Project 2

Sumaiya Uddin

2023-01-22

### Setting the seed

Setting a seed will cause R to sample the same sample each time we knit our document. This will make sure our results don’t change each time we knit, and it will also ensure reproducibility of our work (by setting the same seed it will be possible to reproduce our results).

set.seed(29071995)

### Load packages

In this lab we will explore the data using the ‘dplyr’ package and visualize it using the ‘ggplot2’ package for data visualization. Let’s load the packages.

library(dplyr)  
library(tools)  
library(caTools)  
library(graphics)  
library(ggplot2)  
library(ggfortify)  
library(caret)  
library(tidyverse)

### The data

MNIST (“Modified National Institute of Standards and Technology”) is the de fact “hello world” data set of computer vision. Since its release in 1999, this classic data set of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike.([https://en.wikipedia.org/wiki/MNIST\_database).](https://en.wikipedia.org/wiki/MNIST_database).)

It is a data set of 60,000 small square 28×28 pixel gray scale images of handwritten single digits between 0 and 9 (c.f. Fig.1). The task is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively.

Let’s load and read the Data.

load\_image\_file = function(filename) {  
 ret = list()  
 f = file(filename, 'rb')  
 readBin(f, 'integer', n = 1, size = 4, endian = 'big')  
 n = readBin(f, 'integer', n = 1, size = 4, endian = 'big')  
 nrow = readBin(f, 'integer', n = 1, size = 4, endian = 'big')  
 ncol = readBin(f, 'integer', n = 1, size = 4, endian = 'big')  
 x = readBin(f, 'integer', n = n\*nrow\*ncol, size = 1, signed = FALSE)  
 close(f)  
 data.frame(matrix(x, ncol = nrow\*ncol, byrow = TRUE))  
}  
  
load\_label\_file = function(filename) {  
 f = file(filename, 'rb')  
 readBin(f, 'integer', n = 1, size = 4, endian = 'big')  
 n = readBin(f, 'integer', n = 1, size = 4, endian = 'big')  
 L = readBin(f, 'integer', n = n, size = 1, signed = FALSE)  
 close(f)  
 L  
}  
  
# set working directory  
  
setwd("C:/Users/sumai/OneDrive/Documents/CCNY/Semister-01/Statistics/Project2/Project2Regressions")  
# load images and corresponding labels   
train\_digits = load\_image\_file('train-images.idx3-ubyte')  
train\_Labels = load\_label\_file('train-labels.idx1-ubyte')  
  
mnist <- cbind.data.frame(train\_digits, train\_Labels)  
dim(mnist)

## [1] 60000 785

# Part-01

Find a least-squares binary classifier for handwritten MNIST digit set, i.e. determine if an image 𝑥 is a digit 𝑘 or not digit 𝑘. Pick a digit you like. First, extract the indices of all digit’s with label 𝑘 and randomly separate the samples into equal-sized training and testing groups. Second, do the same for the digits that are not labeled 𝑘. Use label 𝑦$ = 1 if 𝑥$ is digit 𝑘 and 𝑦$ = −1 otherwise. Find 𝛽, and 𝛽 , such that 𝑦$ ≈ 𝑦+$ = 𝐬𝐢𝐠𝐧 (𝛽2𝑥$ + 𝛽)) = 𝐬𝐢𝐠𝐧(𝑦5$). Note that 𝛽 can be visualized as a 2D image of the same 28x28 size as the digits in the MNIST data set. Display it. Compute the classification error rate and confusion matrices for the both the training and test sets. Reduce the number of parameters in the 𝛽 using backward selection methods. Display 𝛽 for the reduced model. Use the plot() function to run a series of diagnostic tests for your regression. Plot “Residuals vs Fitted”, “Normal Q-Q”, “Scale-Location”, and “Residuals vs Leverage” plots.

Checking for unique labels in train labels. Checking to see if the labels are within the expected values.

unique(unlist(train\_Labels))

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

### Binary Classification

We will generate a training and test data set of equal size from the data set. From here, I selected a digit (i.e. k=0) and converted the k labels in the target variable to ‘+1’ and non-k labels to ‘-1’. We will implement a Least Squares method to find the Beta coefficients and classify labels using a sign function. First let’s Define the labels:

# Select digit k and re-label the dataset wrt selected digit  
k = 0  
df <- data.frame(labels = train\_Labels)  
df <- df %>% mutate(ml\_labels = case\_when(labels == k ~ 1,  
 labels !=k ~ -1))  
  
df\_labels <- select(df, -c(labels))  
  
# combine into one dataset  
mnist\_df <- cbind.data.frame(train\_digits, df\_labels)  
dim(mnist\_df)

## [1] 60000 785

Here we created a new data frame with new labels. Now we will split it into training and test subset of equal size. And then we will perform a simple linear regression for both training and test subset. Linear regression models are used to describe the relationship between one or more predictor variables and a response variable.

# Spliting 50% of dataset as training set and 50% as test set  
digits <- sample.split(Y = mnist\_df , SplitRatio =0.5)  
train\_df <- mnist\_df[digits, ]  
test\_df <- mnist\_df[!digits, ]  
   
# running the model for train set.  
model\_train <- lm(ml\_labels ~ . , data = train\_df )

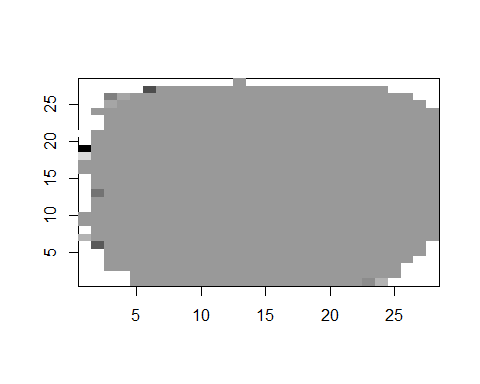
### Display Beta Coefficients

The regression coefficients in linear regression help in predicting the value of an unknown variable using a known variable. Regression coefficients can be defined as estimates of some unknown parameters to describe the relationship between a predictor variable and the corresponding response. In other words, regression coefficients are used to predict the value of an unknown variable using a known variable. For a simple linear regression model:

Y = β0 + β1 X + ε

β1 is the expected change in the outcome Y per unit change in X. Therefore, increasing the predictor X by 1 unit (or going from 1 level to the next) is associated with an increase in Y by β1 units. It can be visualized as a 2D image of the same 28x28 size as the digits in the MNIST data set.

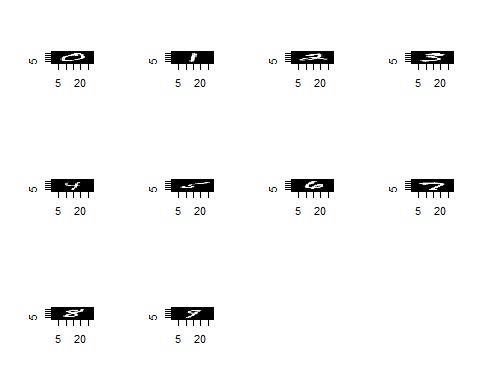
#computing beta for train and showing as image  
coeff\_train <- model\_train$coefficients[2:785]  
  
image(1:28, 1:28, matrix(as.matrix(coeff\_train), nrow=28)[ , 28:1],   
 col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")



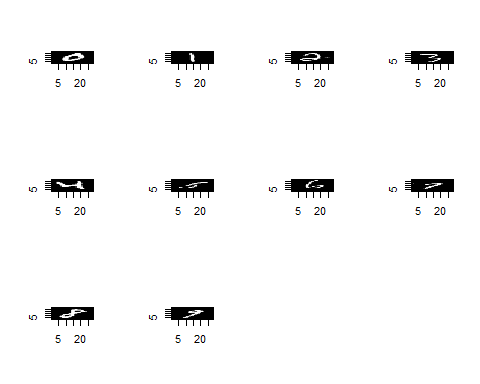
### Display Digits

Now let’s take a look at 0-9 digits for both training and test subset.

# Showing 0-9 digits from treain set  
train\_ind <- c(2,20,3,24,14,8,19,25,21,22)  
par(mfrow=c(3,4))  
for (j in train\_ind) {  
 image(1:28, 1:28, matrix(as.matrix(train\_df[j,]), nrow=28)[ , 28:1],   
 col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")  
   
}  
  
# Showing 0-9 digits from test set  
test\_ind <- c(20,7,9,23,12,19,17,15,16,18)  
par(mfrow=c(3,4))



for (j in test\_ind) {  
 image(1:28, 1:28, matrix(as.matrix(test\_df[j,]), nrow=28)[ , 28:1],   
 col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")  
   
}



### Prediction

Let’s see some prediction for our chosen digit (k=0) for both training and test subset. We fit the model with sign() function for prediction and then we will compare it with actual labels.

# Prediction with train set  
train\_prediction <- sign(predict(model\_train, train\_df[,1:784]))  
train\_prediction[train\_ind]

## 2 41 6 50 27 12 40 53 42 44   
## 1 -1 -1 -1 -1 -1 -1 -1 -1 -1

train\_actual <- sign(train\_df$ml\_labels)  
train\_actual[train\_ind]

## [1] 1 -1 -1 -1 -1 -1 -1 -1 -1 -1

# Prediction with test set  
test\_prediction <- sign(predict(model\_train, test\_df[,1:784]))  
test\_prediction[test\_ind]

## 38 15 17 45 21 36 33 30 32 34   
## 1 -1 -1 -1 -1 -1 -1 -1 -1 -1

test\_actual <- sign(test\_df$ml\_labels)  
test\_actual[test\_ind]

## [1] 1 -1 -1 -1 -1 -1 -1 -1 -1 -1

### Classification Error Rate

The overall classification accuracy rate corresponds to the proportion of observations that have been correctly classified. Determining the raw classification accuracy is the first step in assessing the performance of a model. Inversely, the classification error rate is defined as the proportion of observations that have been misclassified.

Error rate = 1 - accuracy

The classification accuracy and error can be easily computed by comparing with prediction and actual. We will compute classification error rate for both training and test subset.

# Compute the classification error rate for the train set  
train\_accuracy = mean(train\_prediction == train\_actual)  
train\_accuracy

## [1] 0.9846837

train\_error <- mean(train\_prediction != train\_actual)  
train\_error

## [1] 0.01531634

# Classification error rate for the test set  
test\_accuracy = mean(test\_prediction == test\_actual)  
test\_accuracy

## [1] 0.9835842

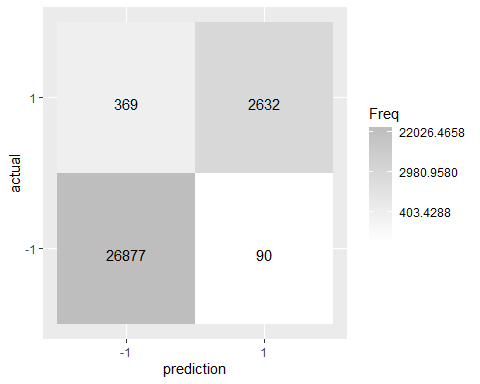
test\_error <- mean(test\_prediction != test\_actual)  
test\_error

## [1] 0.01641582

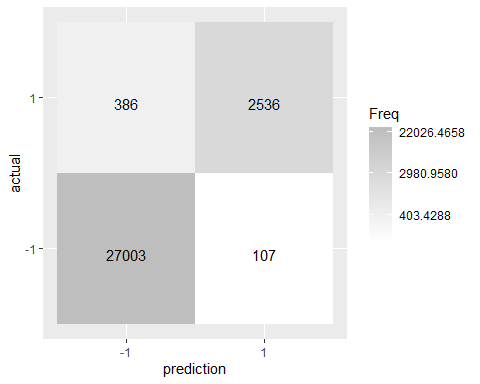
### Confusion Matrix

Confusion matrix is 2x2 table showing four parameters, including the number of true positives, true negatives, false negatives and false positives. The R function table() can be used to produce a confusion matrix in order to determine how many observations were correctly or incorrectly classified. It compares the observed and the predicted outcome values and shows the number of correct and incorrect predictions categorized by type of outcome.

# confusion matrix for train set  
train\_cm\_lr = as.matrix(table(prediction = sign(train\_prediction), actual= sign(train\_actual)))   
  
ggplot(data = train\_cm\_lr,  
 mapping = aes(x = prediction,  
 y = actual)) +  
 geom\_tile(aes(fill = Freq)) +  
 geom\_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +  
 scale\_fill\_gradient(low = "white",  
 high = "gray",  
 trans = "log")



# confusion matrix for test set  
test\_cm\_lr = as.matrix(table(prediction = sign(test\_prediction), actual = sign(test\_actual)))   
  
ggplot(data = test\_cm\_lr,  
 mapping = aes(x = prediction,  
 y = actual)) +  
 geom\_tile(aes(fill = Freq)) +  
 geom\_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +  
 scale\_fill\_gradient(low = "white",  
 high = "gray",  
 trans = "log")

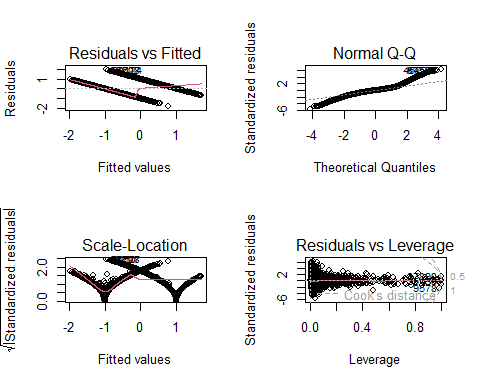


### Diagnostic Test

After running a regression analysis, you should check if the model works well for data. It’s a good idea to also produce diagnostic plot to analyze the residuals of the model and make sure that a linear model is appropriate to use for the particular data we’re working with. These diagnostics include:

1. Residuals vs. fitted values
2. Q-Q plots
3. Scale Location plots
4. Residuals vs. Leverage

# diagnostic tests for the train set  
par(mfrow = c(2, 2))  
plot(model\_train)



### Backward selection

Princple Component Analysis was used to reduce the dimensionality of the data. By reducing the dimensionality of the data set some of the variation is ineviteably lost. We either directly chose how many variables that we want to keep, and then sum over that many indicies of the vector ‘D’ from the singular value decompostion; or we can choose an amount of variation, and determine how many varibles need to be kepted to maintain the percentage of variation within the data. Here the variation is set to .99. This cut the number of variables almost in half, from 784 down to 399.

## Principle Component Analysis for Variable Reduction ----  
  
Labels = as.factor(mnist$train\_Labels)  
# Transform data to be on a 0 - 1 scale  
mnist[,-1] <- (mnist[,-1]/255)  
  
# create covariance matrix  
Sigma <- var(as.matrix(mnist[,-1]))   
diag(Sigma) <- 0  
  
#Singular Value Decomposition  
D <- svd(Sigma)$d   
U <- svd(Sigma)$u  
  
#Value deciding how much variation to retain in the PCA  
variation <- .99  
  
#Figure out how many variables (k) to include to keep that level of variation  
for (k in seq(length(D)))  
{  
 if( (sum(D[1:k])/sum(D)) >= variation){break}  
}  
  
#Reduce U matrix to down to 'k' columns  
U <- U[,1:k]  
  
# Use U matrix and data to map data to lower dimension  
ReducedData <- as.matrix(mnist[,-1]) %\*% U  
  
# Replace Transformed Variables into Training Data Set  
mnist\_r <- data.frame(mnist[,1],ReducedData)  
names(mnist\_r)[1] <- "label"  
dim(mnist\_r)

## [1] 60000 400

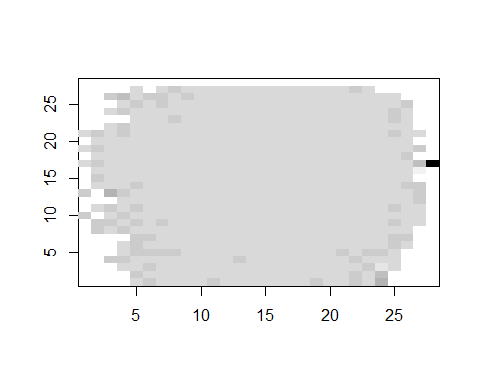
# Part-02

Repeat the above problem for all pairs of digits. For each pair of digits, report the classification error rates for the training and testing sets. The error rates can be formatted nicely into a triangular matrix. For storage and display efficiency, store the testing error in the lower triangle and the training error in the upper triangle. Display 𝛽 and 𝛽) as a 2D images for pairs with lowest and highest error rates.

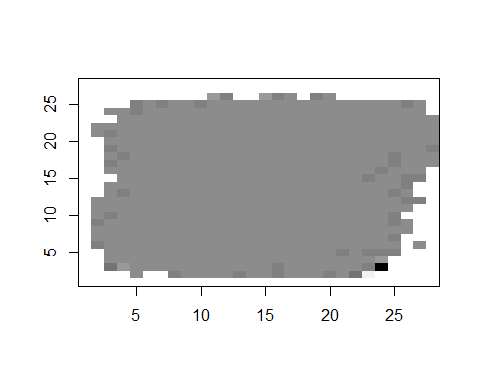
error\_matrix <- matrix(nrow = 10, ncol = 10,dimnames =list(0:9,0:9) )  
  
dataset <- train\_digits  
dataset$y <- train\_Labels  
  
complete\_dataset <- dataset  
#model\_matrix <- matrix(nrow = 10,ncol = 10,dimnames = list(0:9,0:9))  
min\_error <- 1  
max\_error <- 0  
min\_model <- NA  
max\_model <- NA  
sample <- sample.split(complete\_dataset$y, SplitRatio = 0.5)  
train <- subset(complete\_dataset, sample == TRUE)  
test <- subset(complete\_dataset, sample == FALSE)  
for (r in 0:9){  
 val = r+1  
 if (val<=9){  
 for (c in val:9){  
 #k=5  
 train\_temp <- subset(train, y %in% c(r, c))  
 test\_temp <- test[which(test$y==r | test$y==c),]  
 train\_temp <- train\_temp %>% mutate(y = case\_when(y == r ~ 1,  
 y == c ~ -1))  
 test\_temp <- test\_temp %>% mutate(y = case\_when(y == r ~ 1,  
 y == c ~ -1))  
   
 model\_temp <- lm(y ~ ., data = train\_temp)  
 print(paste("Model for r=",c(r,c)))  
 predicted\_train\_temp <- sign(predict(model\_temp, train\_temp[,1:784]))  
 predicted\_test\_temp <- sign(predict(model\_temp, test\_temp[,1:784]))  
 #print(paste(predicted\_train\_temp))  
 tmpp1 = confusionMatrix(data=factor(predicted\_train\_temp),reference = factor(train\_temp$y))  
 error\_matrix[r+1,c+1] = 1-tmpp1$overall[1]  
 tmpp2 = confusionMatrix(data=factor(predicted\_test\_temp),reference = factor(test\_temp$y))  
 error\_matrix[c+1,r+1] = 1-tmpp2$overall[1]  
 error = 1-tmpp2$overall[1]  
 if (error<min\_error){  
 min\_error = error  
 min\_model <- model\_temp  
 }  
 if (error>max\_error){  
 max\_error=error  
 max\_model = model\_temp  
 }  
   
 #print(head(df2$y))  
 # print(paste("r",r,'c',c))  
   
 }  
   
 }  
   
}

## [1] "Model for r= 0" "Model for r= 1"  
## [1] "Model for r= 0" "Model for r= 2"  
## [1] "Model for r= 0" "Model for r= 3"  
## [1] "Model for r= 0" "Model for r= 4"  
## [1] "Model for r= 0" "Model for r= 5"  
## [1] "Model for r= 0" "Model for r= 6"  
## [1] "Model for r= 0" "Model for r= 7"  
## [1] "Model for r= 0" "Model for r= 8"  
## [1] "Model for r= 0" "Model for r= 9"  
## [1] "Model for r= 1" "Model for r= 2"  
## [1] "Model for r= 1" "Model for r= 3"  
## [1] "Model for r= 1" "Model for r= 4"  
## [1] "Model for r= 1" "Model for r= 5"  
## [1] "Model for r= 1" "Model for r= 6"  
## [1] "Model for r= 1" "Model for r= 7"  
## [1] "Model for r= 1" "Model for r= 8"  
## [1] "Model for r= 1" "Model for r= 9"  
## [1] "Model for r= 2" "Model for r= 3"  
## [1] "Model for r= 2" "Model for r= 4"  
## [1] "Model for r= 2" "Model for r= 5"  
## [1] "Model for r= 2" "Model for r= 6"  
## [1] "Model for r= 2" "Model for r= 7"  
## [1] "Model for r= 2" "Model for r= 8"  
## [1] "Model for r= 2" "Model for r= 9"  
## [1] "Model for r= 3" "Model for r= 4"  
## [1] "Model for r= 3" "Model for r= 5"  
## [1] "Model for r= 3" "Model for r= 6"  
## [1] "Model for r= 3" "Model for r= 7"  
## [1] "Model for r= 3" "Model for r= 8"  
## [1] "Model for r= 3" "Model for r= 9"  
## [1] "Model for r= 4" "Model for r= 5"  
## [1] "Model for r= 4" "Model for r= 6"  
## [1] "Model for r= 4" "Model for r= 7"  
## [1] "Model for r= 4" "Model for r= 8"  
## [1] "Model for r= 4" "Model for r= 9"  
## [1] "Model for r= 5" "Model for r= 6"  
## [1] "Model for r= 5" "Model for r= 7"  
## [1] "Model for r= 5" "Model for r= 8"  
## [1] "Model for r= 5" "Model for r= 9"  
## [1] "Model for r= 6" "Model for r= 7"  
## [1] "Model for r= 6" "Model for r= 8"  
## [1] "Model for r= 6" "Model for r= 9"  
## [1] "Model for r= 7" "Model for r= 8"  
## [1] "Model for r= 7" "Model for r= 9"  
## [1] "Model for r= 8" "Model for r= 9"

#computing beta for min model and showing as image  
coeff\_min <- min\_model$coefficients[2:785]  
  
image(1:28, 1:28, matrix(as.matrix(coeff\_min), nrow=28)[ , 28:1],   
 col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")



#computing beta for max model and showing as image  
coeff\_max <- max\_model$coefficients[2:785]  
  
image(1:28, 1:28, matrix(as.matrix(coeff\_max), nrow=28)[ , 28:1],  
 col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")



print(error\_matrix)

## 0 1 2 3 4 5  
## 0 NA 0.004263382 0.01009931 0.005640345 0.003399626 0.010401975  
## 1 0.008686039 NA 0.01196850 0.008233649 0.002383980 0.008386778  
## 2 0.018350168 0.022204724 NA 0.018527709 0.009322034 0.014941114  
## 3 0.011450382 0.018645121 0.03904699 NA 0.003507600 0.031682825  
## 4 0.007990479 0.009853783 0.02220339 0.014032743 NA 0.007991476  
## 5 0.020098731 0.011180533 0.02530756 0.053497230 0.015802557 NA  
## 6 0.014864865 0.010110585 0.02896598 0.011952191 0.012244898 0.028571429  
## 7 0.007712504 0.012915129 0.02094241 0.020167796 0.020812686 0.014202601  
## 8 0.015460415 0.039707751 0.03675474 0.042404007 0.014197742 0.045954578  
## 9 0.009433962 0.009612354 0.01813907 0.024668874 0.044945726 0.017059444  
## 6 7 8 9  
## 0 0.006417835 0.0026255333 0.009341033 0.004716981  
## 1 0.001737757 0.0055359065 0.022868032 0.003309693  
## 2 0.009599192 0.0126002291 0.023708721 0.007055266  
## 3 0.003485477 0.0093578574 0.027202937 0.015562914  
## 4 0.004591837 0.0110688915 0.004446725 0.028498728  
## 5 0.019756571 0.0037658336 0.028743790 0.010731879  
## 6 NA 0.0006567066 0.012064571 0.001854037  
## 7 0.004267892 NA 0.007263123 0.031772028  
## 8 0.018864718 0.0161769561 NA 0.016779661  
## 9 0.005055612 0.0484610347 0.025932203 NA

# Part-03

Use logistic regression and compare your results with Parts 1 and 2.

We will make a Data frame using original train\_digits and train\_Labels. Then Transform target variable “label” from integer to factor, in order to perform classification. Then we will split the data frame into two equal size training and test subset.

# Transform target variable "label" from integer to factor, in order to perform classification  
mnist\_lg <- cbind.data.frame(train\_digits, train\_Labels)  
dim(mnist\_lg)

## [1] 60000 785

train\_Labels = as.factor(mnist\_lg$train\_Labels)  
  
# Spliting 50% of dataset as training set and 50% as test set  
  
digits2 <- sample.split(Y = mnist\_lg , SplitRatio =0.5)  
  
train\_lg <- mnist\_lg[digits2, ]  
  
test\_lg <- mnist\_lg[!digits2, ]  
  
dim(train\_lg)

## [1] 29954 785

dim(test\_lg)

## [1] 30046 785

### logistic regression

Now we will do the multinomial logistic regression as it is a multiclass case. We will implement the multinomial logistic regression model using the nnet package with parameters which include:

1. **MaxNWts=10000:** It allows at most 10,000 weights. In our case, there are(784 dimensions + 1 bias) \* 10 classes = 7850 elements in the weight matrix w.
2. **decay=5e-3:** The regularization strength, the weight decay is 0.005
3. **maxit=100:** The maximum number of iteration is set to be 100

library(nnet)  
# Multinomial logistic regression for train set.  
train\_model\_lg <- multinom(train\_Labels ~ ., data = train\_lg, MaxNWts=10000, decay=5e-3, maxit=100)

## # weights: 7860 (7065 variable)  
## initial value 68971.633876   
## iter 10 value 18295.695104  
## iter 20 value 12461.123220  
## iter 30 value 10666.159300  
## iter 40 value 9963.160794  
## iter 50 value 9548.517678  
## iter 60 value 9283.640475  
## iter 70 value 9118.047498  
## iter 80 value 9008.866368  
## iter 90 value 8912.993781  
## iter 100 value 8809.421696  
## final value 8809.421696   
## stopped after 100 iterations

The error value is printed for every 10 iterations, and it is decreasing. The model converges as the maximum number of iterations is reached.

### Prediction and Classification Error

Take a look at the prediction results of the first ten sample and their true values. And then compute classification error rate. We use the training model to predict the classes of the the tesing samples.

# Prediction and classification error for training set.  
train\_prediction\_lg <- predict(train\_model\_lg, test\_lg, type = "class")  
train\_prediction\_lg[1:10]

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

train\_actual\_lg <- test\_lg$train\_Labels  
train\_actual\_lg[1:10]

## [1] 4 1 2 1 4 3 5 3 1 7

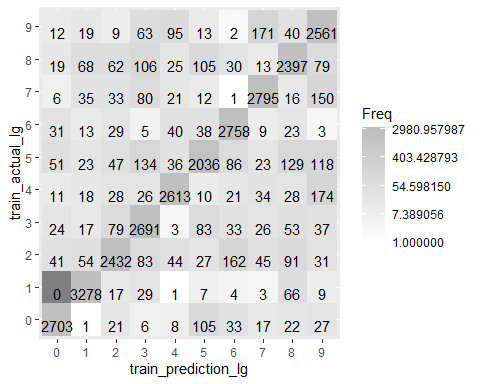
train\_error\_lg <- mean(train\_prediction\_lg != train\_actual\_lg)  
train\_error\_lg

## [1] 0.1258737

### Confusion Matrix

Now we can also obtain the confusion matrix.

# Confusion matrix for training set.  
train\_cm\_lg = table(train\_prediction\_lg, train\_actual\_lg)  
  
ggplot(data = train\_cm\_lg,  
 mapping = aes(x = train\_prediction\_lg,  
 y = train\_actual\_lg)) +  
 geom\_tile(aes(fill = Freq)) +  
 geom\_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +  
 scale\_fill\_gradient(low = "white",  
 high = "gray",  
 trans = "log")



# Part-04

Test for outliers in each group of 10 digits using Cook’s distance. Visualize the ‘mean’ and identified outliers for each digit. Repeat Parts 1 - 3 with outliers removed.

### **Cooks Distance**

Cook’s distance is the scaled change in fitted values, which is useful for identifying outliers in the *X* values (observations for predictor variables). Cook’s distance shows the influence of each observation on the fitted response values. An observation with Cook’s distance larger than three times the mean Cook’s distance might be an outlier.

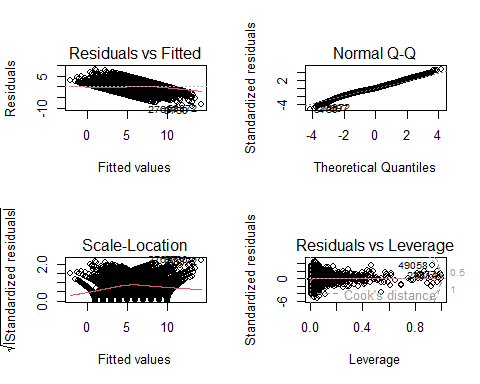
library(ISLR)  
# We are going to run a linear regression on train sub set.  
mnist <- cbind.data.frame(train\_digits, train\_Labels)  
dim(mnist)

## [1] 60000 785

mnist$train\_Labels =as.numeric(mnist$train\_Labels)  
# Spliting 50% of dataset as training set and 50% as test set  
digits\_2 <- sample.split(mnist$train\_Labels , SplitRatio =0.5)  
train\_2 <- mnist[digits\_2, ]  
test\_2 <- mnist[!digits\_2, ]  
  
model <- lm(train\_Labels ~ . , data = train\_2)

Let’s display diagnostic plots.

par(mfrow = c(2,2))  
plot(model)

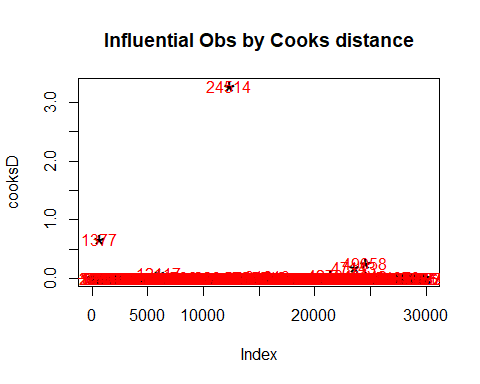


let’s find out outliear with cook’s distance

cooksD <- cooks.distance(model)  
influential <- as.numeric(names(cooksD)[(cooksD > 3 \* mean(cooksD, na.rm = TRUE))])

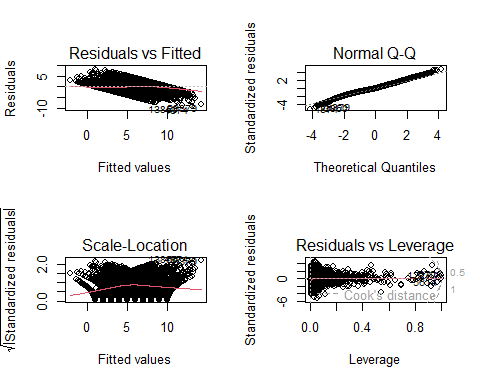
Visualizing the ‘mean’ and identified outliers.

plot(cooksD, pch="\*", cex=2, main="Influential Obs by Cooks distance") # plot cook's distance  
abline(h = 3\*mean(cooksD, na.rm=T), col="red") # add cutoff line  
text(x=1:length(cooksD)+1, y=cooksD, labels=ifelse(cooksD>3\*mean(cooksD, na.rm=T),names(cooksD),""), col="red")



Now we will remove outliers and then again run a model without outliers. also we will display diagnostic plots to see changes.

outliers <- train\_2[influential, ]  
  
# Removing outliers  
train\_no\_outliers <- train\_2 %>% anti\_join(outliers)  
  
model\_no\_ouliers <- lm(train\_Labels ~ . , data = train\_no\_outliers)  
  
par(mfrow = c(2,2))  
plot(model\_no\_ouliers)



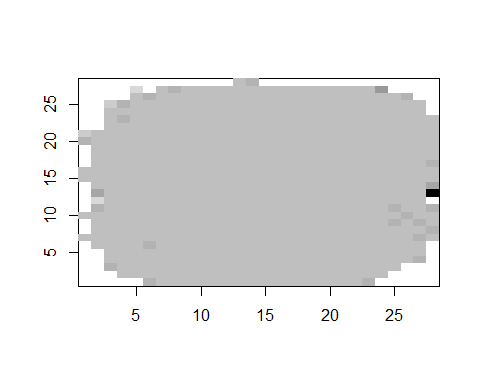
Now we will select a digit (i.e. k=0) and converted the k labels in the target variable to ‘+1’ and non-k labels to ‘-1’. We will implement a Least Squares method to find the Beta coefficients and classify labels using a sign function. First let’s Define the labels:

# Select digit k and re-label the dataset wrt selected digit  
k = 0  
train\_no\_outliers <- train\_no\_outliers %>%   
 mutate(ml\_labels = case\_when(train\_Labels == k ~ 1,  
 train\_Labels !=k ~ -1))  
  
train\_no\_outliers2 <- select(train\_no\_outliers, -c(train\_Labels))  
dim(train\_no\_outliers2)

## [1] 29919 785

Running linear model and displaying beta coefficient.

model\_no\_outliers <- lm(ml\_labels ~ . , data = train\_no\_outliers2)  
  
#computing beta for no outliers model showing as image  
coeff\_no\_outliers <- model\_no\_outliers$coefficients[2:785]  
  
image(1:28, 1:28, matrix(as.matrix(coeff\_no\_outliers), nrow=28)[ , 28:1],   
 col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")



prediction\_no\_outliers <- sign(predict(model\_no\_outliers))  
prediction\_no\_outliers[1:10]

## 1 2 3 4 5 6 7 8 9 10   
## -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

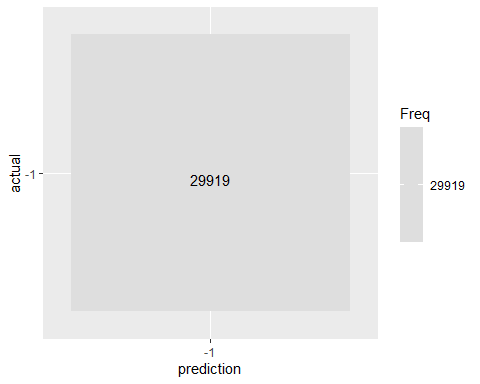
actual\_no\_outliers <- train\_no\_outliers2$ml\_labels  
actual\_no\_outliers[1:10]

## [1] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

# Compute the classification error rate.  
error\_no\_outliers <- mean(prediction\_no\_outliers != actual\_no\_outliers)  
error\_no\_outliers

## [1] 0

# confusion matrix for train set  
cm\_no\_outliers = as.matrix(table(prediction = sign(prediction\_no\_outliers), actual= sign(actual\_no\_outliers)))   
  
ggplot(data = cm\_no\_outliers,  
 mapping = aes(x = prediction,  
 y = actual)) +  
 geom\_tile(aes(fill = Freq)) +  
 geom\_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +  
 scale\_fill\_gradient(low = "white",  
 high = "gray",  
 trans = "log")



Now we will do logistic regression for no outliers. we will run model on train set and test it on testing set.

# Transform target variable "label" from integer to factor, in order to perform classification  
lg\_no\_outliers <- select(train\_no\_outliers, -c(ml\_labels))  
  
lg\_Labels = as.factor(lg\_no\_outliers$train\_Labels)  
  
library(nnet)  
# Multinomial logistic regression for train set.  
model\_lg\_no <- multinom(lg\_Labels ~ ., data = lg\_no\_outliers, MaxNWts=10000, decay=5e-3, maxit=100)

## # weights: 7870 (7074 variable)  
## initial value 68891.043397   
## iter 10 value 17127.090667  
## iter 20 value 11452.391733  
## iter 30 value 9711.226887  
## iter 40 value 9041.858481  
## iter 50 value 8647.835367  
## iter 60 value 8372.340966  
## iter 70 value 8176.476497  
## iter 80 value 8067.838171  
## iter 90 value 7927.715416  
## iter 100 value 7768.698423  
## final value 7768.698423   
## stopped after 100 iterations

# Prediction and classification error.  
prediction\_lg <- predict(model\_lg\_no, test\_2, type = "class")  
prediction\_lg[1:10]

## [1] 4 1 4 2 10 2 4 5 4 9   
## Levels: 1 2 3 4 5 6 7 8 9 10

actual\_lg <- test\_2$train\_Labels  
actual\_lg[1:10]

## [1] 6 1 5 2 10 2 4 5 4 6

error\_lg <- mean(prediction\_lg != actual\_lg)  
error\_lg

## [1] 0.1143

# Confusion matrix for training set.  
cm\_lg = table(prediction\_lg, actual\_lg)  
  
ggplot(data = cm\_lg,  
 mapping = aes(x = prediction\_lg,  
 y = actual\_lg)) +  
 geom\_tile(aes(fill = Freq)) +  
 geom\_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +  
 scale\_fill\_gradient(low = "white",  
 high = "gray",  
 trans = "log")

