2020

Introduction to Machine Learning

Module number: DALT7011

Student Number: 19132761

MSc Course: MSc in Data Analytics

Word count: 1380

Content

Introduction

Machine learning refers to applying artificial intelligence to enable systems to learn and improve without the essence of being programmed. The programs access the data and make accurate predictions without the programmer coding to tell the program what to do. The AI trains the algorithm to identify features and patterns in vast amounts of data in order to make decisions.

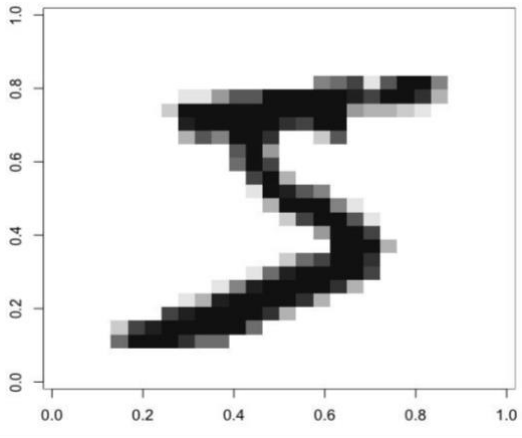
Classification is a central problem that is used to group data based on a particular condition. It can be done based on predetermined characteristics, which is a type of supervised learning. Classification plays an essential role in today's big data world, which classifies data based on similar features and attributes. The computer program uses the data fed into and learns using the algorithm to specify the classes to which the various data elements belongs to or make new observations.

The data should be separated into training and test models to evaluate the models that have been applied to the given data set. The use of similar data for training and test can reduce the effects of data discrepancies and better understand the characteristics of the data model. When it is split, most of the data is used for training, and a small portion of data is used for testing. When the training set processes a model, the test set is used to make predictions by applying it. It is easy to determine whether the model performs well by making predictions against the test data.

The MNIST ("Modified National Institute of Standards and Technology") is the "hello world" dataset created for computer vision. It was released in 1999, and it consists of handwritten images that form the basis for classification problems. This dataset consists of binary images of handwritten digits. The MNIST comprise of 10000 testing images, and 60000 training images, halves of the testing and the training sets were taken from the NIST's testing dataset while the other halves were taken from MNIST's training dataset. This well-crafted merge of datasets from two sources came about since initially the NIST's training dataset was obtained from American Census Bureau while the American high school students gave the testing data, and this combination was not reliable in many machine learning experiments.

Dataset Information

Let look at the first image and see how the pictures look like:



The First Image

The k-nearest neighbours (kNN) algorithm does not make assumptions about the distribution of the data. Thus, we need to use Normalizing to make all the numeric columns' values to a standard scale in the dataset.

```
# Normalize the numeric data

train_normalized <- as.data.frame(scale(train$x,scale = FALSE, center = TRUE))

test_normalized <- as.data.frame(scale(test$x,scale = FALSE, center = TRUE))
```

The covariance is a measure of how much two random variables vary together. The calculation for the covariance matrix can be expressed as:

$$C = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})^T$$

Principal Component Analysis

Principal Component Analysis (PCA) is a general idea to reduce the dimensions of the MNIST dataset while choosing the essential features that still represent the dataset by preserving those with more variance and reject those with less variance. With fewer dimensions, visualization can be more viable.

```
# Run PCA using the training covariance matrix

pca_train <- prcomp(train_normalized)
```

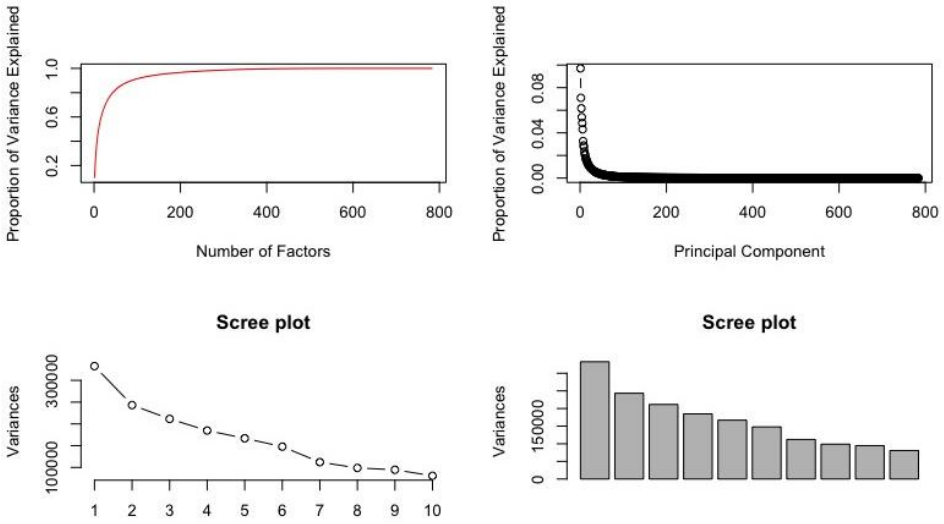
The variance is explained by dividing the sum of squares between variances by all the 784 components.

```
# Look at the 50th variance
```

NmbrPCs	CumVar
50	0.8246469

By Heuristics, the cumulative percentage of the explained variance is more than 80%. We choose 50 PCs that with 50 PCs contain 82.5% of the variances in data.

```
# Plot the data.
```



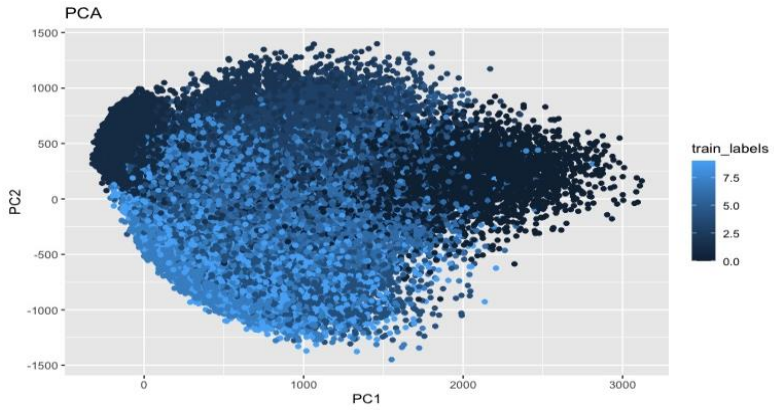
In the graphs, the "knee-deep" is at about 50 PCs, more than 50 PCs just explaining a small percent of the variances.

```
> summary(pca_train)$importance[,1:50]
```

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10
Standard deviation	576.82291	493.23822	459.89930	429.85624	408.56680	384.50613	334.93015	314.44305	307.72756	284.27069
Proportion of Variance	0.09705	0.07096	0.06169	0.05389	0.04869	0.04312	0.03272	0.02884	0.02762	0.02357
Cumulative Proportion	0.09705	0.16801	0.22970	0.28359	0.33228	0.37540	0.40812	0.43696	0.46458	0.48815
	PC11	PC12	PC13	PC14	PC15	PC16	PC17	PC18	PC19	PC20
Standard deviation	268.91192	263.35964	242.54251	240.86113	232.64513	225.48408	213.10234	209.23303	201.75565	198.79585
Proportion of Variance	0.02109	0.02023	0.01716	0.01692	0.01579	0.01483	0.01325	0.01277	0.01187	0.01153
Cumulative Proportion	0.50924	0.52947	0.54663	0.56355	0.57934	0.59417	0.60741	0.62018	0.63205	0.64358
	PC21	PC22	PC23	PC24	PC25	PC26	PC27	PC28	PC29	PC30
Standard deviation	191.18979	185.78265	180.81285	176.88015	174.03319	169.63513	166.91101	164.19678	159.79109	153.90297
Proportion of Variance	0.01066	0.01007	0.00954	0.00913	0.00883	0.00839	0.00813	0.00786	0.00745	0.00691
Cumulative Proportion	0.65424	0.66431	0.67385	0.68297	0.69180	0.70020	0.70832	0.71619	0.72363	0.73054
	PC31	PC32	PC33	PC34	PC35	PC36	PC37	PC38	PC39	PC40
Standard deviation	150.20906	149.06967	143.73821	141.81321	139.7970	136.52213	131.68485	129.32998	128.47497	127.24647
Proportion of Variance	0.00658	0.00648	0.00603	0.00587	0.0057	0.00544	0.00506	0.00488	0.00481	0.00472
Cumulative Proportion	0.73712	0.74361	0.74963	0.75550	0.7612	0.76663	0.77169	0.77657	0.78139	0.78611
	PC41	PC42	PC43	PC44	PC45	PC46	PC47	PC48	PC49	PC50
Standard deviation	125.13824	123.49581	119.78451	116.84530	114.88631	113.40383	111.40683	109.79215	107.97641	105.04987
Proportion of Variance	0.00457	0.00445	0.00419	0.00398	0.00385	0.00375	0.00362	0.00352	0.00340	0.00322
Cumulative Proportion	0.79068	0.79512	0.79931	0.80329	0.80714	0.81089	0.81451	0.81803	0.82143	0.82465

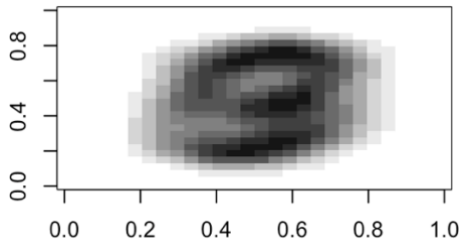
We can see that with 50 PCs already explain a large percent of the variability.

Having reduced the overwhelming dimensions, we plot the first two most influential PCs that account for the initial influential characteristics. Our plot below does not indicate the presence of any clusters in the dataset, implying the 2-component PCA does not give us the desired data patterns. Not even a 3-D plot gives any clusters.

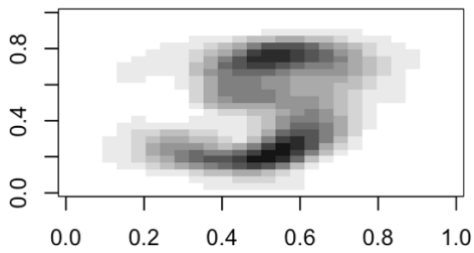



```
# Give an example of digit reconstruction.
PCA <- c(1,10,50,784)
```

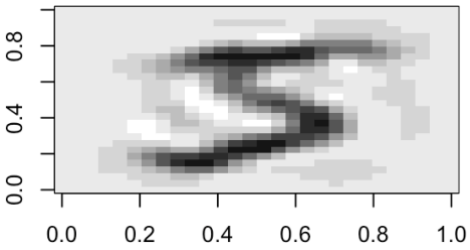
PCA = 1



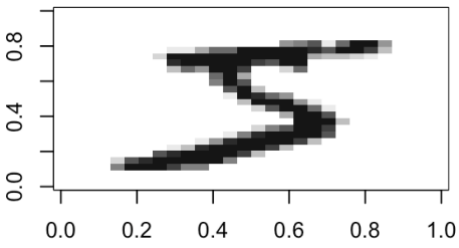
PCA = 10



PCA = 50



PCA = 784



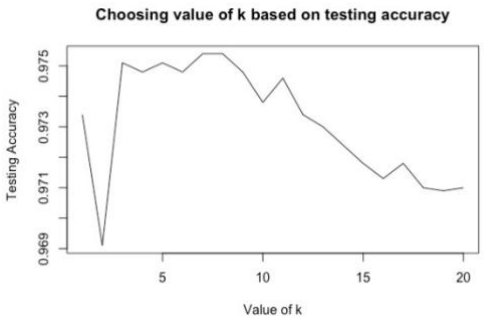
So PCA = 50 is acceptable to see the images of the digits.

The k-Nearest Neighbor Classifier

The k-Nearest Neighbor is the most straightforward image classification algorithm. For predicting a new instance, KNN calculates the Euclidean Distance between the new instance and all the instances in the entire training set. Then, the algorithm looks for the top K nearest (most similar) instances and outputs the class with the highest frequency (most vote) as the prediction. (Wikipedia, 2020)

```
# Run KNN with k from 1 to 20 and plot the accuracy for different k (1:20)
```

```
> t(knn_acc)
      [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
[1,] 1.0000 2.0000 3.0000 4.0000 5.0000 6.0000
[2,] 0.9734 0.9691 0.9751 0.9748 0.9751 0.9748
      [,7]   [,8]   [,9]   [,10]   [,11]
[1,] 7.0000 8.0000 9.0000 10.0000 11.0000
[2,] 0.9754 0.9754 0.9748  0.9738  0.9746
      [,12]  [,13]   [,14]   [,15]   [,16]
[1,] 12.0000 13.000 14.0000 15.0000 16.0000
[2,]  0.9734  0.973  0.9724  0.9718  0.9713
      [,17]  [,18]   [,19]  [,20]
[1,] 17.0000 18.000 19.0000 20.000
[2,]  0.9718  0.971  0.9709  0.971
```



From the plot, choose the k = 8, which gives the best accuracy.

Confusion Matrix

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2    3    4    5    6    7    8    9
         0  971    0    8    0    1    3    2    0    4    3
         1    1 1131    0    1    4    0    5   17    0    4
         2    1    2 1002    1    0    0    0    4    3    2
         3    0    0    0  978    0    8    0    0   11    6
         4    0    0    1    1  953    1    3    4    5    7
         5    1    0    0   11    0  866    1    0    6    4
         6    5    1    2    0    5    8  947    0    2    1
         7    1    0   15    6    1    1    0  989    5    5
         8    0    0    4    5    1    3    0    0  933    4
         9    0    1    0    7   17    2    0   14    5  973

Overall Statistics

               Accuracy : 0.9743
                 95% CI : (0.971, 0.9773)
    No Information Rate : 0.1135
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9714

 Mcnemar's Test P-Value : NA

Statistics by Class:
```

Calculate the accuracy for k = 8

```
> table_prediction
           test_labels
prediction    0    1    2    3    4    5    6    7    8    9
         0  971    0   10    0    0    3    5    0    4    4
         1    1 1129    2    1    4    0    5   23    0    5
         2    1    2  996    2    0    0    0    4    3    3
         3    0    1    1  976    0   10    0    0   13    5
         4    0    0    1    1  946    1    2    2    5    9
         5    1    0    0    9    0  865    1    0    7    6
         6    5    2    3    0    7    9  945    0    2    1
         7    1    0   13    7    0    1    0  983    3    6
         8    0    0    6    9    2    0    0    0  930    1
         9    0    1    0    5   23    3    0   16    7  969
> accuracy = hit/sum(table_prediction)
> accuracy
[1] 0.9743
```

The most misclassifying couple numbers are 4 ~ 9, 7 ~ 9 and 7 ~ 1. The number 9 is the easiest number to be misclassified.

The algorithm had chosen the 20 as the initial value of k from the training set square root of observations. The value k = 8 with the best accuracy indicates that the machine KNN algorithm model will predict the class of any input digit with 97.43% accuracy using the 8 nearest neighbours.

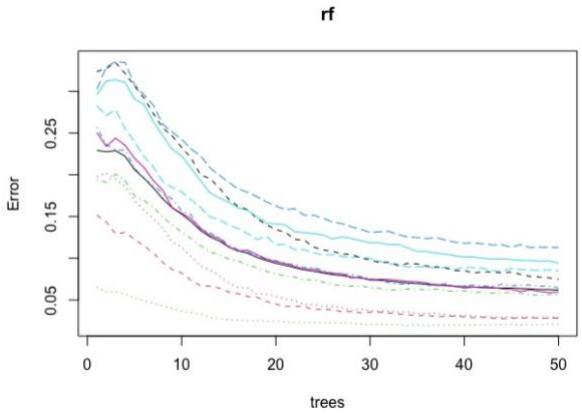
Random Forest

The random forest is a classification algorithm consisting of many decision trees. It uses bagging and features randomness when building each tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree. (Yiu, 2019)

```
# Generate a random sample of "numTrain" indexes
```

```
Call:
 randomForest(x = train_final, y = train_labels, ntree = numTrees)
               Type of random forest: classification
                     Number of trees: 50
No. of variables tried at each split: 7

        OOB estimate of  error rate: 6.19%
Confusion matrix:
     0    1    2    3    4    5    6    7    8    9 class.error
0 5755    0   15   13   16   27   55    9   26    7  0.02836400
1    1 6602   43   22    7   10    8   13   27    9  0.02076535
2   51    8 5574   71   40   15   26   58   98   17  0.06445116
3   13    3  106 5610    7  122   21   51  145   53  0.08497798
4   12   20   48    8 5495    4   39   23   37  156  0.05939747
5   31    3   26  144   39 5015   56    8   60   39  0.07489393
6   31    7   21    9   15   72 5745    1   15    2  0.02923285
7    5   31   69   15   60   12    1 5914   20  138  0.05602554
8   23   44   80  177   40  149   39   33 5191   75  0.11280123
9   25    9   32   90  183   34    4  135   49 5388  0.09430156
```




```
# Confusion Matrix
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2    3    4    5    6    7    8    9
         0  961    0   12    1    1    4    8    1    7    2
         1    0 1122    0    0    1    1    2    6    0    5
         2    2    4  967    6    3    5    3   15   10    6
         3    0    1   11  954    0   14    0    1   15   10
         4    0    0    6    0  935    3    3    8    8   22
         5    2    2    1   17    3  846    7    0   17    8
         6    9    3    6    2   10    7  933    0    4    2
         7    1    0   10    6    2    1    0  971    8   10
         8    3    2   18   16    5    7    2    3  896    9
         9    2    1    1    8   22    4    0   23    9  935

Overall Statistics

               Accuracy : 0.952
                 95% CI : (0.9476, 0.9561)
    No Information Rate : 0.1135
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9466

 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            0.9806   0.9885   0.9370   0.9446   0.9521   0.9484   0.9739   0.9446   0.9199   0.9267
Specificity            0.9960   0.9983   0.9940   0.9942   0.9945   0.9937   0.9952   0.9958   0.9928   0.9922
Pos Pred Value         0.9639   0.9868   0.9471   0.9483   0.9492   0.9369   0.9559   0.9623   0.9324   0.9303
Neg Pred Value         0.9979   0.9985   0.9928   0.9938   0.9948   0.9949   0.9972   0.9937   0.9914   0.9918
Prevalence             0.0980   0.1135   0.1032   0.1010   0.0982   0.0892   0.0958   0.1028   0.0974   0.1009
Detection Rate         0.0961   0.1122   0.0967   0.0954   0.0935   0.0846   0.0933   0.0971   0.0896   0.0935
Detection Prevalence   0.0997   0.1137   0.1021   0.1006   0.0985   0.0903   0.0976   0.1009   0.0961   0.1005
Balanced Accuracy      0.9883   0.9934   0.9655   0.9694   0.9733   0.9711   0.9846   0.9702   0.9564   0.9594
```

The predictions were submitted and received a score of 94.66%.

In this Confusion Matrix, there are mostly confusions between 2, 7, 8 and 9. Also, fewer couple numbers are classifying as previous Confusion Matrix.

Random Forest performed surprisingly well with PCA, was easy to train, and relatively quick.

Now let take a look at Random Forest without PCA:

```
# Confusion Matrix without PCA

confusionMatrix(pred2, test_labels_rf2)
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2    3    4    5    6    7    8    9
         0  970    0    5    0    1    6    8    0    3    4
         1    0 1122    0    0    0    0    3    6    0    4
         2    1    3  998   13    2    0    1   20    5    2
         3    0    4    6  968    0   15    0    0   13   12
         4    0    0    1    0  952    1    4    4    6   11
         5    2    2    0    8    0  855    3    0    6    9
         6    3    2    4    0    8    5  933    0    5    1
         7    1    0    8   11    1    3    0  983    4    5
         8    3    1    8    6    3    5    6    4  924    4
         9    0    1    2    4   15    2    0   11    8  957

Overall Statistics

               Accuracy : 0.9662
                 95% CI : (0.9625, 0.9697)
    No Information Rate : 0.1135
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9624

 Mcnemar's Test P-Value : NA

Statistics by Class:
```

The accuracy is 96.62%, pretty much higher than with PCA (94.66%) but takes a longer time to run.

Conclusion

The analysis aimed at solving digit recognition by training the models to receive an input image and classify it accordingly. PCA reduced the dimensions of the data to only retain the components with the highest variance, which contains the essential characteristics for digit classification. However, from the 2-component plot, there is no existence of clusters; hence PCA is not the best for digit recognition at hand. The KNN algorithm for digit recognition realizes that we can predict the identity of an input digit with 5 nearest neighbours and achieve 96.98% accuracy.

On the other hand, the random forest algorithm can perform digit recognition at 94.86% accuracy using 50 trees and 4 candidate randomly sampled variables at each split. The K-Nearest Neighbour (KNN) is a simple yet accurate algorithm to solve the Digit Recognition problem. Among these, kNN predicts higher accuracy, and it is also faster than Random Forest. Apart from classification error, the size of the dataset, training time and linearity can be used in deciding the method of classification. For the small dataset, we use the algorithms (Naïve Bayes and linear regression) with high bias while we use algorithms with low bias (decision trees and KNN). Linearity implies whether a straight line can separate the classes in the data. Absence of linearity or presence of linearity dictates the type of linearity algorithm. The training time also determines the choice of algorithm, and the higher the training time, the higher the accuracy. While our analysis realizes KNN as the best algorithm, it is slow to train, and its accuracy can further increase. The Nearest Cluster approach is used to improve the accuracy of KNN by optimizing the number of k neighbours and lessen the training time (Alizadeh, 2009)

Further, voting between two techniques produces a more accurate prediction. We have two predictions. We combine them in one data frame, and the final prediction comes from voting between two models. If all models predict different digits, we rely on the kNN model since it has higher accuracy (Marjani, 2017)

References

Alizadeh, H. M.-B. (2009). *A New Method for Improving the Performance of K Nearest Neighbor using Clustering Technique*. J. Convergence Inf. Technol.

Marjani, A. (2017, October 9). *Digit recognition with KNN, neural network, and SVM*. Retrieved from Amazonaws: https://rstudio-pubs-static.s3.amazonaws.com/326787_431decc80c1a4ec797c3ef1e0b095e41.h

Wikipedia. (2020, November). *K-nearest neighbours algorithm*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Yiu, T. (2019, June 12). *Understanding Random Forest*. Retrieved from towardsdatascience: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

Appendix

```
# Load dataset
library(dslabs)
mnist <- read_mnist()
train_images <- mnist$train$images
test_images <- mnist$test$images
train_labels <- mnist$train$labels
test_labels <- mnist$test$labels


# Inspect contents
#summary(train_images)
summary(train_labels)

# How the picture looks like
train_images[1,]

show_digit <- function(arr784, col=gray(12:1/12), ...) {
  image(matrix(arr784, nrow=28)[,28:1], col=col, ...)}
show_digit(train_images[1,])



# Normalize the numeric data
train_normalized <- as.data.frame(scale(train_images,scale = FALSE,center = TRUE))
test_normalized <- as.data.frame(scale(test_images,scale = FALSE,center = TRUE))



# PCA
set.seed(132)
pca_train <- prcomp(train_normalized)


# The percent of the variances in data
variance_explained <- as.data.frame(pca_train$sdev^2/sum(pca_train$sdev^2))
variance_explained <- cbind(c(1:784), cumsum(variance_explained))
colnames(variance_explained) <- c("NmbrPCs","CumVar")


# Look at 50th variance
variance_explained[50, ]

##    NmbrPCs    CumVar
## 50      50 0.8246469

# Plot the data.
par(mfrow=c(2,2))
plot (variance_explained$NmbrPCs, variance_explained$CumVar,
      xlab = "Number of Factors", ylab = "Proportion of Variance Explained",
      type = "l", col = "red")
plot (pca_train$sdev^2/sum(pca_train$sdev^2), xlab = "Principal Component",
     ylab = "Proportion of Variance Explained", type = "b" )
plot(pca_train, type = "l", main = "Scree plot")
plot(pca_train, type = "barplot", main = "Scree plot")
```

```r
# summary(pca_train)
summary(pca_train)$importance[,1:50]

# Do the actual dimension reduction (matrix of 784 columns should be converted
# to matrix of 50 columns)
pca_rot <- pca_train$rotation[,1:50]
train_final <- as.matrix(train_images)%*%(pca_rot)
test_final <- as.matrix(test_images)%*%(pca_rot)
test_final <- data.frame(test_final)
train_final <- data.frame(train_final)


# Visualize the different classes in two dimensions
library(ggplot2)

ggplot(train_final, aes(PC1,PC2,color=train_labels)) + geom_point() +
  labs(x= "PC1",y= "PC2") + ggtitle("PCA")



# Inspect the reconstruction of the original data from these two components
mean.train = colMeans(train_images)
PCA <- c(1,10,50,784)
par(mfrow=c(2,2))
i <- 1
while(i <= length(PCA))
{nComp = PCA[i]
  train.pca.re = pca_train$x[,1:nComp] %*% t(pca_train$rotation[,1:nComp])
  train.pca.re = scale(train.pca.re, center = -mean.train,scale=F)
  show_digit(train.pca.re[1,])
  i<-i+1}

# So PCA = 50 is acceptable to see the images of the digits.



# Run KNN with k from 1 to 20
library(class)
set.seed(132)
knn_acc <- matrix(nrow=20,ncol=2)
i=1
for (k in 1:20){
  prediction = knn(train_final,test_final,train_labels,k=k)
  knn_acc[i,] <- c(k,mean(prediction==test_labels))
  i = i+1
}
t(knn_acc)



# Plot the accuracy for different k (1:20)
par(mfrow=c(1,1))
plot(knn_acc[,1],knn_acc[,2],type = "l",
     xlab = "Value of k",
     ylab = "Testing Accuracy",
     main = "Choosing value of k based on testing accuracy")

# From the plot, choose the k = 8, which gives the best accuracy.
best_prediction <- knn(train_final,test_final,train_labels,k=8)
test_labels_knn <- factor(test_labels)
```

```r
# Confusion Matrix
library(caret)

confusionMatrix(best_prediction, test_labels_knn)

# Calculate the accuracy for k = 8
hit = 0
for (i in 1:10){
  hit = hit + table(best_prediction,test_labels)[i,i]
}
table_prediction <- table(prediction,test_labels)
table_prediction

accuracy = hit/sum(table_prediction)
accuracy



# Random Forest with PCA
library(randomForest)

library(readr)
library(lattice)


set.seed(132)
numTrain <- 40000
numTrees <- 50

# Generate a random sample of "numTrain" indexes
rows <- sample(1:nrow(train_final), numTrain)
train_labels <- factor(train_labels)
rf <- randomForest(train_final, train_labels, ntree=numTrees)

plot(rf)

# Make prediction
pred <- predict(rf, test_final)
test_labels_rf <- factor(test_labels)

# Confusion Matrix with PCA
confusionMatrix(pred, test_labels_rf)



# Run Random Forest without PCA
train_final2 <- as.matrix(train_images)
test_final2 <- as.matrix(test_images)
test_final2 <- data.frame(test_final2)
train_final2 <- data.frame(train_final2)

rows <- sample(1:nrow(train_final2), numTrain)
train_labels <- factor(train_labels)
rf <- randomForest(train_final2, train_labels, ntree=numTrees)

plot(rf)

# Make prediction
pred2 <- predict(rf, test_final2)
test_labels_rf2 <- factor(test_labels)
```

```
# Confusion Matrix without PCA
confusionMatrix(pred2, test_labels_rf2)
```