Handwritten Digit Recognition Using Naive Bayes, Decision Tree, KNN, SVM and Random Forest

Juan Ventosa

8/26/2020

**INTRODUCTION:**

The Modified National Institute of Standards and Technology (MNIST) database is a collection of handwritten digits (0 through 9) that has been widely used for training and testing in the field of machine learning. The dataset has been made widely available to the public through online sites such as Kaggle.com allowing for anyone to analyze and train on machine learning classification techniques on the data.

The applications for classifying and correctly identifying handwritten images are numerous. Some examples include the ability to recognize handwritten amounts on checks that can now be deposited through an image submitted by an app and the ability to sort U.S. mail by scanning the handwritten postal zip codes on envelopes. The concept of identifying handwritten images can also be applied to classifying and identifying other images such as searching for images of cats on the web to facial recognition.

In the spirit of continuing the legacy of the MNIST database for machine learning an attempt to find the best machine learning algorithm with the highest accuracy to correctly classify the handwritten images will be made. Five machine learning algorithms will be tested: Naive Bayes, Decision Tree, k Nearest Neighbor (KNN), Support Vector Machine (SVM) and Random Forest. The National Institute of Standard and Technology (NIST) has set a high bar for these algorithms to replicate. In their original paper, "Gradient Based Learning Applied to Document Recognition” by LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998), the NIST authors used a support vector machine to get a 0.8% error rate.

Libraries

**ABOUT THE DATA:**

Load Data: The Kaggle digits data contains a train and test data set. The train data set contains 42000 records. Only a 25% sample of the train data set will be used to train and cross validate the models.

DigitTrain <- read.csv("~/Desktop/Kaggle-digit-train.csv", header = TRUE, stringsAsFactors = TRUE)  
DigitTrain$label <- as.factor(DigitTrain$label)  
dim(DigitTrain)

## [1] 42000 785

DigitTrain contains 42000 rows and 785 columns.

Each record represents a handwritten image of a digit made up of 28 x 28 pixels for a total of 784 pixels. There is an interger pixel value associated with each pixel ranging from 0 to 255 representing gradient shades of gray from lightest to darkest respectively. The first column “label” shows what number the pixel data represents. The remaining 784 columns represent the pixel position and their values.

EXPLORATORY DATA ANALYSIS:

Transform label variable so 0 = “zero”, 1 = “one, 2 =”two…, 9 =“nine”

DigitTrain$label=recode(DigitTrain$label, '0'="zero",'1'="one",'2'="two",'3'="three",  
 '4'="four",'5'="five",'6'="six",'7'="seven",'8'="eight", '9'="nine")  
  
summary(DigitTrain$label)

## zero one two three four five six seven eight nine   
## 4132 4684 4177 4351 4072 3795 4137 4401 4063 4188

str(DigitTrain$label)

## Factor w/ 10 levels "zero","one","two",..: 2 1 2 5 1 1 8 4 6 4 ...

A random 25% sample of the DigitTrain data set will be used for cross validation experiments for each of the models (Naive Bayes, Decision Tree, K Nearest Neighbor, Support Vector Machine and Random Forest).

# Take a random 25% sample of the MNIST Train data.  
set.seed(275)  
Split <- sample(nrow(DigitTrain),nrow(DigitTrain)\*0.25)  
TSample <- DigitTrain[Split,]  
# Reserve the remaining MNIST data as a final test for the best performing models  
UTest <- DigitTrain[-Split,]  
dim(TSample)

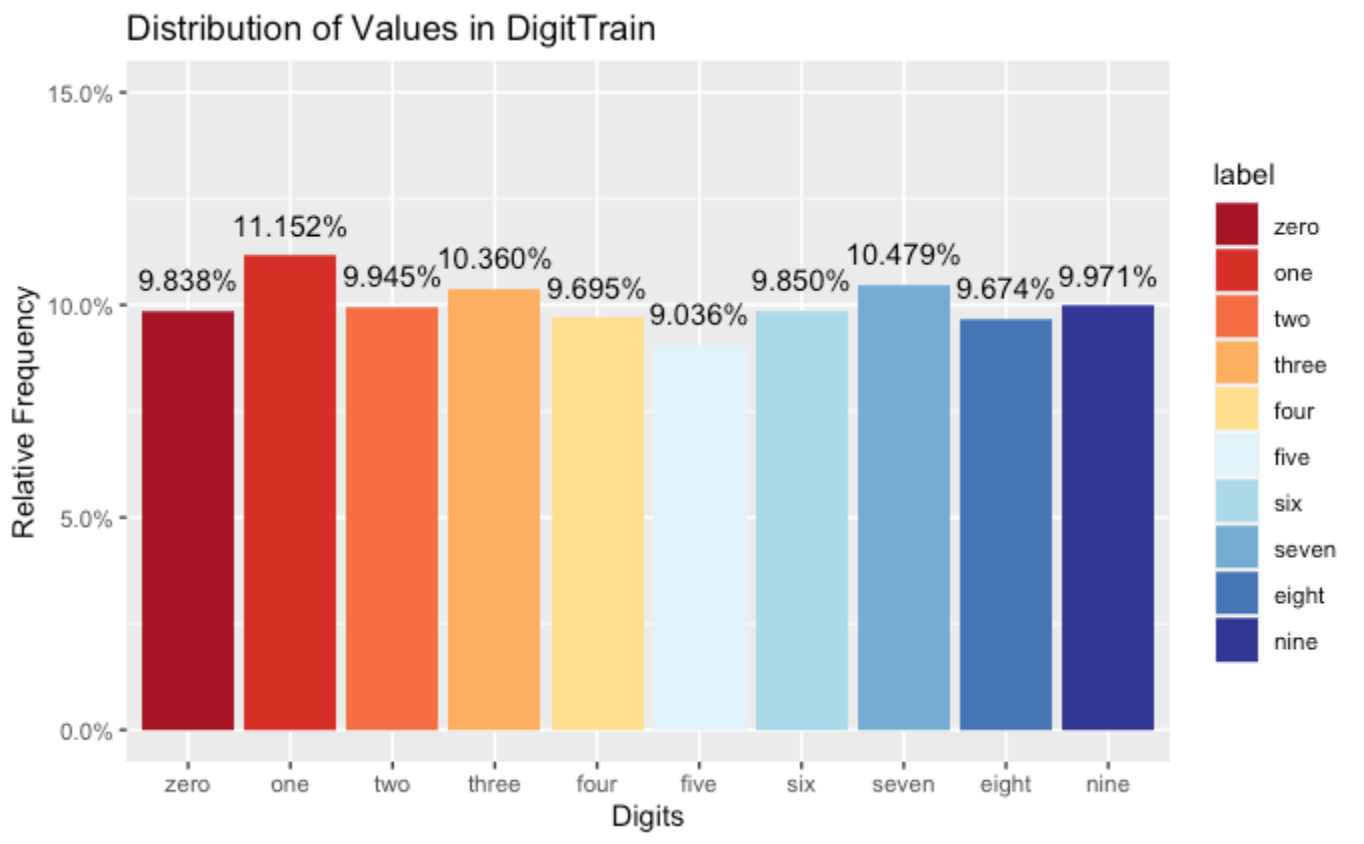
## [1] 10500 785

dim(UTest)

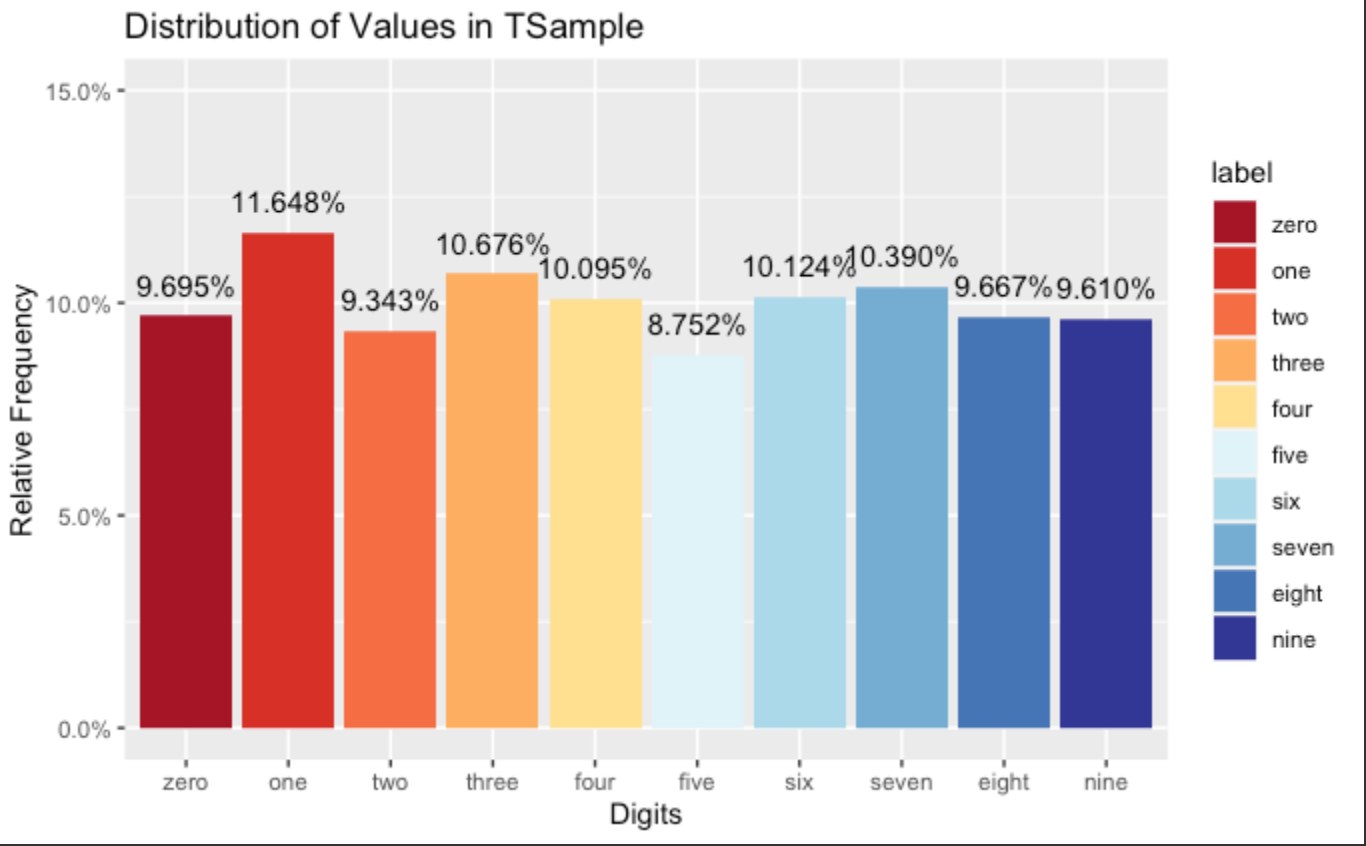
## [1] 31500 785

Compare distribution of values in the full DigitTrain data set with the 25% sample data set, TSample

plotall <- ggplot(DigitTrain, aes(x=label, y= (..count..)/sum(..count..),fill = label)) + geom\_bar() + labs(x="Digits", y="Relative Frequency", title = "Distribution of Values in DigitTrain") + scale\_y\_continuous(labels=scales::percent,limits = c(0,0.15)) +   
 geom\_text(stat="count", aes(label= scales::percent((..count..)/sum(..count..)),vjust=-1)) +  
 scale\_fill\_brewer(palette = "RdYlBu")  
  
plotsample <- ggplot(TSample, aes(x=label, y= (..count..)/sum(..count..),fill = label)) + geom\_bar() + labs(x="Digits", y="Relative Frequency", title = "Distribution of Values in TSample") + scale\_y\_continuous(labels=scales::percent,limits = c(0,0.15)) +   
 geom\_text(stat="count", aes(label= scales::percent((..count..)/sum(..count..)),vjust=-1)) +  
 scale\_fill\_brewer(palette = "RdYlBu")  
  
plotall



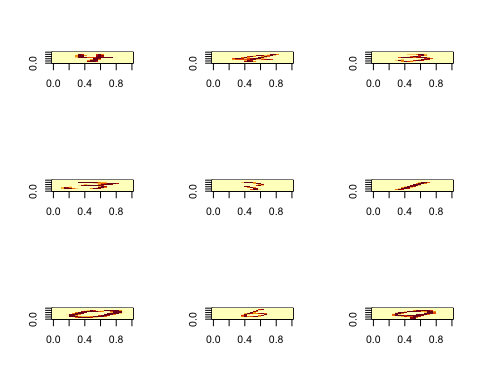
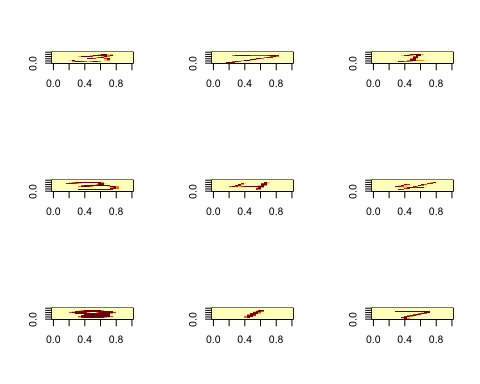
plotsample



The distribution of label variables in the full and sample data sets are acceptably comparable. The train sample (TSample) set will be used for cross validation experiments.

Examine sample digit images by plotting the data images using first 18 records in DigitTrain.

flip <- function(matrix) {  
 apply(matrix,2,rev)  
}  
  
par(mfrow=c(3,3))  
for (i in 1:18) {  
 digitimage <- flip(matrix(rev(as.numeric(TSample[i,-c(1,786)])),nrow = 28))  
 image(digitimage)  
}

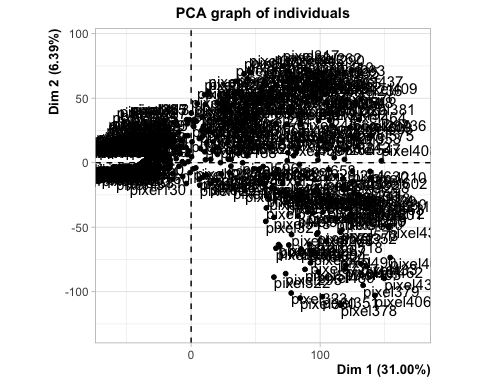


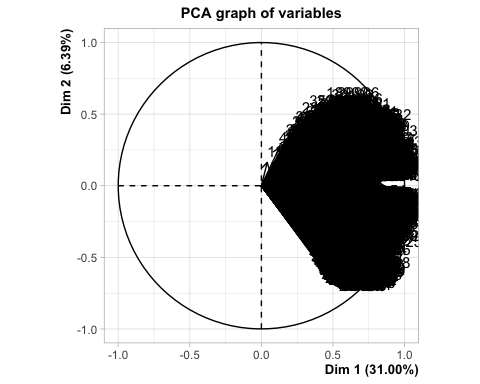
PRINCIPAL COMPONENT ANALYSIS

Principal Component Analysis (PCA) is a multivariate technique that allows us to summarize the systematic patterns of variations in the data. PCA is used for studying one table of observations and variables with the main idea of transforming the observed variables into a set of new variables, the principal components, which are uncorrelated and explain the variation in the data. PCA allows the reduction of a “complex” data set to a lower dimension in order to reveal the structures or the dominant types of variations in both the observations and the variables.

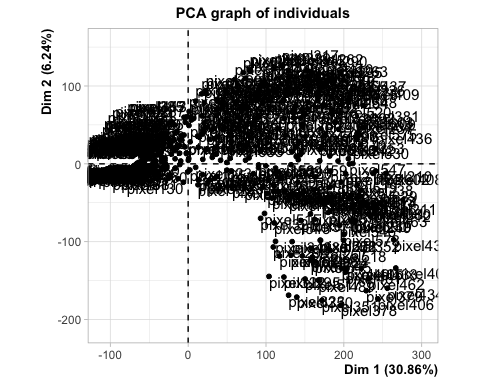
Using this technique on the MNIST data may be effective due to the large amount of “white space” defined by the pixel value 0. The PCA function in the FactoMineR package defaults to finding 5 dominant dimensions. A value of 196 pricipal components or 25% of the dimensions in the original data will be used for a reduced yet robust PCA sample.

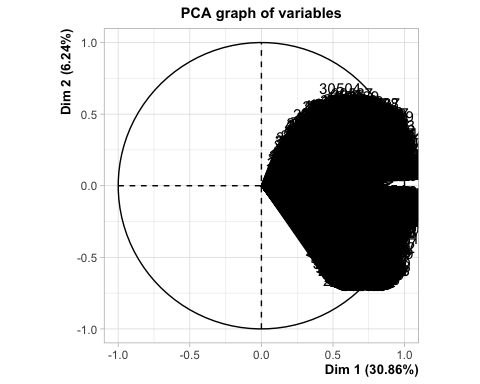
pcaSample1 = PCA(t(TSample[,-1]), ncp = 196)





pcaSample2 = PCA(t(UTest[,-1]), ncp = 196)





Combine the pcaSample dimension results with the labels to create pcaTrain and pcaUtest data.

pcaTrain <- data.frame(label = TSample$label, pcaSample1$var$coord)  
pcaUTest <- data.frame(label = UTest$label, pcaSample2$var$coord)

MODEL:

NAIVE BAYES

The first of five models to be tested on its accuracy to correctly predict the label in the MNIST data set is Naive Bayes. Naive Bayes is an algorithm that uses Bayes’ theorem to classify objects. Naive Bayes classifiers assume strong, or naive, independence between attributes of data points.

Perform a 10 fold holdout cross validation test on both the TSample and pcaTrain data sets.

Create a function to perform holdout cross validation on naive Bayes

# Create result bins for holdoutcv function  
AllResultest <- list()  
AllLabelstest <- list()  
  
NBholdoutcv <- function (data, N, kfolds) {  
 holdout <- split(sample(1:N),1:kfolds)  
 for (k in 1:kfolds) {  
   
 test= data[holdout[[k]],]  
 train = data[-holdout[[k]],]  
  
# Make sure to remove Response from the testing data  
 test\_response <- test$label  
 test\_no\_response <- test[,-1]  
  
# {naivebayes} package  
 model <- naive\_bayes(label~., data = train, na.action = na.pass,   
 laplace = 1)  
 predtest <- predict(model,test\_no\_response)  
   
 # Accumulate results from each fold  
 AllResultest <- c(AllResultest, predtest)  
 AllLabelstest <- c(AllLabelstest, test\_response)  
 ConfMatrix <- table(Actual = unlist(AllLabelstest), Prediction = unlist(AllResultest))}  
 return(ConfMatrix)  
}

Run the NBholdoutcv function on the pcaTrain data and record the accuracy.

# Number of observations  
Np <- nrow(pcaTrain)  
# Number of desired splits  
kfolds <- 10  
# Perform holdout cross validation using NBholdoutcv  
set.seed(123)  
NBholdoutcv(pcaTrain,Np,kfolds)

## Prediction  
## Actual 1 2 3 4 5 6 7 8 9 10  
## 1 885 0 38 3 0 83 7 0 2 0  
## 2 0 1098 57 5 18 17 3 3 19 3  
## 3 21 15 824 35 9 13 13 6 40 5  
## 4 7 10 55 868 1 114 3 13 28 22  
## 5 2 3 52 2 844 23 5 1 6 122  
## 6 53 2 21 92 4 686 13 0 28 20  
## 7 23 6 69 3 5 41 913 0 3 0  
## 8 4 16 54 7 37 38 0 876 7 52  
## 9 23 32 72 25 4 89 5 2 729 34  
## 10 12 2 62 10 50 56 1 18 9 789

Repeat NBholdoutcv function on the TSample data and record the accuracy.

# Number of observations  
Nt <- nrow(TSample)  
# Number of desired splits  
kfolds <- 10  
# Perform holdout cross validation using NBholdoutcv  
set.seed(123)  
NBholdoutcv(TSample,Nt,kfolds)

## Prediction  
## Actual 1 2 3 4 5 6 7 8 9 10  
## 1 966 1 1 0 0 2 23 0 13 12  
## 2 2 1198 0 2 1 0 5 0 9 6  
## 3 186 53 164 72 6 13 293 3 162 29  
## 4 192 133 6 329 1 2 79 11 251 117  
## 5 50 21 1 2 101 8 130 3 125 619  
## 6 270 63 5 12 6 60 74 0 323 106  
## 7 21 46 3 0 1 10 968 0 10 4  
## 8 19 53 1 5 3 1 11 316 35 647  
## 9 40 254 2 11 2 8 44 1 467 186  
## 10 19 40 2 2 4 0 3 8 14 917

The Naive Bayes algorithm had higher prediction accuracy on the train data that was transformed by principal component analysis (pcaTrain) versus the original sample data set (TSample). Naive Bayes had an 81.1% accuracy at correctly predicting the labels on the pcaTrain. A significant improvement over a 52.2% accuracy on the original sample data set, Tsample.

Utilize caret package to find optimal fit of Naive Bayes algorithm

library(caret)  
# Lookup package model tuning options for optimal fit  
modelLookup(model = 'nb')

## model parameter label forReg forClass probModel  
## 1 nb fL Laplace Correction FALSE TRUE TRUE  
## 2 nb usekernel Distribution Type FALSE TRUE TRUE  
## 3 nb adjust Bandwidth Adjustment FALSE TRUE TRUE

# Train the model for best fit  
set.seed(123)  
NBbest <- train(label~., data=pcaTrain, method = "naive\_bayes",  
 trControl = trainControl("cv", number = 10),  
 tuneLength = 10)  
  
NBbest$results

## usekernel laplace adjust Accuracy Kappa AccuracySD KappaSD  
## 1 FALSE 0 1 0.8109524 0.7899654 0.01642113 0.01820695  
## 2 TRUE 0 1 0.8095250 0.7883419 0.01657253 0.01838484

NBbest$bestTune

## laplace usekernel adjust  
## 1 0 FALSE 1

The caret package 10 fold cross validation results suggest minor accuracy improvement if model used best tune of usekernel = FALSE, laplace = 0 and adjust = 1. The tuning changes did not have much of an accuracy improvement on default settings of the naive\_bayes algorithm.

DECISION TREE

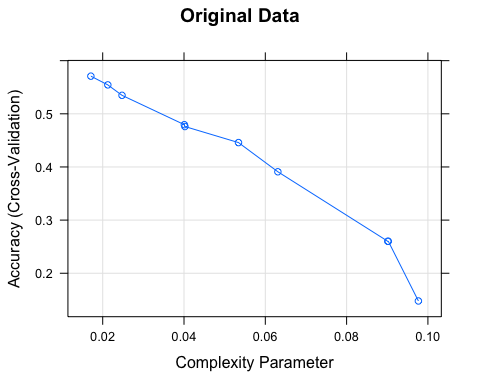
A tree model in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target feature or the arc leads to a subordinate decision node on a different input feature.

The caret package will be utilized to execute a 10 fold cross validation to find the optimum pruning of the tree for best fit.

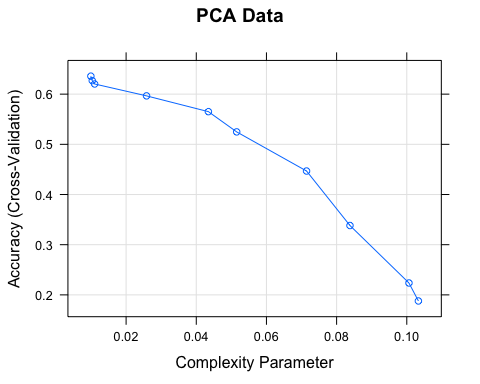
# Split the TSample and pcaTrain data into train and test data sets at 80/20 split  
set.seed(275)  
TSsampledt <- sample(nrow(TSample),nrow(TSample)\*0.80)  
TSDTtrain <- TSample[TSsampledt,]  
TSDTtest <- TSample[-TSsampledt,]  
  
set.seed(275)  
pcasampledt <- sample(nrow(pcaTrain),nrow(pcaTrain)\*0.80)  
pcaDTtrain <- pcaTrain[pcasampledt,]  
pcaDTtest <- pcaTrain[-pcasampledt,]  
  
# Lookup caret model tuning options for rpart decision tree   
modelLookup(model = "rpart")

## model parameter label forReg forClass probModel  
## 1 rpart cp Complexity Parameter TRUE TRUE TRUE

# Train model for best fit using 10 fold cross validation  
set.seed(123)  
TSDTmodel <- train(label~., data = TSDTtrain , method = "rpart",  
 trControl = trainControl("cv", number = 10),  
 tuneLength = 10)  
  
set.seed(123)  
pcaDTmodel <- train(label~., data = pcaDTtrain , method = "rpart",  
 trControl = trainControl("cv", number = 10),  
 tuneLength = 10)  
  
# Plot model accuracy vs different values of cp  
plot(TSDTmodel, main = "Original Data")



plot(pcaDTmodel, main = "PCA Data")

 At 63.58% accuracy with complexity parameter (cp) = 0.00995, the transformed principal component sample data provided better model training results than the best tune for the original data at cp = 0.01708 and 57.08% accuracy.

# TSDTmodel results  
head(TSDTmodel$results,3)

## cp Accuracy Kappa AccuracySD KappaSD  
## 1 0.01708367 0.5708416 0.5215290 0.02010254 0.02244804  
## 2 0.02125370 0.5544110 0.5031251 0.01586805 0.01775435  
## 3 0.02475114 0.5347725 0.4809546 0.02398977 0.02687471

# pcaDTmodel results  
head(pcaDTmodel$results, 3)

## cp Accuracy Kappa AccuracySD KappaSD  
## 1 0.009954264 0.6358333 0.5947656 0.01504422 0.01681986  
## 2 0.010357815 0.6269088 0.5847170 0.02022586 0.02258871  
## 3 0.011030401 0.6202428 0.5771953 0.02189770 0.02440493

Use final pcaDTmodel to predict the pcaDTtest and use confusion matrix to record accuracy.

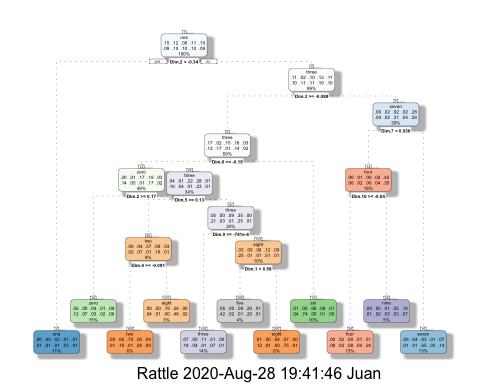
# Execute model to predict label on test data  
pcaDTprediction <- predict(pcaDTmodel, pcaDTtest)  
  
# Confusion matrix of prediction results against actuals  
table(Actual = pcaDTtest$label, Prediction = pcaDTprediction)

## Prediction  
## Actual zero one two three four five six seven eight nine  
## zero 168 0 1 24 0 0 3 0 0 0  
## one 0 218 9 0 2 0 7 18 3 0  
## two 20 4 93 20 3 13 12 8 17 4  
## three 3 1 1 127 2 19 13 3 28 6  
## four 18 1 5 2 130 1 2 19 0 19  
## five 41 1 1 57 20 26 8 2 33 2  
## six 14 5 17 10 6 1 145 1 0 1  
## seven 9 3 0 1 19 2 0 191 2 2  
## eight 4 7 3 21 13 25 1 13 114 4  
## nine 17 2 1 3 80 1 1 35 4 84

The model correctly predicted the label at 61.71% accuracy, which is in line with the cross validation results on the PCA transformed train data accuracy of 63.58% indicating minimal over fitting on the model.

Plot the final decision tree model.

# Plot the final Decision Tree model  
fancyRpartPlot(pcaDTmodel$finalModel)



K NEAREST NEIGHBORS (KNN)

KNN has been used in statistical estimation and pattern recognition as a non-parametric technique. It estimates how likely a data point is to be a member of one group or the other depending on the data points nearest proximity to a specified number (k) groups. The k-nearest-neighbor is an example of a “lazy learner” algorithm, meaning that it does not build a model using the training set until a query of the data set is performed.

Utilize the caret package to execute a knn 10 fold cross validation using the PCA transformed sample data (pcaTrain)

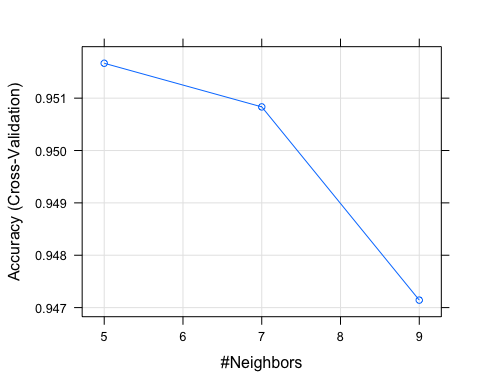
# Split the pcaTrain data into train and test using an 80/20 split  
set.seed(275)  
pcasampleknn <- sample(nrow(pcaTrain),nrow(pcaTrain)\*0.80)  
pcaknntrain <- pcaTrain[pcasampleknn,]  
pcaknntest <- pcaTrain[-pcasampleknn,]  
  
# Lookup caret parameter tuning options for knn  
modelLookup(model="knn")

## model parameter label forReg forClass probModel  
## 1 knn k #Neighbors TRUE TRUE TRUE

# Train the model using caret 10 fold cross validation to find best fit  
set.seed(123)  
pcaknnmodel <- train(label~., data = pcaknntrain, method = 'knn',  
 trControl = trainControl("cv", number = 10))  
  
# Call cross validation results  
pcaknnmodel$results

## k Accuracy Kappa AccuracySD KappaSD  
## 1 5 0.9516664 0.9462548 0.008774459 0.009758551  
## 2 7 0.9508329 0.9453272 0.009789190 0.010887481  
## 3 9 0.9471427 0.9412217 0.009280694 0.010323920

# plot accuracy parameter results  
plot(pcaknnmodel)



Caret 10 fold cross validation on the PCA transformed train sample data resulted in a knn model with a 95.17% accuracy at correctly predicting the labels with k = 5.

Run the best tune final model based on the cross validation results on the test data and calculate the accuracy using a confusion matrix.

# Predict the test data using best tune final knn model  
set.seed(123)  
pcaknnprediction <- predict(pcaknnmodel, pcaknntest[,-1])  
  
# Confusion matrix of results  
table(Actual = pcaknntest$label, Prediction = pcaknnprediction)

## Prediction  
## Actual zero one two three four five six seven eight nine  
## zero 193 0 0 0 0 0 2 0 1 0  
## one 0 256 0 1 0 0 0 0 0 0  
## two 7 0 181 2 0 0 0 3 1 0  
## three 0 0 0 193 0 3 0 2 2 3  
## four 1 2 0 0 183 0 1 1 1 8  
## five 1 0 1 5 0 181 0 2 1 0  
## six 0 0 0 0 0 0 200 0 0 0  
## seven 0 2 3 0 0 0 0 220 0 4  
## eight 2 2 0 6 2 2 2 2 186 1  
## nine 3 0 0 5 3 0 0 4 1 212

The knn model accurately predicted the test label correctly with 95.48% accuracy with fairly high precision on all labels. The accuracy was also comparable to the cross validation results on the train data indicating minimal over fitting of the model.

SUPPORT VECTOR MACHINE

A discriminative classifier formally defined by a separating hyperplane. The algorithm creates a line or a hyperplane, which separates the data into classes.

Utilize caret package cross validation function to find best model for SVM Linear using the MNIST data transformed by principal component analysis.

# Split the TSample and pcaTrain data into train and test data sets at 80/20 split  
set.seed(275)  
pcasamplesvm <- sample(nrow(pcaTrain),nrow(pcaTrain)\*0.80)  
pcaSVMtrain <- pcaTrain[pcasamplesvm,]  
pcaSVMtest <- pcaTrain[-pcasamplesvm,]  
  
# Lookup caret model tuning options for rpart decision tree   
modelLookup(model = "svmLinear")

## model parameter label forReg forClass probModel  
## 1 svmLinear C Cost TRUE TRUE TRUE

# Train model for best fit using 10 fold cross validation  
set.seed(123)  
pcaSVMlinear <- train(label~., data = pcaSVMtrain , method = "svmLinear",  
 trControl = trainControl("cv", number = 10),  
 tuneLength = 10)  
  
pcaSVMlinear$results

## C Accuracy Kappa AccuracySD KappaSD  
## 1 1 0.8898813 0.8775395 0.009831848 0.01093142

Cross validation results for SVM model on PCA transformed data resulted in an 88.99% accuracy with the best model parameter of cost = 1.

# Use pcaSVMlinear final model to predict test data  
pcaSVMprediction <- predict(pcaSVMlinear, pcaSVMtest)  
  
# Confusion Matrix to calculate accuracy on model on test data  
table(Actual = pcaSVMtest$label, Prediction = pcaSVMprediction)

## Prediction  
## Actual zero one two three four five six seven eight nine  
## zero 190 0 0 0 1 1 3 0 1 0  
## one 0 254 0 1 1 0 0 0 1 0  
## two 7 2 163 3 8 0 7 2 1 1  
## three 1 2 10 168 1 14 0 3 4 0  
## four 0 1 2 0 180 0 2 2 1 9  
## five 0 1 2 15 3 155 6 0 6 3  
## six 2 0 1 1 1 4 188 0 3 0  
## seven 1 2 3 0 3 0 0 212 1 7  
## eight 2 5 6 12 5 8 4 1 160 2  
## nine 3 0 2 5 20 2 1 8 2 185

The SVM model using the PCA transformed train sample data had an 88.33% accuracy which is a minimal difference from the pcaTrain cross validation accuracy result of 88.99% suggesting minimal over fitting.

Due to an inablity to scale constant variables, the caret package was unable to execute an SVM cross validation on the original data sample (TSample) for comparison. Instead a 10 fold holdout cross validation function will be created and performed on both the original data sample and the PCA transformed sample data for comparison.

Create a function to perform a 10 fold holdout cross validation evaluations of SVM models.

library(e1071)  
# Create result bins for SVMholdoutcv function  
AllResultest <- list()  
AllLabelstest <- list()  
  
SVMholdoutcv <- function (data, N, kfolds) {  
 holdout <- split(sample(1:N),1:kfolds)  
 for (k in 1:kfolds) {  
   
 train= data[holdout[[k]],]  
 test = data[-holdout[[k]],]  
 test\_label <- test$label  
  
# {e1071} package  
 model <- svm(label~., data = train, kernel = "linear", cost = 1)  
 predtest <- predict(model,newdata = test, type=c("class"))  
   
 # Accumulate results from each fold  
 AllResultest <- c(AllResultest, predtest)  
 AllLabelstest <- c(AllLabelstest, test\_label)  
 ConfMatrix <- table(Actual = unlist(AllLabelstest), Prediction = unlist(AllResultest))}  
 return(ConfMatrix)  
}

Execute 10 fold cross validation SVM evaluation on the original sample data (TSample) and calculate the accuracy through a confusion matrix.

# Set up an experimental evaluation.  
# Number of observations  
Nt <- nrow(TSample)  
# Number of desired splits  
kfolds <- 10  
# Perform SVM holdout cross validation using SVMholdoutcv function  
set.seed(123)  
SVMholdoutcv(TSample, Nt, kfolds)

## Prediction  
## Actual 1 2 3 4 5 6 7 8 9 10  
## 1 8763 0 55 16 22 117 93 17 48 31  
## 2 0 10781 27 45 20 25 7 19 66 17  
## 3 123 85 7471 175 205 62 263 173 218 54  
## 4 28 82 262 8551 27 522 55 104 312 146  
## 5 32 52 79 6 8610 32 83 52 36 558  
## 6 109 105 55 534 133 6763 201 47 199 125  
## 7 106 36 173 6 99 154 8931 5 56 1  
## 8 32 120 159 74 165 35 9 8781 36 408  
## 9 54 266 199 394 65 408 131 61 7361 196  
## 10 73 47 40 124 552 61 16 380 91 7697

Execute 10 fold cross validation SVM evaluation on the sample data transformed by pricipal component analysis (pcaTrain)

# Set up an experimental evaluation.  
# Number of observations  
Np <- nrow(pcaTrain)  
# Number of desired splits  
kfolds <- 10  
# Perform SVM holdout cross validation using SVMholdoutcv function  
set.seed(123)  
SVMholdoutcv(pcaTrain, Np, kfolds)

## Prediction  
## Actual 1 2 3 4 5 6 7 8 9 10  
## 1 8435 8 113 65 32 181 146 35 106 41  
## 2 2 10591 74 53 34 49 20 20 148 16  
## 3 177 217 6690 432 211 118 313 196 379 96  
## 4 80 160 478 7634 77 716 93 162 378 311  
## 5 53 71 142 74 7703 151 122 100 171 953  
## 6 238 104 134 712 221 5681 318 134 501 228  
## 7 194 74 193 36 147 238 8503 14 147 21  
## 8 96 162 189 181 269 61 24 8086 101 650  
## 9 104 446 313 438 218 544 185 100 6396 391  
## 10 101 56 70 212 765 110 22 547 158 7040

In the case of the 10 fold holdout cross validation the Support Vector Machine (SVM) performed better on the original sample data with an 88.58% accuracy at predicting the label correctly. The SVM model was less accurate at predicting the correct label on the data transformed by principal component analysis with an 81.26% accuracy. The results on the PCA data using the caret cross validation function against the holdout cross validation provided fairly different accuracies with 88.99% and 81.26% respectively. However, the overall results for the SVM model have fallen more frequently in the 88% accuracy range.

RANDOM FOREST

An ensemble classifier that consists of many decision trees and outputs the class that is the mode of the classes output by individual trees.

Once again the caret package will be utilized to execute a random forest 10 fold cross validation on both the original sample data and the PCA transformed sample data.

Create a function to perform holdout cross validation evaluations of Random Forest models. The function will include an ntree and mtry designation.

library(randomForest)  
# Create result bins for RFholdoutcv function  
AllResultest <- list()  
AllLabelstest <- list()  
  
RFholdoutcv <- function (data, N, kfolds, treesize) {  
 holdout <- split(sample(1:N),1:kfolds)  
 for (k in 1:kfolds) {  
   
 train= data[holdout[[k]],]  
 test = data[-holdout[[k]],]  
 test\_label <- test$label  
  
# {e1071} package  
 model <- randomForest(label~., data = train, tree = treesize)  
 predtest <- predict(model,test, type=c("class"))  
   
 # Accumulate results from each fold  
 AllResultest <- c(AllResultest, predtest)  
 AllLabelstest <- c(AllLabelstest, test\_label)  
 ConfMatrix <- table(Actual = unlist(AllLabelstest), Prediction = unlist(AllResultest))}  
 return(ConfMatrix)  
}

Execute 10 fold cross validation random forest evaluation on the original sample data (TSample) with tree = 500 and calculate the accuracy through a confusion matrix.

# Set up an experimental evaluation.  
# Number of observations  
Nt <- nrow(TSample)  
# Number of desired splits  
kfolds <- 10  
# Perform SVM holdout cross validation using SVMholdoutcv function  
# Note: default mtry is calculated as rounded total predictor variables/3 = 196/3 = 65  
set.seed(123)  
RFholdoutcv(TSample, Nt, kfolds, 500)

## Prediction  
## Actual 1 2 3 4 5 6 7 8 9 10  
## 1 8876 1 26 9 11 29 100 11 98 1  
## 2 0 10842 15 38 15 6 10 28 35 18  
## 3 133 94 7781 104 133 30 189 217 125 23  
## 4 48 136 171 8821 51 301 39 134 218 170  
## 5 39 62 41 10 8741 5 97 23 52 470  
## 6 144 126 13 569 113 6879 147 43 108 129  
## 7 147 76 60 1 91 107 9047 11 27 0  
## 8 49 155 164 32 153 11 0 8889 53 313  
## 9 37 224 86 379 54 139 139 34 7821 222  
## 10 88 58 35 145 441 63 43 293 70 7845

Execute 10 fold cross validation random forest evaluation on the PCA transformed sample data (pcaTrain) with tree = 500 and calculate the accuracy through a confusion matrix.

# Set up an experimental evaluation.  
# Number of observations  
Nt <- nrow(TSample)  
# Number of desired splits  
kfolds <- 10  
# Perform SVM holdout cross validation using RFholdoutcv function  
# Note: default mtry is calculated as rounded total predictor variables/3 = 196/3 = 65  
set.seed(123)  
RFholdoutcv(pcaTrain, Nt, kfolds, 500)

## Prediction  
## Actual 1 2 3 4 5 6 7 8 9 10  
## 1 8756 0 46 69 14 48 149 0 78 2  
## 2 0 10786 39 51 21 6 43 16 33 12  
## 3 176 142 7259 255 164 27 302 199 290 15  
## 4 50 68 237 8787 43 218 152 124 285 125  
## 5 27 69 139 46 8265 14 222 45 57 656  
## 6 254 55 87 975 192 5940 297 94 249 128  
## 7 200 69 132 36 59 111 8912 2 45 1  
## 8 89 164 169 39 233 25 15 8741 39 305  
## 9 78 200 170 702 105 273 113 130 7106 258  
## 10 201 44 80 183 936 44 34 568 106 6885

The Random Forest models favored training on the original sample data with 784 predictor variables or dimensions. It had a 90.5% accuracy at correctly classifying the handwritten image. By comparison the random forest model trained on the sample data transformed by PCA with 196 predictor variables had an 86.2% accuracy.

Overall the KNN model trained on the sample data transformed by PCA had the highest accuracy and precision results. As a final test, the best KNN model will be used to predict the remaining 75% of the MNIST train data that was excluded from the sample data.

# Use the best KNN model to predict the 75% MNIST train data.  
KNNfinal <- predict(pcaknnmodel, pcaUTest[,-1])  
  
# Print results in a confusion matrix  
table(Actual = pcaUTest$label, Prediction = KNNfinal)

## Prediction  
## Actual zero one two three four five six seven eight nine  
## zero 3073 1 4 4 2 0 17 2 10 1  
## one 0 3424 18 7 0 0 5 2 3 2  
## two 43 41 2891 41 32 0 21 60 52 15  
## three 14 18 29 2983 4 49 5 22 85 21  
## four 5 32 1 0 2659 1 39 6 4 265  
## five 36 15 0 119 12 2527 63 4 37 63  
## six 41 14 1 0 5 8 2999 0 6 0  
## seven 8 62 9 2 13 0 0 3130 7 79  
## eight 21 65 5 176 36 34 23 9 2619 60  
## nine 22 37 5 43 85 8 0 85 34 2860

The KNN model’s result on the final test had a 92.7% accuracy. This was slightly less than the cross validation results on the train data which had a 95.2% accuracy.

RESULTS:

All models were trained on a 25% sample of the original MNIST train database along with the same sample data set transformed by principal component analysis using 196 principal components. The models were all subjected to a 10 fold cross validation to measure their classification accuracy. Only the best model results by algorithm will be covered. The algorithms will be discussed in the order of their performance from worst to best accuracy at correctly classifying the handwritten digits.

The best Decision Tree model was trained on the sample data transformed by principal component analysis (PCA). Utilizing the caret package to run a 10 fold cross validation on the rpart algorithm with 10 tuning values used for the complexity parameter (cp), the best fit model used cp = 0.00995 for a nearly full grown tree. The use of PCA on the training data increased the accuracy from 57.1% without PCA to 63.6% with PCA. The precision at correctly classifying the digits were also low ranging from 29.6% accuracy on class ‘5’ to 90.1% accuracy on class ‘7’ with a mean precision of 61.0% for all classes combined.

The best Naive Bayes model was also trained on the sample data transformed by PCA. The best results were based on the naive\_bayes algorithm with parameters set to laplace = 0, usekernel = FALSE and adjust =1. The use of PCA on the training data significantly increased the model’s accuracy from 52.2% to 81.1% at predicting the handwritten digits correctly. The precision at correctly identifying the digits ranged from a low of 59.1% for class ‘5’ to 95.3% for class ‘7’ with a mean precision of 81.9% for all classes combined.

The most accurate Support Vector Machine model was trained on the sample data without PCA transformation. The best model used was a linear classifier with a cost parameter = 1. It had an accuracy of 88.58% based on a 10 fold holdout cross validation. The best model trained on the sample data transformed by PCA performed only slightly less accurate at 88.33%. The difference is so minimal that distinguishing one to have performed better may be trivial. The precision for correctly classifying the digits were also close with the mean precision for the model trained on the original sample data at 88.4% accurate with a min = 83.4% and max = to 94.0% vs. the mean precision of the model trained on the sample data transformed by PCA at 88.1% with min = 80.7% and max = 95.3%.

The Random Forest models had higher cross validation accuracies when trained on the original sample data compared to the model trained on the sample data transformed by PCA. Both models were placed through a 10 fold holdout cross validation with tree size set to 500. The best model had an accuracy of 90.5% at correctly classifying the the digits with class precisions ranging from 85.4% to 92.8%. The cross validation results on the model trained on the sample data transformed by PCA had an accuracy of 86.2% with class precisions ranging from 78.9% to 93.0%.

The most accurate k-nearest neighbor (KNN) model was trained on the sample data transformed by PCA. The caret package 10 fold cross validation function with 10 values for k on the knn3 algorithm resulted in a best fit model with k = 5. The model had an accuracy of 95.2% at correctly classifying the digits. It should also be noted that KNN performed well with k= 5 through 9 with accuracies for those values falling between 94.7% to 95.2%. The precision for the best fit KNN model on correctly classifying the digit was also very accurate with a mean precision of 95.5% with a range of 91.0% to 97.8%.

As reminder all the models were trained on a random 25% sample of the MNIST train data. With the highest accuracy and precision of all the 5 algorithms tested, the final best KNN model was placed through the final test of classifying the remaining 75% of the MNIST train data. In this final test the KNN model accurately classified the handwritten images correctly 92.7% of the time.

Summary of Best Model Accuracy Results:

* Decision Tree = 63.6%
* Naive Bayes = 81.9%
* Support Vector Machine = 88.4%
* Random Forest = 90.5%
* K-Nearest Neighbors = 95.2%

CONCLUSION:

Of the five algorithms tested Decision Tree had the least overall accuracy at only 63.6%. Naive Bayes and Support Vector Machine models both performed adequately well with results that correctly identified the handwritten image 82% to 88% of the time. The Random Forest model provided slightly higher accuracies with the ability to identify the image 90% of the time. Lastly the KNN model achieved the best accuracy results correctly identifying the image 95% of the time.

Although the perfomance of the algorithms varied in accuracy, each had their advantages and disadvantages. The Decision Tree and Naive Bayes models may have had the worse accuracy performance, but had the advantage of running fast classification models that required little tuning. The Decision Tree also has the added advantage of providing a guide on how its algorithm came to its classifications. SVM and Random Forest have been known to provide high accuracies in their classification results along with high precision and recall. However, the models may require more time to calculate their results. KNN models can be very effective at classifying models primed for clusters. In this case KNN proved to be both effective and fast, having classified the hand written images in nearly half the time taken by Random Forest with a higher accuracy and precision.

In the end the model of choice may depend on the needs of the user. Will providing results quickly be more economically feasible or is accuracy more important than speed? The precision and recall of the models should also be considered since overall accuracy may not be as important a factor as having a high precision on a particular class prediction. It should also be noted that if accuracy trumps speed, all of these models may be tuned to possibly perform better. For example the random forest model cross validation tests used here only attempted the use of 500 trees. Increasing the trees to 1000 or 20000 may change results and tweaking other parameters such as mtry which indicates the number of tree variations may also lead to more accurate predictions. The KNN model parameters used here may have ultimately provided the best results but other attempts by other users would likely change based on such factors as transforming the train data to give algorithms an edge or by tweaking the algorithms themselves. Although random forest and KNN performed best here, it should be noted that NIST achieved a 99.2% accuracy at classifying the same images using an SVM model. In the end choosing the best algorithm for the job may depend on the data and may require multiple experiments and attempts to optimize results.