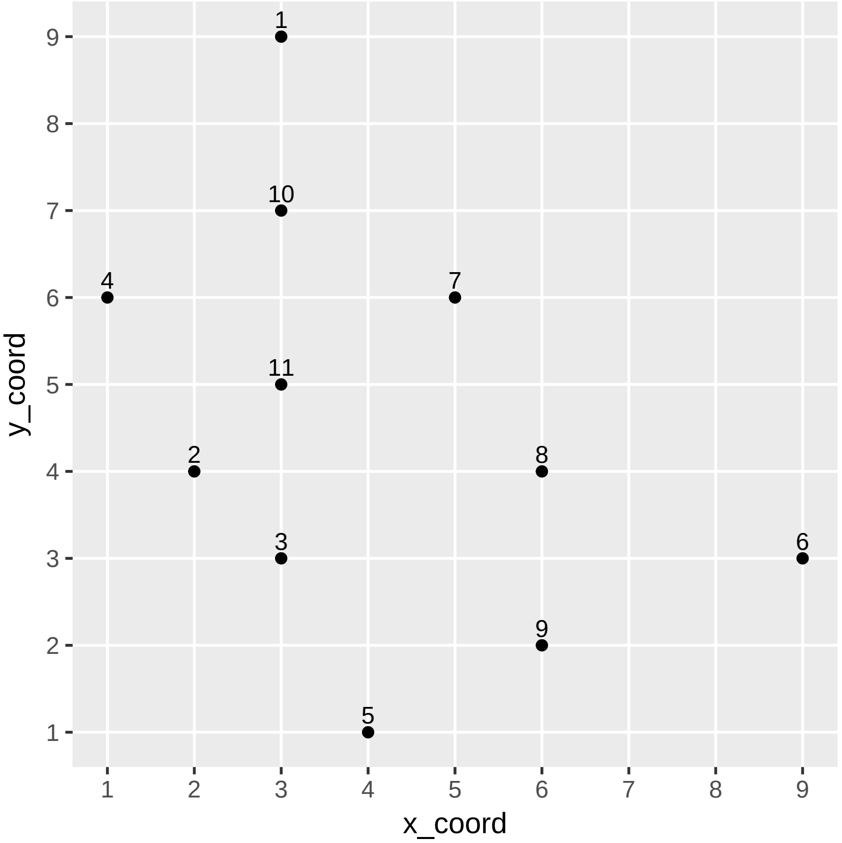
Homework5

Ivan Ojiambo

October 26, 2017

Exe1

```
library(ggplot2)
data <- data.frame(ID      = c(1,2,3,4,5,6,7,8,9,10,11),
                   x_coord = c(3,2,3,1,4,9,5,6,6,3,3),
                   y_coord = c(9,4,3,6,1,3,6,4,2,7,5),
                   class   = c(1,1,1,1,0,0,0,0,0,0,"???"))

data$class <- as.factor(data$class)
#str(data)
ggplot(data, aes(x=x_coord, y=y_coord, label=ID, group = 1) ) +
  geom_point() +
  geom_text(vjust = -0.5, size = 3) +
  coord_fixed() +
  scale_x_continuous(minor_breaks = seq(0 , 10, 1), breaks = seq(0, 10, 1)) +
  scale_y_continuous(minor_breaks = seq(0 , 10, 1), breaks = seq(0, 10, 1))
```



when K = 1

- By looking at the graph id = 2 is the closed Neighbour to the coordinates (3, 5) therefore it belongs to class 1

when k = 5

- 3 => class = 1
- 4 => class = 1
- 10 => class = 0
- 7 => class = 0
- from the observation majority of the neighbouring coordinates have class = 1 , therefore class of coordinate (3, 5) = 1

when k = 10

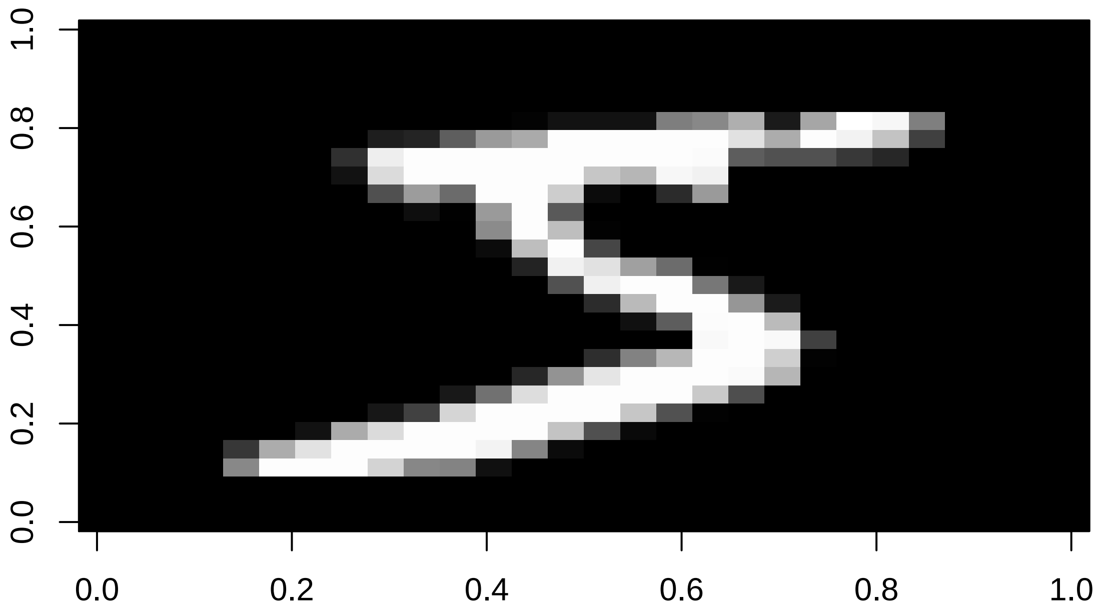
- majority of the cordinated have class = 0, therefore coordinate (3, 5) = 0

Exe2

a

b. Visualise few examples of each class. Code for visualising one image is given below. Do the labels of the images correspond to what is on the image?

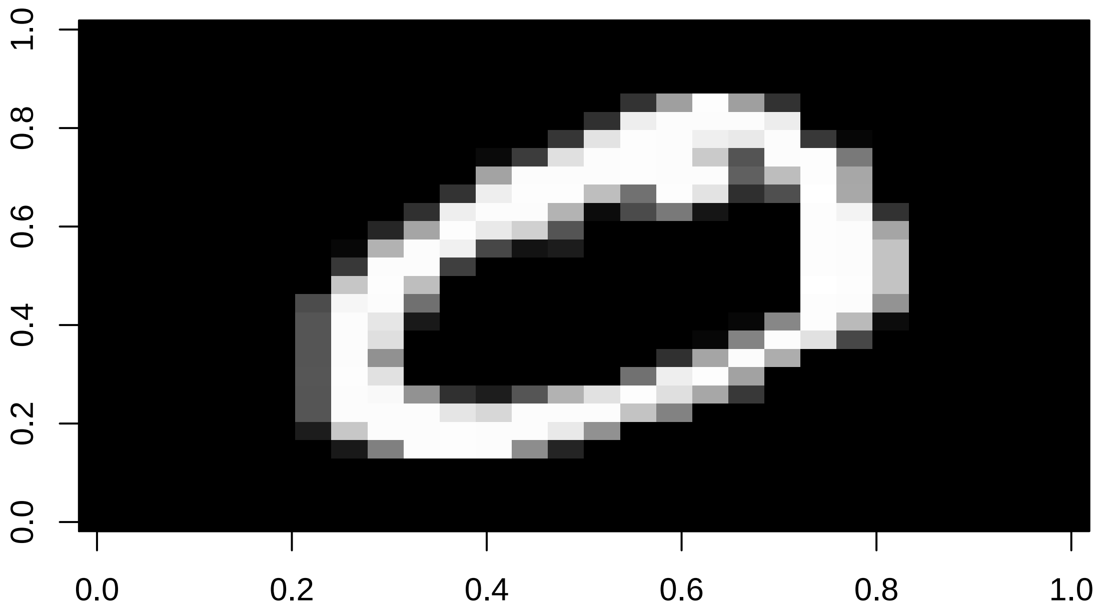
```
## first we need to define colors:
colors <- c('black', 'white')
cus_col <- colorRampPalette(colors=colors)

# Play around with an index of image that you want to visualise
visualise_img <- function(data,index){
  img <- array(data[index,],dim=c(28,28))
  img <- img[,28:1]
  image(img, col=cus_col(256))
}

visualise_img(mnist$x,1)
```



```
print(paste("Correct label of the first image is:", mnist$y[1]))
```

```
## [1] "Correct label of the first image is: 5"
```

```
visualise_img(mnist$x, 2)
```



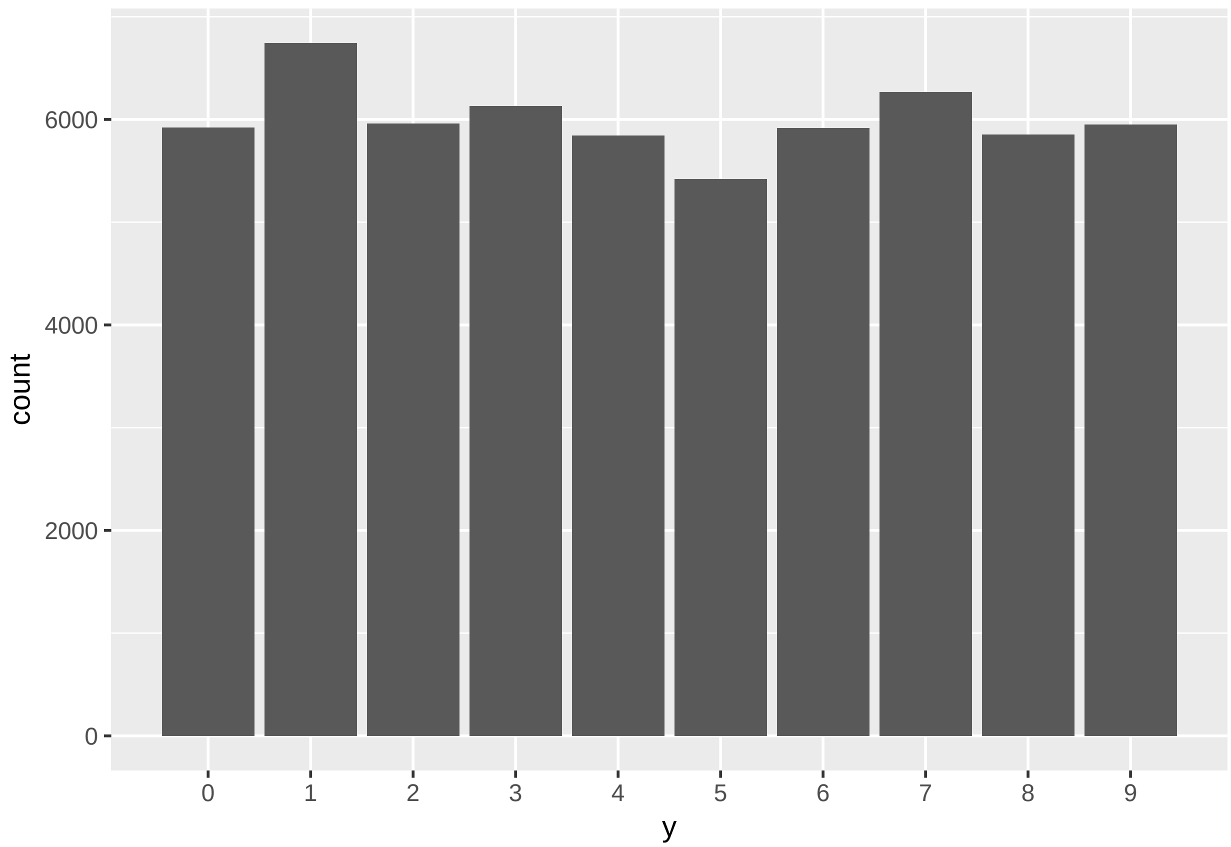
```r
print(paste("Correct label of the first image is:", mnist$y[2]))
```

```
## [1] "Correct label of the first image is: 0"
```

```r
#yes the labels do correspond to the image
```

c

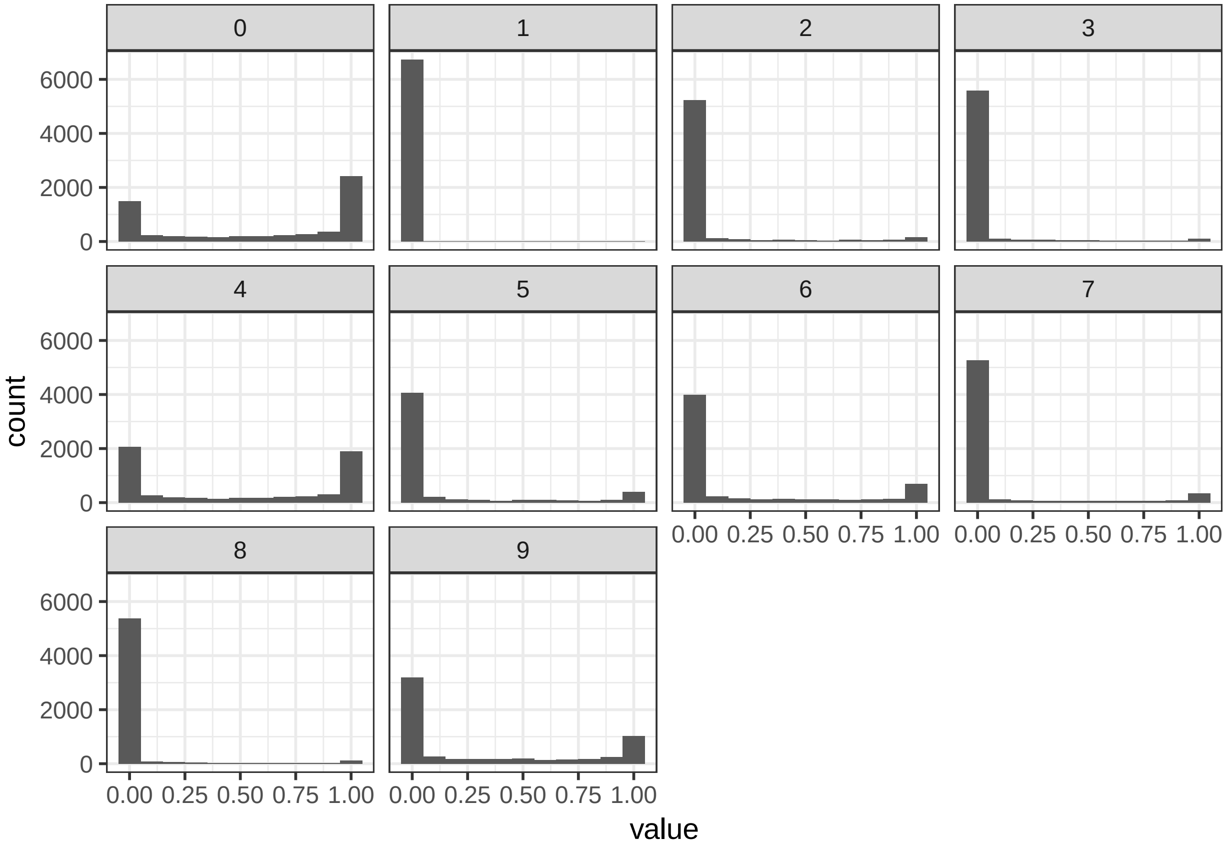
```r
#c How can we visualise these labels in a bit more compact way
dataframe <- as.data.frame(mnist)
ggplot(dataframe, aes(x=y)) + geom_bar() +
  scale_x_continuous(minor_breaks = seq(0 , 10, 1), breaks = seq(0, 10, 1))
```



##d

```
pixel <- data.frame('value' = mnist$x[,400],
                    'label' = mnist$y)

ggplot(pixel, aes(x = value)) +
  geom_bar(binwidth = 0.1) +
  theme_bw()+
  facet_wrap(~label)
```

```
## Warning: `geom_bar()` no longer has a `binwidth` parameter. Please use
## `geom_histogram()` instead.
```



From the graph, + For label 0 , the 400th pixel has it has majority of

- For label 1, the 400th pixel is centered around the zero value
- FOr label 2, the black color is centered around 0 value

Exe3

a Split the data into training and testing data by filling in the ? parts in the following code:

```
# You can use set.seed() in order to reproduce stochastic results
set.seed(1111)
new_indx <- sample(c(1:nrow(mnist$x)), size = 4000, replace = FALSE)

sample_img <- mnist$x[new_indx, ]

sample_labels <- mnist$y[new_indx]


train_img    <- sample_img[1:3000,]
train_labels <- sample_labels[1:3000]
#head(train_labels)

test_img    <- sample_img[3001:4000,]
```

```r
test_labels <- sample_labels[3001:4000]

str(train_img) # Make sure you have 3000 rows here
```

```
##  num [1:3000, 1:784] 0 0 0 0 0 0 0 0 0 0 ...
```

```r
str(test_img) # Make sure you have 1000 rows here
```

```
##  num [1:1000, 1:784] 0 0 0 0 0 0 0 0 0 0 ...
```

b

```r
#Next we define a distance function between two images to be euclidean distance.
#Fill in the code below so the function would calculate and return the euclidean distance between img1 a

dist <- function(img1, img2) {
 sqrt(sum((img1 -img2) ^2))
}

print(paste("Distance between images of class", train_labels[2], "and", train_labels[8], "is", dist(trai
```

```
## [1] "Distance between images of class 9 and 9 is 8.82156666623062"
```

```r
print(paste("Distance between images of class",train_labels[2], "and", train_labels[4], "is", dist(train
```

```
## [1] "Distance between images of class 9 and 6 is 10.0315864086725"
```

c

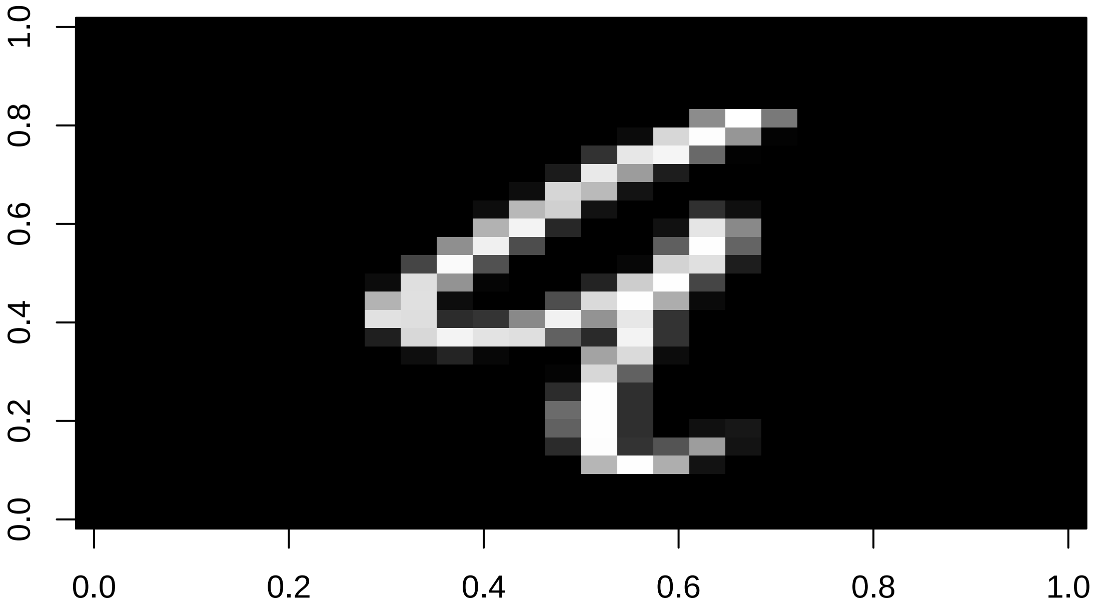
```r
unknown_img <- test_img[4,]
true_label <- test_labels[4]

all_distances <- apply(train_img, 1, function(img) dist(unknown_img, img))
#head(all_distances)


#c.2. Now let's find out which image is closest to our `unknown_img`. Fill in the code.

closest_index <- which(all_distances == min(all_distances))

#c.3. Almost done, now report a label with index i in labels by filling in the code.

predicted_label <-train_labels[closest_index]

#Compare it to the true label of the first image in the test labels. Is it the same?

(predicted_label == true_label)
```

```
## [1] TRUE
```

```r
print(paste("Predicted class for the first image is", predicted_label ,"and the true label is",  true_la
```

```
## [1] "Predicted class for the first image is 9 and the true label is 9"
```

```
#visulaising the image
visualise_img(train_img, closest_index)
```



d

```
classify <- function(unknown_img) {
    all_distances <-  apply(train_img, 1, function(img) dist(unknown_img, img))

    closest_index <- which(all_distances <= min(all_distances))
    predicted_label <-train_labels[closest_index]
    return(predicted_label)
}
#testing
print(paste("Predicted class for the first image is", classify(unknown_img),"and the true label is",  t
```

```
## [1] "Predicted class for the first image is 9 and the true label is 9"
```

e

```
classify_knn <- function(unknown_img, k = 5) {
  # This step we already know from the previous exercises
  all_distances <-  apply(train_img, 1, function(img) dist(unknown_img, img))
  df <- data.frame("index"= c(1:3000),
                   "dist" = all_distances)
  df <- df[with(df, order(dist)), ]


  # We need to get indexes of K smallest distances
  # (hint: use functions order() and head())
  knn = head(df$index, k)

  # you can print potential predictions
  #print(train_labels[knn])
  # print(names(sort(table(train_labels[knn]), decreasing = TRUE)))

  # Very small step is left, return the most frequently predicted label
  return(names(sort(table(train_labels[knn]), decreasing = TRUE))[1])
}
classify_knn(unknown_img)
```

```
## [1] "9"
```

```
#Test this version of KNN, experiment with different `K`s

print(paste("Predicted class for the first image is", classify_knn(unknown_img, k = 5),"and the true la
```

```
## [1] "Predicted class for the first image is 9 and the true label is 9"
```

Exe4

a

```
test_predicted <- apply(test_img, 1,  function(img) classify_knn(img))

correct_predication <-  sum(as.numeric(test_predicted) - as.numeric(test_labels) ==0)
 print(paste("Number of correct predicition =", correct_predication ))
```

```
## [1] "Number of correct predicition = 908"
```

```
#accuracy
knn_accuracy = (correct_predication/1000)
print(paste("Final accuracy of our nearest neighbor classifier is", knn_accuracy,"- not bad!"))
```

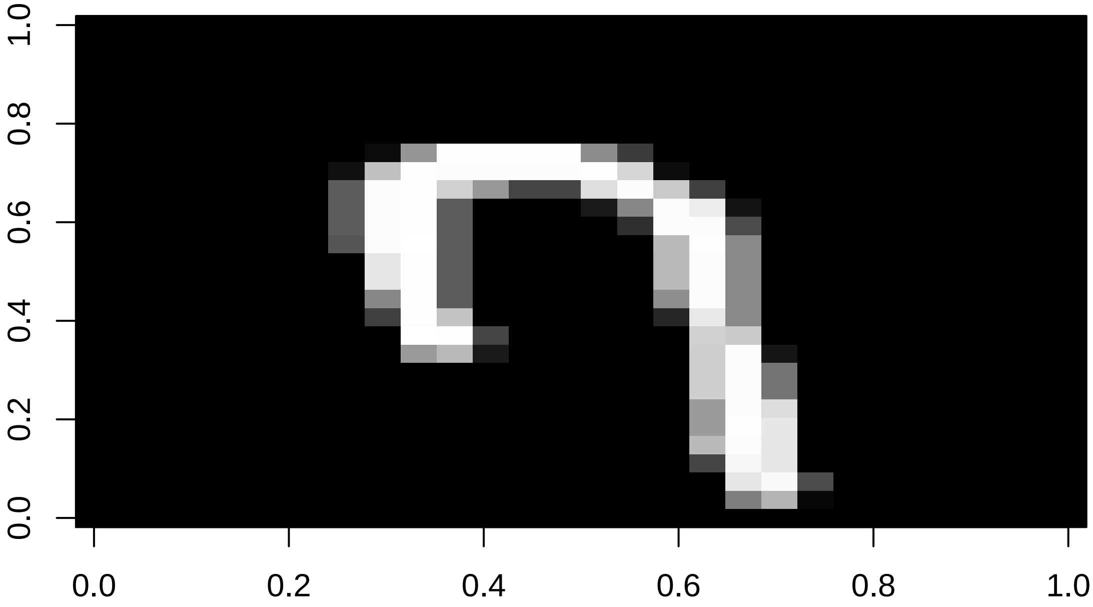
```
## [1] "Final accuracy of our nearest neighbor classifier is 0.908 - not bad!"
```

```
#for training dataset

train_predication <- vector(mode = "character", length= 3000)
for(i in 1:3000){
  unknown_img <- train_img[i,]
  #remove i th item from train_image
  testing_img <- train_img[-i,]

   #remove i th item from train_labels
  true_labels <- train_labels[-i]
```

```r
  #difference in distance
  all_distances <-  apply(testing_img, 1, function(img) dist(unknown_img, img))

  #adding the differences in a data frame
  df <- data.frame("index"= c(1:2999),
                   "dist" = all_distances)

  #sort the distance in descending order
  df <- df[with(df, order(dist)), ]

  #return 5 rows
  knn = head(df$index, 5)

  train_predication[i] <- names(sort(table(train_labels[knn]), decreasing = TRUE))[1]

}


#find number of train image which are the same train labels
train_img_predicated <- sum(as.numeric(train_predication) == as.numeric(train_labels))

#calculating the accuracy proportion
knn_accuracy <- (train_img_predicated/length(train_labels))*100
print(paste("Final accuracy of our nearest neighbor classifier is", knn_accuracy,"- not bad!"))
```

```
## [1] "Final accuracy of our nearest neighbor classifier is 65.0333333333333 - not bad!"
```

```r
#misclasified images

# Set an index of missclassified instance you want to examine
index = 12

miss_ind = which(test_predicted != test_labels)[index] # remember function `which` in R?

colors<-c('black','white')
cus_col<-colorRampPalette(colors=colors)

img <- array(test_img[miss_ind,],dim=c(28,28))
img <- img[,28:1]
image(img, col=cus_col(256))
```



```
print(paste("This image has a class", test_labels[miss_ind],"was incorrectly predicted as",  test_predi
```

```
## [1] "This image has a class 7 was incorrectly predicted as 9"
```

b

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.4.2

## Loading required package: lattice
```

```
# We will discuss this line further in more details,
# we need it now as without it, caret tries to be very smart
# and training takes too much time...
ctrl <- trainControl(method="none", number = 1, repeats = 1)
```

```
## Warning: `repeats` has no meaning for this resampling method.
```

```
# we should use train_labels as factors, otherwise caret thinks that
# this is a regression problem
knn_predication <-function(k=5, img){
  knn_fit <-  train(y = as.factor(train_labels), x = data.frame(train_img),
                   method = "knn", trControl = ctrl, tuneGrid = data.frame(k))

  #Use learned nearest neighbor classifier (model) for predicting test images:
  test_predicted = predict(knn_fit, data.frame(img))
```

```r
  return(test_predicted)
}

test_predicted <- knn_predication(5, test_img)
print(paste("Accuracy of caret nearest neighbor classifier is", sum(test_predicted == test_labels)/leng
```

```
## [1] "Accuracy of caret nearest neighbor classifier is 0.906"
```

```r
#accuracy on training data when k = 20
test_predicted1 <- knn_predication(20, train_img)
print(paste("Accuracy of caret nearest neighbor classifier is", sum(test_predicted1 == test_labels)/leng
```

```
## [1] "Accuracy of caret nearest neighbor classifier is 0.301"
```

```r
#accuracy on training data when k = 120
test_predicted1 <- knn_predication(120, train_img)
print(paste("Accuracy of caret nearest neighbor classifier is", sum(test_predicted1 == test_labels)/leng
```

```
## [1] "Accuracy of caret nearest neighbor classifier is 0.325"
```

```r
#accuracy on testing data when k = 20
test_predicted1 <- knn_predication(20, test_img)
print(paste("Accuracy of caret nearest neighbor classifier is", sum(test_predicted1 == test_labels)/leng
```

```
## [1] "Accuracy of caret nearest neighbor classifier is 0.888"
```

```r
#accuracy on testing data when k = 120
test_predicted1 <- knn_predication(120, test_img)
print(paste("Accuracy of caret nearest neighbor classifier is", sum(test_predicted1 == test_labels)/leng
```

```
## [1] "Accuracy of caret nearest neighbor classifier is 0.791"
```

```r
#using cinfusion matrix
confusionMatrix(test_predicted, test_labels)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 100   0   2   0   0   0   2   0   2   0
##          1   0  95   7   2   1   0   0   4   2   1
##          2   1   0  79   0   0   0   0   0   0   0
##          3   0   0   1  96   1   2   0   0   6   2
##          4   0   0   1   0  87   0   0   0   1   2
##          5   1   0   0   6   1  73   1   0   2   0
##          6   0   0   0   1   0   4  84   0   0   0
##          7   0   0   1   4   0   0   0 106   0   3
##          8   0   0   3   3   0   0   0   0  85   0
##          9   0   0   0   2  11   4   0   4   3 101
##
## Overall Statistics
##
##                Accuracy : 0.906
##                  95% CI : (0.8862, 0.9234)
##     No Information Rate : 0.114
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8954
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9804   1.0000   0.8404   0.8421   0.8614   0.8795
## Specificity            0.9933   0.9812   0.9989   0.9865   0.9956   0.9880
## Pos Pred Value         0.9434   0.8482   0.9875   0.8889   0.9560   0.8690
## Neg Pred Value         0.9978   1.0000   0.9837   0.9798   0.9846   0.9891
## Prevalence             0.1020   0.0950   0.0940   0.1140   0.1010   0.0830
## Detection Rate         0.1000   0.0950   0.0790   0.0960   0.0870   0.0730
## Detection Prevalence   0.1060   0.1120   0.0800   0.1080   0.0910   0.0840
## Balanced Accuracy      0.9869   0.9906   0.9197   0.9143   0.9285   0.9338
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9655   0.9298   0.8416   0.9266
## Specificity            0.9945   0.9910   0.9933   0.9731
## Pos Pred Value         0.9438   0.9298   0.9341   0.8080
## Neg Pred Value         0.9967   0.9910   0.9824   0.9909
## Prevalence             0.0870   0.1140   0.1010   0.1090
## Detection Rate         0.0840   0.1060   0.0850   0.1010
## Detection Prevalence   0.0890   0.1140   0.0910   0.1250
## Balanced Accuracy      0.9800   0.9604   0.9175   0.9498
```

c

```
library(caret)
##training data
rpfit <- train(y = as.factor(train_labels), x = data.frame(train_img), method = "rpart", tuneGrid = data

#accuracy of test images when using rpart method
test_predicted1 = predict(rpfit, data.frame(test_img))
print(paste("Accuracy of caret nearest neighbor using rpart", sum(test_predicted1 == test_labels)/lengt
```

```
## [1] "Accuracy of caret nearest neighbor using rpart 0.623"
```

```
##confusion matrix for rpart on test images
confusionMatrix(test_predicted1, test_labels)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0  84   2   8   8   0   8   2   1   2   0
##          1   0  64   3   4   0   1   0   1   8   2
##          2   0   8  50   3   1   1   5   0   6   0
##          3   1   5   5  60   0  11   0   0   5   5
##          4   2   1   0   3  62   9   4   1   3   3
##          5   1   1   1  10   4  31   7   0   5   7
##          6   6   3  11   8   7  10  53   1  14   2
##          7   5   9   8   9   4   4   0 100   6  11
##          8   1   1   2   2   0   0   2   1  40   0
##          9   2   1   6   7  23   8  14   9  12  79
##
## Overall Statistics
```

```
##
##                Accuracy : 0.623
##                  95% CI : (0.5921, 0.6531)
##     No Information Rate : 0.114
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5803
##  Mcnemar's Test P-Value : 2.406e-14
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.8235   0.6737   0.5319   0.5263   0.6139   0.3735
## Specificity            0.9655   0.9790   0.9735   0.9639   0.9711   0.9607
## Pos Pred Value         0.7304   0.7711   0.6757   0.6522   0.7045   0.4627
## Neg Pred Value         0.9797   0.9662   0.9525   0.9405   0.9572   0.9443
## Prevalence             0.1020   0.0950   0.0940   0.1140   0.1010   0.0830
## Detection Rate         0.0840   0.0640   0.0500   0.0600   0.0620   0.0310
## Detection Prevalence   0.1150   0.0830   0.0740   0.0920   0.0880   0.0670
## Balanced Accuracy      0.8945   0.8263   0.7527   0.7451   0.7925   0.6671
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.6092   0.8772   0.3960   0.7248
## Specificity            0.9321   0.9368   0.9900   0.9080
## Pos Pred Value         0.4609   0.6410   0.8163   0.4907
## Neg Pred Value         0.9616   0.9834   0.9359   0.9642
## Prevalence             0.0870   0.1140   0.1010   0.1090
## Detection Rate         0.0530   0.1000   0.0400   0.0790
## Detection Prevalence   0.1150   0.1560   0.0490   0.1610
## Balanced Accuracy      0.7706   0.9070   0.6930   0.8164
```

```
##Accuracy of training images when using rpart method
test_predicted2 = predict(rpfit, data.frame(train_img))
print(paste("Accuracy of caret nearest neighbor using rpart", sum(test_predicted2 == test_labels)/lengt
```

```
## [1] "Accuracy of caret nearest neighbor using rpart 0.31"
```

```
##confusion matrix for rpart on train images
confusionMatrix(test_predicted2, train_labels)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 239   0  27  14   2  21  12   1   7   0
##          1   1 287   8   6   1   5   0  10  18   2
##          2   1  13 151  10   2   4  17   5  12   1
##          3   2   5  24 194   9  37   5   6  17   8
##          4   1   0   4   8 197  18  18   7  14  11
##          5  10   2   6  23   7 125  16   5  15  30
##          6  10   4  36  15  27  32 184   0  48  13
##          7  10  19   9  16   5  15   1 261  12  21
##          8   4   5   6  13   0   6   4   2 124   0
##          9  14   6  29   8  30  24  17  20  27 222
##
## Overall Statistics
##
```

```
##                 Accuracy : 0.6613
##                   95% CI : (0.6441, 0.6783)
##      No Information Rate : 0.1137
##      P-Value [Acc > NIR] : < 2.2e-16
##
##                    Kappa : 0.6235
##   Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.81849  0.84164  0.50333  0.63192  0.70357  0.43554
## Specificity           0.96898  0.98082  0.97593  0.95804  0.97022  0.95798
## Pos Pred Value        0.73994  0.84911  0.69907  0.63192  0.70863  0.52301
## Neg Pred Value        0.98020  0.97971  0.94648  0.95804  0.96951  0.94133
## Prevalence            0.09733  0.11367  0.10000  0.10233  0.09333  0.09567
## Detection Rate        0.07967  0.09567  0.05033  0.06467  0.06567  0.04167
## Detection Prevalence  0.10767  0.11267  0.07200  0.10233  0.09267  0.07967
## Balanced Accuracy     0.89374  0.91123  0.73963  0.79498  0.83690  0.69676
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.67153   0.8233  0.42177   0.7208
## Specificity           0.93213   0.9597  0.98522   0.9350
## Pos Pred Value        0.49864   0.7073  0.75610   0.5592
## Neg Pred Value        0.96579   0.9787  0.94006   0.9670
## Prevalence            0.09133   0.1057  0.09800   0.1027
## Detection Rate        0.06133   0.0870  0.04133   0.0740
## Detection Prevalence  0.12300   0.1230  0.05467   0.1323
## Balanced Accuracy     0.80183   0.8915  0.70349   0.8279
```

Summarise the results by nicely showing all accuracies you calculated

(on both training and testing data). Comment on the results. Which model worked the best?

```
print(paste("Accuracy of test images using rpart =", sum(test_predicted1 == test_labels)/length(test_la
```

```
## [1] "Accuracy of test images using rpart = 0.623"
```

```
print(paste("Accuracy of train image using rpart =", sum(test_predicted2 == test_labels)/length(test_la
```

```
## [1] "Accuracy of train image using rpart = 0.31"
```

- From the result Knn model worked the best since it has a high proportions compared to rpart model