 1 260 8 30 | h = 7  
## 10 14 12 13 14 14 11 5 233 8 | i = 8  
## 8 3 3 13 31 11 2 32 7 214 | j = 9

Visualization of the decision tree (Fig. 10).

#Apply the model with test dataset  
pred=predict (m1, newdata = test\_set, type = c("class"))  
DT <- rpart(label ~.,data=train\_set, cp=.022)  
rpart.plot(DT,main = paste("Decision Tree (fig. ",tc,")", sep = ""), box.palette="Gn")



tc=tc + 1

### Naïve Bayes

This probabilistic machine learning algorithm that can be used in a wide variety of classification tasks and assumes that all predictors are independent.

## using naivebayes package  
## https://cran.r-project.org/web/packages/naivebayes/naivebayes.pdf  
  
NB\_object<- naiveBayes(label~., data=train\_set, kernel=KT\_P, cost=100, scale=FALSE)  
NB\_prediction<-predict(NB\_object, train\_set\_numonly, type = c("class"))  
paste("Naive Bayes Evaluation (fig. ",tc,")", sep = "")

## [1] "Naive Bayes Evaluation (fig. 11)"

tc=tc+1  
NB\_object$apriori

## Y  
## 0 1 2 3 4 5 6 7 8 9   
## 348 392 320 336 347 310 304 345 334 324

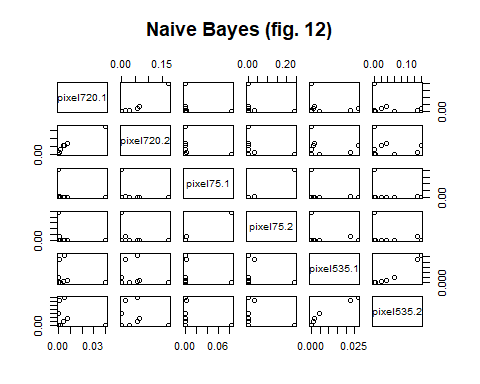
#as.data.frame(NB\_object$tables[1:5])  
head(NB\_prediction)

## [1] 7 0 1 2 7 0  
## Levels: 0 1 2 3 4 5 6 7 8 9

confusionMatrix(NB\_prediction,train\_set\_labels)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 309 0 2 4 0 17 1 3 3 5  
## 1 0 375 11 14 7 20 7 16 49 10  
## 2 2 0 192 13 3 4 2 2 5 1  
## 3 0 2 8 223 0 25 0 3 7 2  
## 4 0 0 4 5 195 14 1 5 4 5  
## 5 1 1 2 4 3 113 6 0 5 0  
## 6 17 5 69 15 16 17 284 0 7 2  
## 7 0 0 4 13 14 8 0 292 4 19  
## 8 11 4 26 26 3 73 3 2 211 0  
## 9 8 5 2 19 106 19 0 22 39 280  
##   
## Overall Statistics  
##   
## Accuracy : 0.7363   
## 95% CI : (0.7211, 0.7511)  
## No Information Rate : 0.1167   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.7067   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.88793 0.9566 0.60000 0.66369 0.56196 0.36452  
## Specificity 0.98838 0.9549 0.98947 0.98446 0.98739 0.99279  
## Pos Pred Value 0.89826 0.7367 0.85714 0.82593 0.83691 0.83704  
## Neg Pred Value 0.98707 0.9940 0.95918 0.96343 0.95139 0.93891  
## Prevalence 0.10357 0.1167 0.09524 0.10000 0.10327 0.09226  
## Detection Rate 0.09196 0.1116 0.05714 0.06637 0.05804 0.03363  
## Detection Prevalence 0.10238 0.1515 0.06667 0.08036 0.06935 0.04018  
## Balanced Accuracy 0.93816 0.9557 0.79474 0.82407 0.77467 0.67865  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.93421 0.8464 0.6317 0.86420  
## Specificity 0.95157 0.9794 0.9511 0.92754  
## Pos Pred Value 0.65741 0.8249 0.5877 0.56000  
## Neg Pred Value 0.99317 0.9824 0.9590 0.98462  
## Prevalence 0.09048 0.1027 0.0994 0.09643  
## Detection Rate 0.08452 0.0869 0.0628 0.08333  
## Detection Prevalence 0.12857 0.1054 0.1068 0.14881  
## Balanced Accuracy 0.94289 0.9129 0.7914 0.89587

plot(as.data.frame(NB\_object$tables[1:3]),main = paste("Naive Bayes (fig. ",tc,")", sep = ""))



tc=tc + 1

### Support Vector Machines (SVM)

SVM are powerful and flexible supervised machine learning algorithms. They are used both for classification and regression. Generally, they are used in classification problems.

SVM <- svm(label~., data=train\_set, cost=100, scale=FALSE)  
paste("SVM Evaluation (fig. ",tc,")", sep = "")

## [1] "SVM Evaluation (fig. 13)"

tc=tc+1  
SVM

##   
## Call:  
## svm(formula = label ~ ., data = train\_set, cost = 100, scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 100   
##   
## Number of Support Vectors: 1406

SVM\_Pred <- predict(SVM,test\_set\_numonly, type="class")  
confusionMatrix(test\_set\_labels, SVM\_Pred)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 74 0 0 1 0 0 0 0 0 0  
## 1 0 84 0 1 0 0 0 0 0 0  
## 2 1 0 78 1 1 0 2 4 0 0  
## 3 0 1 2 83 0 4 1 1 0 4  
## 4 0 0 0 0 97 0 1 0 1 5  
## 5 0 0 0 5 0 58 4 0 1 2  
## 6 0 0 0 0 2 3 85 0 0 0  
## 7 0 0 2 1 0 1 0 73 0 1  
## 8 0 1 0 0 0 5 1 1 69 0  
## 9 1 0 1 1 4 0 0 3 0 68  
##   
## Overall Statistics  
##   
## Accuracy : 0.9155   
## 95% CI : (0.8946, 0.9334)  
## No Information Rate : 0.1238   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9059   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.97368 0.9767 0.93976 0.89247 0.9327 0.81690  
## Specificity 0.99869 0.9987 0.98811 0.98260 0.9905 0.98440  
## Pos Pred Value 0.98667 0.9882 0.89655 0.86458 0.9327 0.82857  
## Neg Pred Value 0.99739 0.9974 0.99336 0.98656 0.9905 0.98312  
## Prevalence 0.09048 0.1024 0.09881 0.11071 0.1238 0.08452  
## Detection Rate 0.08810 0.1000 0.09286 0.09881 0.1155 0.06905  
## Detection Prevalence 0.08929 0.1012 0.10357 0.11429 0.1238 0.08333  
## Balanced Accuracy 0.98619 0.9877 0.96394 0.93754 0.9616 0.90065  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.9043 0.89024 0.97183 0.85000  
## Specificity 0.9933 0.99340 0.98960 0.98684  
## Pos Pred Value 0.9444 0.93590 0.89610 0.87179  
## Neg Pred Value 0.9880 0.98819 0.99738 0.98425  
## Prevalence 0.1119 0.09762 0.08452 0.09524  
## Detection Rate 0.1012 0.08690 0.08214 0.08095  
## Detection Prevalence 0.1071 0.09286 0.09167 0.09286  
## Balanced Accuracy 0.9488 0.94182 0.98071 0.91842

### K-Nearest Neighbor (kNN)

kNN can be used for both regression and classification tasks. It assigns classification based on the proximity to the target variable.

k <- round(sqrt(nrow(train\_clean)))  
KNN\_Model <- knn(train=train\_set\_numonly, test=test\_set\_numonly,  
 cl=train\_set\_labels, k=k, prob = TRUE)  
paste("kNN Evaluation (fig. ",tc,")", sep = "")

## [1] "kNN Evaluation (fig. 14)"

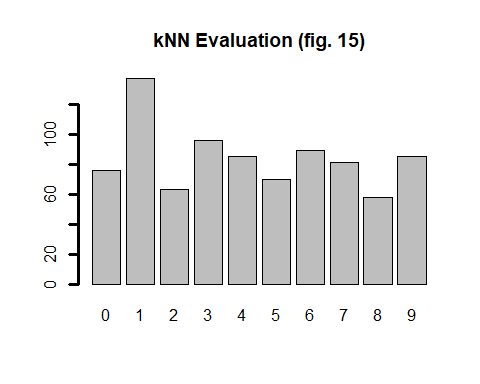
tc=tc+1  
head(KNN\_Model, n = 10L)

## [1] 8 7 4 9 3 4 7 3 6 2  
## Levels: 0 1 2 3 4 5 6 7 8 9

confusionMatrix(test\_set\_labels, KNN\_Model)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 69 0 0 0 0 3 3 0 0 0  
## 1 0 84 0 1 0 0 0 0 0 0  
## 2 3 12 62 3 1 0 0 5 1 0  
## 3 0 8 1 79 0 6 1 0 0 1  
## 4 0 6 0 0 80 0 1 2 1 14  
## 5 1 4 0 6 0 54 3 0 1 1  
## 6 0 4 0 0 1 4 81 0 0 0  
## 7 0 5 0 1 0 0 0 68 0 4  
## 8 0 11 0 5 1 3 0 1 55 1  
## 9 3 3 0 1 2 0 0 5 0 64  
##   
## Overall Statistics  
##   
## Accuracy : 0.8286   
## 95% CI : (0.8013, 0.8535)  
## No Information Rate : 0.1631   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8093   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.90789 0.6131 0.98413 0.82292 0.94118 0.77143  
## Specificity 0.99215 0.9986 0.96782 0.97715 0.96821 0.97922  
## Pos Pred Value 0.92000 0.9882 0.71264 0.82292 0.76923 0.77143  
## Neg Pred Value 0.99085 0.9298 0.99867 0.97715 0.99321 0.97922  
## Prevalence 0.09048 0.1631 0.07500 0.11429 0.10119 0.08333  
## Detection Rate 0.08214 0.1000 0.07381 0.09405 0.09524 0.06429  
## Detection Prevalence 0.08929 0.1012 0.10357 0.11429 0.12381 0.08333  
## Balanced Accuracy 0.95002 0.8059 0.97598 0.90003 0.95469 0.87532  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.91011 0.83951 0.94828 0.75294  
## Specificity 0.98802 0.98682 0.97187 0.98146  
## Pos Pred Value 0.90000 0.87179 0.71429 0.82051  
## Neg Pred Value 0.98933 0.98294 0.99607 0.97244  
## Prevalence 0.10595 0.09643 0.06905 0.10119  
## Detection Rate 0.09643 0.08095 0.06548 0.07619  
## Detection Prevalence 0.10714 0.09286 0.09167 0.09286  
## Balanced Accuracy 0.94906 0.91317 0.96007 0.86720

plot(KNN\_Model, lwd=3, main=paste("kNN Evaluation (fig. ",tc,")", sep = ""))



tc=tc+1

### Random Forest

Random Forest is used both in classification as well as in regression problems. This algorithm combines several machine learning algorithms (Decision trees) to obtain better accuracy.

paste("Random Forest Evaluation (fig. ",tc,")", sep = "")

## [1] "Random Forest Evaluation (fig. 16)"

tc=tc+1  
RF\_Model <- randomForest(label~.,train\_set)  
RF\_Model

##   
## Call:  
## randomForest(formula = label ~ ., data = train\_set)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 21  
##   
## OOB estimate of error rate: 6.28%  
## Confusion matrix:  
## 0 1 2 3 4 5 6 7 8 9 class.error  
## 0 342 0 1 0 0 0 1 0 4 0 0.01724138  
## 1 0 386 2 1 0 2 0 0 0 1 0.01530612  
## 2 3 2 301 0 4 1 3 5 1 0 0.05937500  
## 3 1 1 11 302 1 9 3 1 4 3 0.10119048  
## 4 0 2 0 0 328 1 2 3 4 7 0.05475504  
## 5 5 4 1 6 2 284 3 1 2 2 0.08387097  
## 6 4 2 2 0 1 4 290 0 1 0 0.04605263  
## 7 2 3 5 1 5 0 0 317 3 9 0.08115942  
## 8 0 3 0 5 4 3 3 1 307 8 0.08083832  
## 9 4 3 1 4 9 0 1 4 6 292 0.09876543

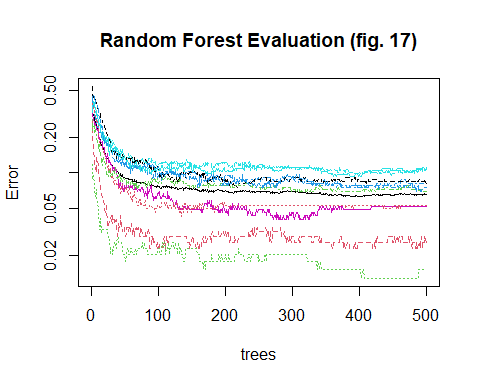
RF\_Pred <- predict(RF\_Model, test\_set)  
head(RF\_Pred, n = 10L)

## 3 6 14 22 47 50 53 57 58 61   
## 8 7 4 9 5 4 4 3 6 2   
## Levels: 0 1 2 3 4 5 6 7 8 9

confusionMatrix(test\_set$label, RF\_Pred)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 74 0 0 0 0 0 0 0 1 0  
## 1 0 81 1 1 0 0 0 2 0 0  
## 2 1 0 79 1 2 0 1 3 0 0  
## 3 0 1 3 85 0 3 1 2 0 1  
## 4 0 0 0 0 98 0 1 0 1 4  
## 5 0 0 2 4 1 60 2 0 1 0  
## 6 0 0 0 0 3 2 84 0 1 0  
## 7 1 1 4 0 0 0 0 66 0 6  
## 8 0 1 1 1 0 0 0 0 72 2  
## 9 1 0 1 2 4 2 0 2 1 65  
##   
## Overall Statistics  
##   
## Accuracy : 0.9095   
## 95% CI : (0.8881, 0.9281)  
## No Information Rate : 0.1286   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8993   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.96104 0.96429 0.86813 0.9043 0.9074 0.89552  
## Specificity 0.99869 0.99471 0.98932 0.9853 0.9918 0.98706  
## Pos Pred Value 0.98667 0.95294 0.90805 0.8854 0.9423 0.85714  
## Neg Pred Value 0.99608 0.99603 0.98406 0.9879 0.9864 0.99091  
## Prevalence 0.09167 0.10000 0.10833 0.1119 0.1286 0.07976  
## Detection Rate 0.08810 0.09643 0.09405 0.1012 0.1167 0.07143  
## Detection Prevalence 0.08929 0.10119 0.10357 0.1143 0.1238 0.08333  
## Balanced Accuracy 0.97986 0.97950 0.92873 0.9448 0.9496 0.94129  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.9438 0.88000 0.93506 0.83333  
## Specificity 0.9920 0.98431 0.99345 0.98294  
## Pos Pred Value 0.9333 0.84615 0.93506 0.83333  
## Neg Pred Value 0.9933 0.98819 0.99345 0.98294  
## Prevalence 0.1060 0.08929 0.09167 0.09286  
## Detection Rate 0.1000 0.07857 0.08571 0.07738  
## Detection Prevalence 0.1071 0.09286 0.09167 0.09286  
## Balanced Accuracy 0.9679 0.93216 0.96426 0.90814

plot(randomForest(label~.,train\_set, keep.forest=FALSE, ntree=500), log="y",  
 main=paste("Random Forest Evaluation (fig. ",tc,")", sep = ""))



tc=tc+1

## Results

All of these algorithms performed very well. The decision tree and Naïve Bayes performed the worst in terms of accuracy. Naïve Bayes and Random Forest performed the worst is processing speed. Those run slower based on how those predictive algorithms loop through the data to generate all the variations of answers. The others are basically faster because they are measuring distance to predict the answers.

### Decision tree

The high accuracy : *0.7485* along with the high Kappa score *0.7203*, Precision *0.747*, Recall *0.749*, and F-Measure *0.747* we conclude this model to be a reliable measure for classification.

### Naïve Bayes

The high accuracy : *0.7363* along with the high Kappa score *0.7067* and very low P-Value *2.2e-16* we conclude this model to be a reliable measure for classification.

### SVM

The high accuracy : *0.9155* along with the high Kappa score *0.9059* and very low P-Value *2.2e-16* we conclude this model to be a reliable measure for classification.

### kNN

The high accuracy : *0.8286* along with the high Kappa score *0.8093* and very low P-Value *2.2e-16* we conclude this model to be a reliable measure for classification.

### Random Forest

The high accuracy : *0.9095* along with the high Kappa score *0.8993* and very low P-Value *2.2e-16* we conclude this model to be a reliable measure for classification.

## Conclusion

### General:

To evaluate this handwriting recognition data set we had used various models. All the models performed well but the SVM seemed to have the best results overall. The benefit of the SVM is that you can capture much more complex relationships between your data points without having to perform difficult transformations on your own.

SVM performs well in higher dimension; In the real world there are infinite dimensions (and not just 2D and 3D). For instance image data, gene data, medical data etc. has higher dimensions and SVM is useful in that. Basically when the number of features/columns are higher, SVM does well. (Gupta, 2020)

This research assignment exemplifies the journey of a data scientist. It leverages the various models and examines their strengths and weaknesses. Although we concluded that the **SVM** algorithm performed the best there are other models that are not that far behind and would be reliable measures for classification.

##References

*Digit Recognizer | Kaggle.* (n.d.). Kaggle.com. Retrieved March 13, 2021 from <https://www.kaggle.com/c/digit-recognizer/overview>

Gupta, S. (2020, June 23). *Pros and cons of various Machine Learning algorithms* Medium. Retrieved March 13, 2021 from <https://towardsdatascience.com/pros-and-cons-of-various-classification-ml-algorithms-3b5bfb3c87d6>