Naive Bayes & Gradient Boosting

Parthasarathi Samantaray

February 7, 2019

Installing and loading the required packages

```
options(repos=structure(c(CRAN="http://cran.stat.ucla.edu/")))

install.packages("Metrics")

## package 'Metrics' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\samantaray.p\AppData\Local\Temp\RtmpK8h8Fc\downloaded_packages

install.packages("tidyverse")

## package 'tidyverse' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\samantaray.p\AppData\Local\Temp\RtmpK8h8Fc\downloaded_packages

install.packages("gbm")

## package 'gbm' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\samantaray.p\AppData\Local\Temp\RtmpK8h8Fc\downloaded_packages

install.packages("caret")

## package 'caret' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\samantaray.p\AppData\Local\Temp\RtmpK8h8Fc\downloaded_packages
```

Introduction

I have loaded the required packages, to use the package functionalities.

```
library(tidyverse)

## Warning: package 'tidyverse' was built under R version 3.5.2

## -- Attaching packages ---------------------------------------------------
----------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.7
## v tidyr   0.8.2     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0

## -- Conflicts ------------------------------------------------------------
----------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

library(data.table)

##
## Attaching package: 'data.table'

library(Metrics)

library(gbm)

library(caret)

library(gdata)

install.packages("caTools")

## package 'caTools' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\samantaray.p\AppData\Local\Temp\RtmpK8h8Fc\downloaded_packages

library(caTools)

## Warning: package 'caTools' was built under R version 3.5.2

set.seed(123)
```

I have saved the datasets mnist_train.csv and mnist_test.csv into tibbles training_set and test_set respectively.Due to limited cmputing power, I will use 60% of the training and testing dataset . However to ensure the propertion of labels remain consistent in the reduced datasets, I am using the split function of caTools package.

```
#Loading the datasets and saving as tibbles
training_set <- fread("mnist_train.csv", sep = ",", header = FALSE)
test_set <- fread("mnist_test.csv", sep = ",", header = FALSE)
training_set <-as.tibble(training_set)
test_set <- as.tibble(test_set)

#Sampling 60% of datasets by ensuring label proportion in the train and test
datasets
training_set %>% dim()

## [1] 60000   785
```

```
test_set %>% dim()

## [1] 10000   785

sampleTrain <-sample.split( training_set$V1, SplitRatio = .6 )
sampleTest <-sample.split( test_set$V1, SplitRatio = .6 )

training_set<- training_set[sampleTrain,]
test_set<- test_set[sampleTest,]

training_set %>% dim()

## [1] 36001   785

test_set %>% dim()

## [1] 5999  785
```

The distribution of labels in the reduced triaining and test datasets as follows:-

```
table(training_set$V1)

##
##    0    1    2    3    4    5    6    7    8    9
## 3554 4045 3575 3679 3505 3253 3551 3759 3511 3569

table(test_set$V1)

##
##   0   1   2   3   4   5   6   7   8   9
## 588 681 619 606 589 535 575 617 584 605
```

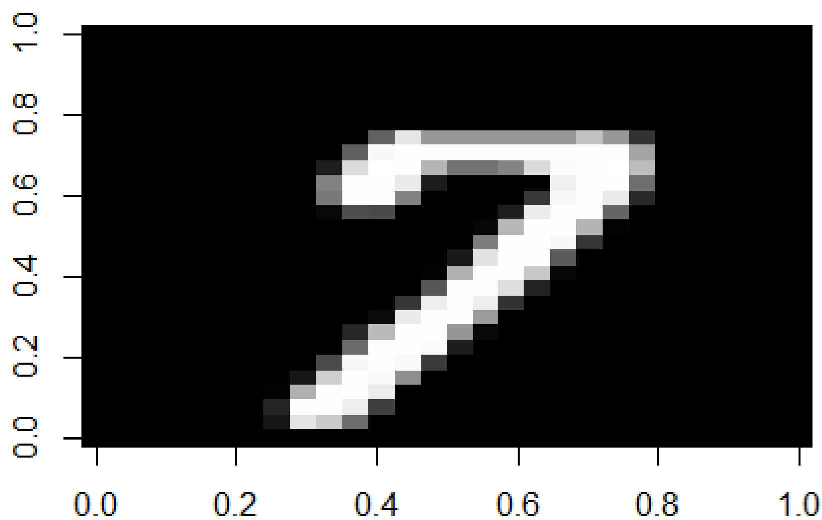
As we are using the same dataset for past 3 week's assignments , I already know the dimensions, and variable names of datasets

```
training_set %>% names()

##   [1] "V1"   "V2"   "V3"   "V4"   "V5"   "V6"   "V7"   "V8"   "V9"   "V10"
##  [11] "V11"  "V12"  "V13"  "V14"  "V15"  "V16"  "V17"  "V18"  "V19"  "V20"
##  [21] "V21"  "V22"  "V23"  "V24"  "V25"  "V26"  "V27"  "V28"  "V29"  "V30"
##  [31] "V31"  "V32"  "V33"  "V34"  "V35"  "V36"  "V37"  "V38"  "V39"  "V40"
##  [41] "V41"  "V42"  "V43"  "V44"  "V45"  "V46"  "V47"  "V48"  "V49"  "V50"
##  [51] "V51"  "V52"  "V53"  "V54"  "V55"  "V56"  "V57"  "V58"  "V59"  "V60"
##  [61] "V61"  "V62"  "V63"  "V64"  "V65"  "V66"  "V67"  "V68"  "V69"  "V70"
##  [71] "V71"  "V72"  "V73"  "V74"  "V75"  "V76"  "V77"  "V78"  "V79"  "V80"
##  [81] "V81"  "V82"  "V83"  "V84"  "V85"  "V86"  "V87"  "V88"  "V89"  "V90"
##  [91] "V91"  "V92"  "V93"  "V94"  "V95"  "V96"  "V97"  "V98"  "V99"  "V100
"
## [771] "V771" "V772" "V773" "V774" "V775" "V776" "V777" "V778" "V779" "V780
"
## [781] "V781" "V782" "V783" "V784" "V785"
```

The below functions visualise the pixels

```r
rotate <- function(x) {
  return(t(apply(x, 2, rev)))
}

plot_matrix <- function(vec) {
  q <- matrix(vec, 28, 28, byrow = TRUE)
  nq <- apply(q, 2, as.numeric)
  image(rotate(nq), col = gray((0:255)/255))
}

plot_matrix(training_set[500,2:784])
```

```
## Warning in matrix(vec, 28, 28, byrow = TRUE): data length [783] is not a
## sub-multiple or multiple of the number of rows [28]
```



I am creating a ideal partial probability matrix . I will deduct the partial probabilities produced from the naive bayes model to get the error terms.

```r
train_ideal_prob <- matrix(rep(0,nrow(training_set)*10),ncol = 10)
head(train_ideal_prob)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    0    0    0    0    0    0    0    0     0
## [2,]    0    0    0    0    0    0    0    0    0     0
## [3,]    0    0    0    0    0    0    0    0    0     0
## [4,]    0    0    0    0    0    0    0    0    0     0
## [5,]    0    0    0    0    0    0    0    0    0     0
## [6,]    0    0    0    0    0    0    0    0    0     0
```

```r
test_ideal_prob <- matrix(rep(0,nrow(test_set)*10),ncol = 10)
head(test_ideal_prob)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    0    0    0    0    0    0    0    0     0
## [2,]    0    0    0    0    0    0    0    0    0     0
## [3,]    0    0    0    0    0    0    0    0    0     0
## [4,]    0    0    0    0    0    0    0    0    0     0
## [5,]    0    0    0    0    0    0    0    0    0     0
## [6,]    0    0    0    0    0    0    0    0    0     0
```

```r
for (i in 1:nrow(training_set)){
  j<- training_set$V1[i]+1
  train_ideal_prob[i,j]<-1
}

for (i in 1:nrow(test_set)){
  j<- test_set$V1[i]+1
  test_ideal_prob[i,j]<-1
}


head(training_set$V1)
```

```
## [1] 5 1 9 2 1 3
```

```r
head(train_ideal_prob)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    0    0    0    0    1    0    0    0     0
## [2,]    0    1    0    0    0    0    0    0    0     0
## [3,]    0    0    0    0    0    0    0    0    0     1
## [4,]    0    0    1    0    0    0    0    0    0     0
## [5,]    0    1    0    0    0    0    0    0    0     0
## [6,]    0    0    0    1    0    0    0    0    0     0
```

```r
head(test_set$V1)
```

```
## [1] 7 2 1 9 5 9
```

```r
head(test_ideal_prob)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    0    0    0    0    0    0    1    0     0
## [2,]    0    0    1    0    0    0    0    0    0     0
## [3,]    0    1    0    0    0    0    0    0    0     0
## [4,]    0    0    0    0    0    0    0    0    0     1
## [5,]    0    0    0    0    0    1    0    0    0     0
## [6,]    0    0    0    0    0    0    0    0    0     1
```

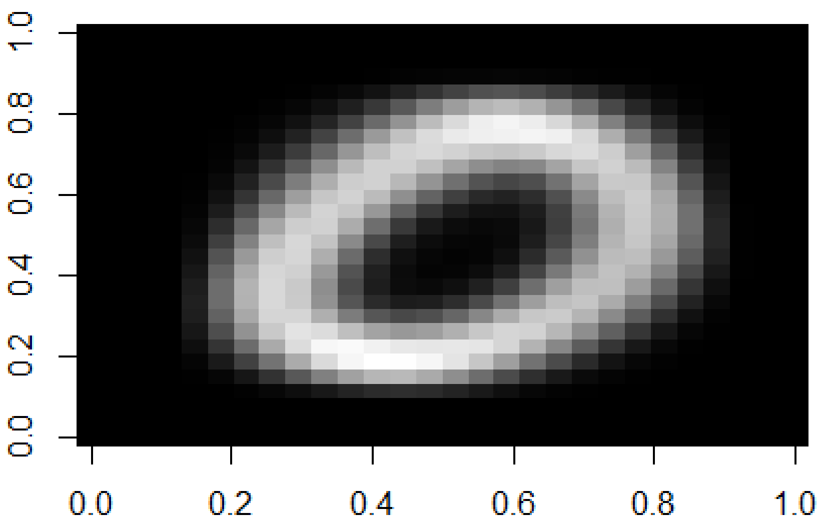
Model1 Naive Bayes (provided with the assignment)

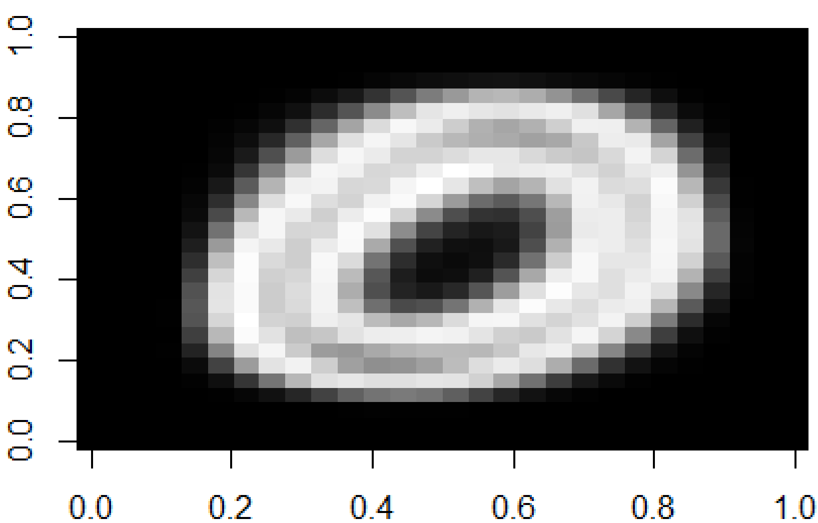
```
## Training function

naive_bayes_training <- function(training_set) {

  ## Calculates priors
  freq <- table(training_set[,1])
  priors <- freq/sum(freq)

  ##Training

  means <- data.frame(label = seq(0,9))
  mu <- matrix(rep(0, 10*784), nrow = 10, ncol = 784)
  means <- cbind(means,mu)

  variances <- data.frame(label = seq(0,9))
  sigma <- matrix(rep(0, 10*784), nrow = 10, ncol = 784)
  variances <- cbind(variances, sigma)

  # For each label, calculate images representing the mean and variances of a
ll the images belonging to a label.
  for (i in 0:9) {
    class_set <- training_set[which(training_set[,1] == i),]

    class_mean <- apply(class_set, 2, mean)
    plot_matrix(class_mean[2:785])
    means[i + 1, 2:785] <- class_mean[2:785]

    class_var <- apply(class_set, 2, var)
    plot_matrix(class_var[2:785])
    variances[i + 1, 2:785] <- class_var[2:785]
  }

  #Returns data "ingredients" that when put together form a Naive Bayes model
  return(list(priors = priors, means = means, variances = variances))

}
```

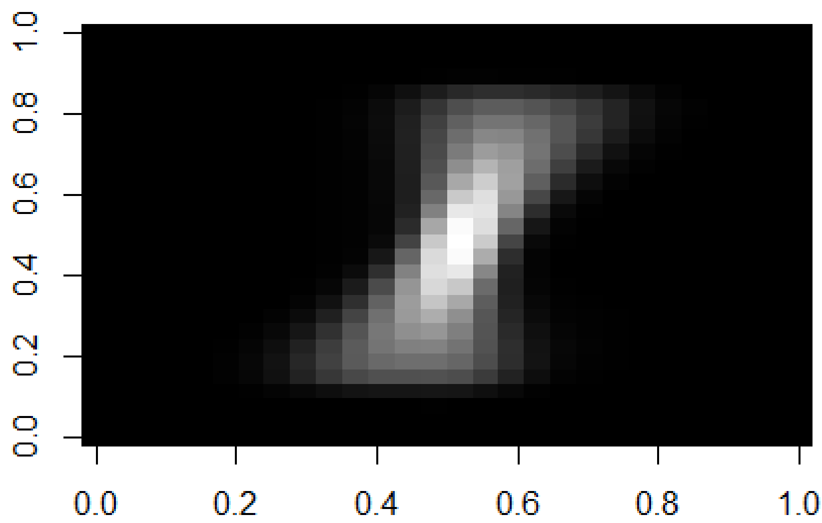
I have modified the testing function to capture the prediction and individual posteriors.

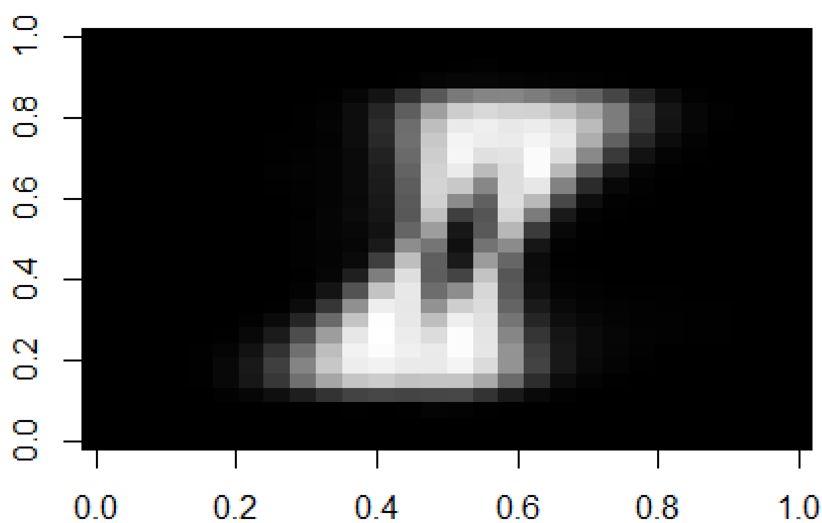
```
## Testing function

naive_bayes_testing <- function(test_set,model=model,type="prob"){

  #Testing

  predicted <- c(rep(0,nrow(test_set)))
  prob_matrix <- matrix(rep(0,nrow(test_set)*10), nrow = nrow(test_set), ncol
=10)
```

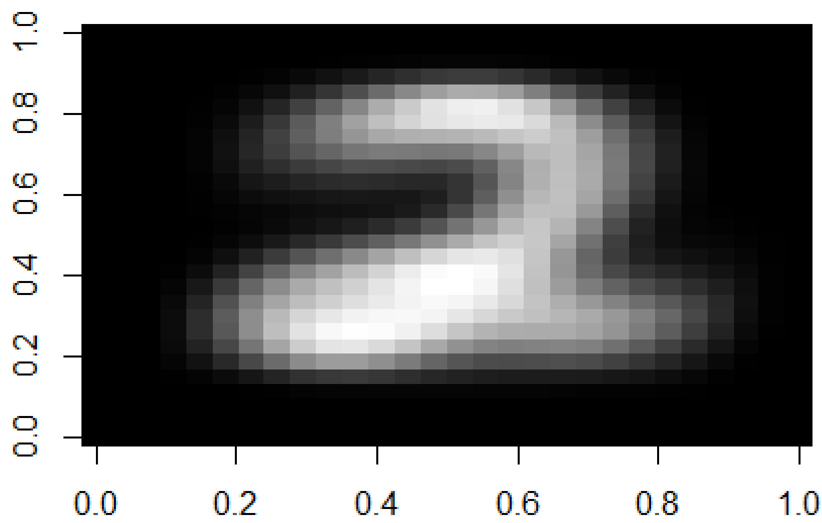
```r
  #Classifies all test cases one by one
  for (i in 1:nrow(test_set)) {

    #Test case
    test_vector <- test_set[i,2:785]

    #Calculates the posterior probabilities associated with all the digits
    #The classfication is the one with the highest posterior.
    #posteriors <- rep(0,10)
    for (j in 1:10) {

      #Corresponding mean and variance vector
      m <- model$means[j,]
      v <- model$variances[j,]

      likelihood <- 0
      for (k in 1:784) {
        if (v[[k]] > 0) {
          likelihood <- likelihood + log(dnorm(test_vector[[k]], m[[k]], sqrt
(v[[k]])))
        }
      }

      prob_matrix[i,j] <- likelihood + log(model$priors[j])


    }

    predicted[i] <- which.max(prob_matrix[i,])-1

  }

  if(type=="prob"){
    return(prob_matrix)
  }else if(type == "class"){
    return(predicted )
  }

}

model <- naive_bayes_training(training_set)
```

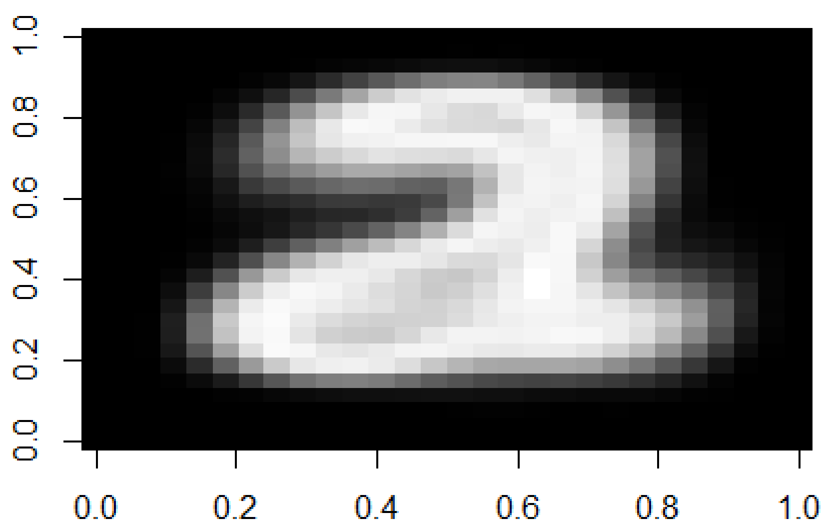

0.0
0.2
0.4
0.6
0.8
1.0

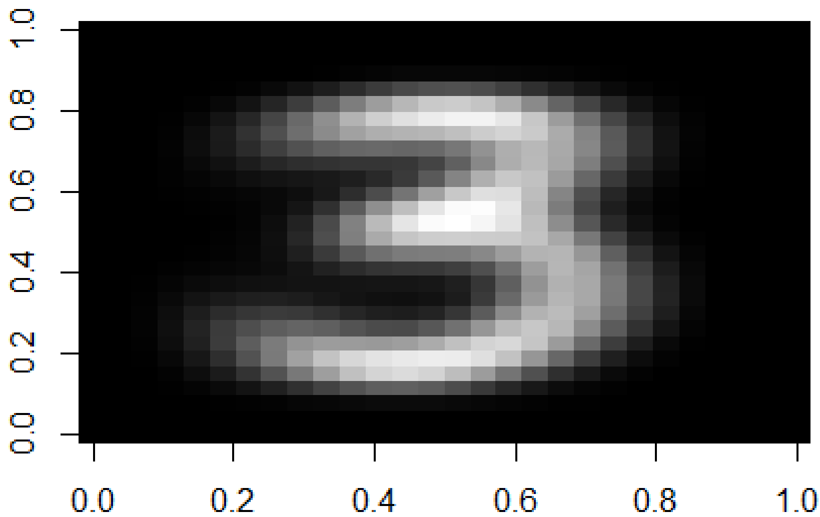

0.0
0.2
0.4
0.6
0.8
1.0

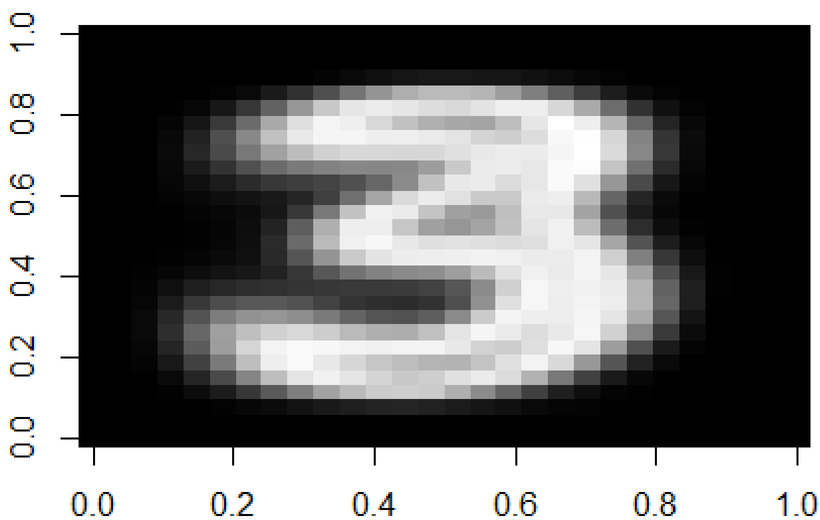

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

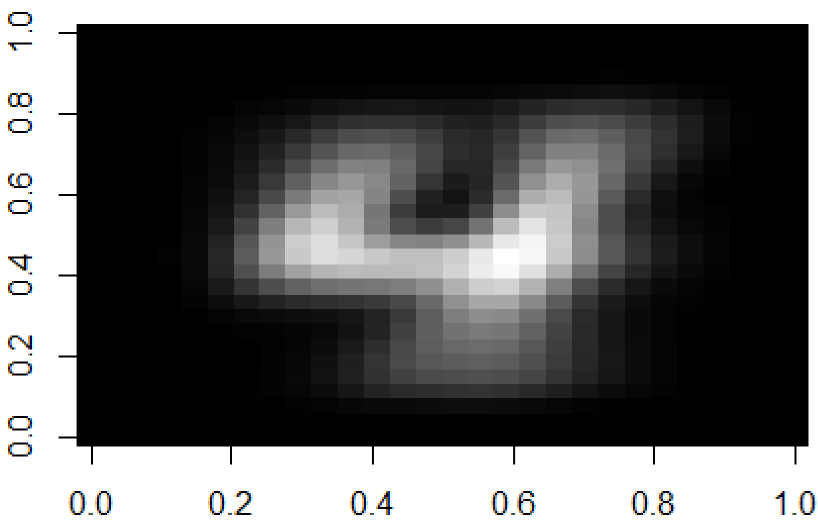

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

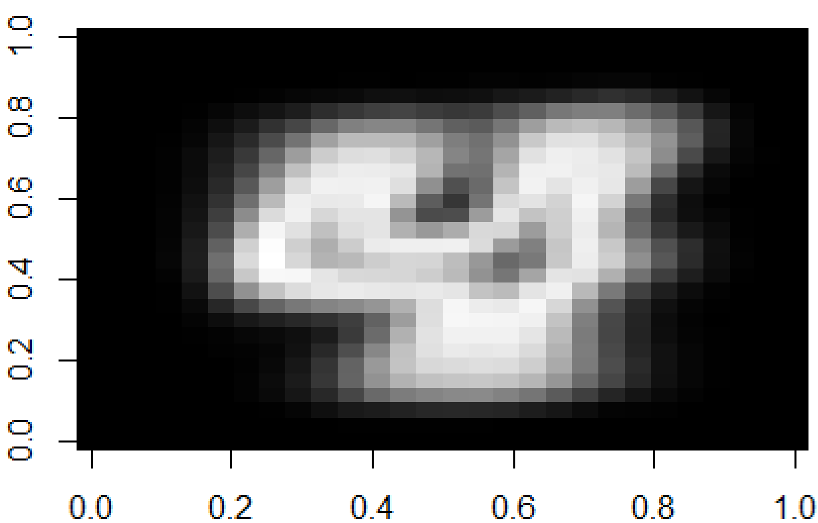

1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.2 0.4 0.6 0.8 1.0

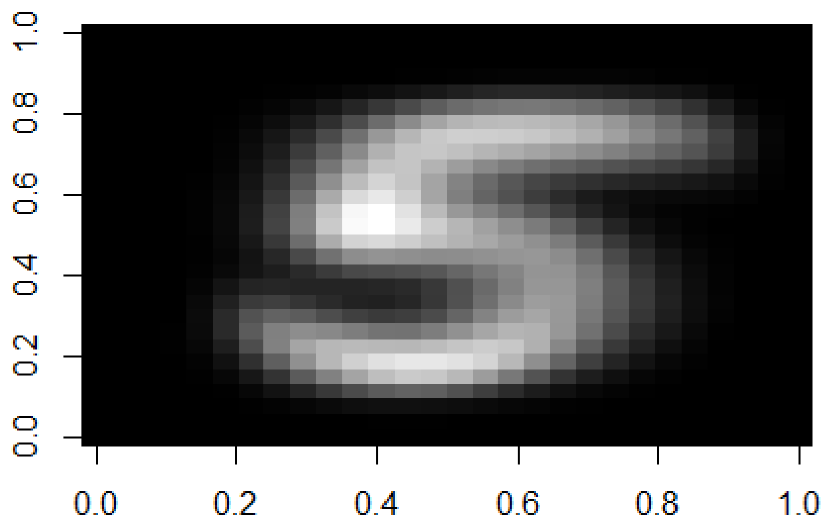

1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.2 0.4 0.6 0.8 1.0

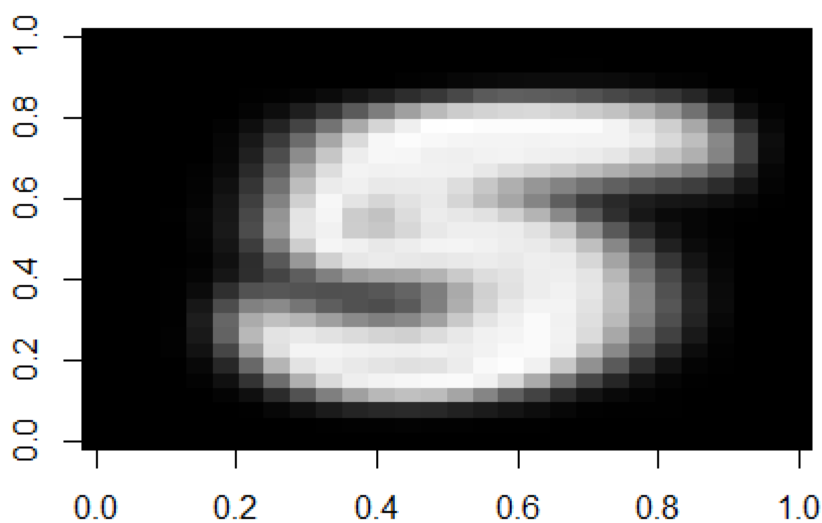

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

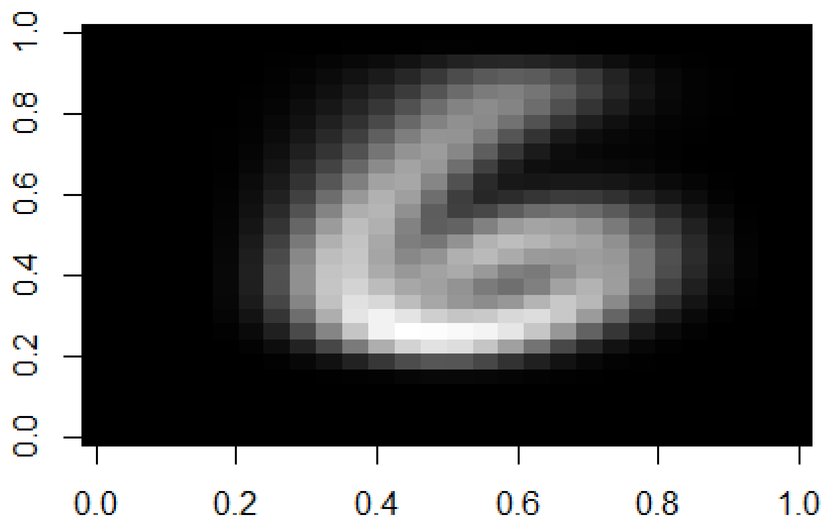

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

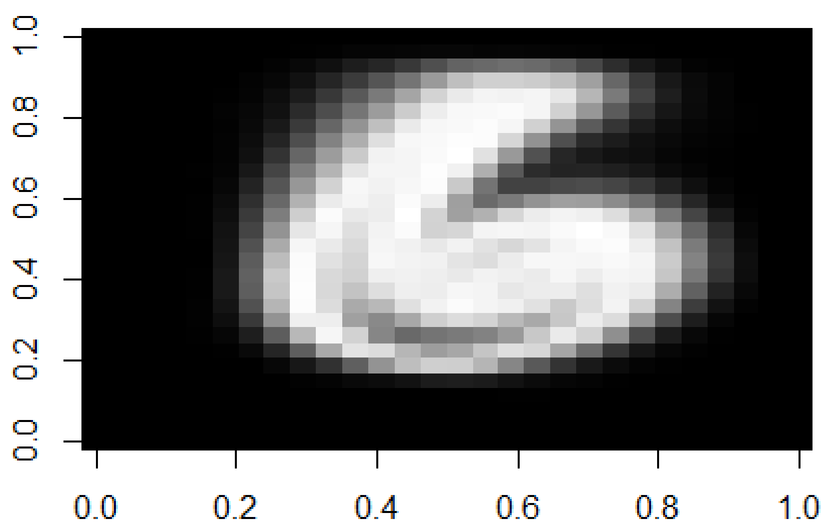

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

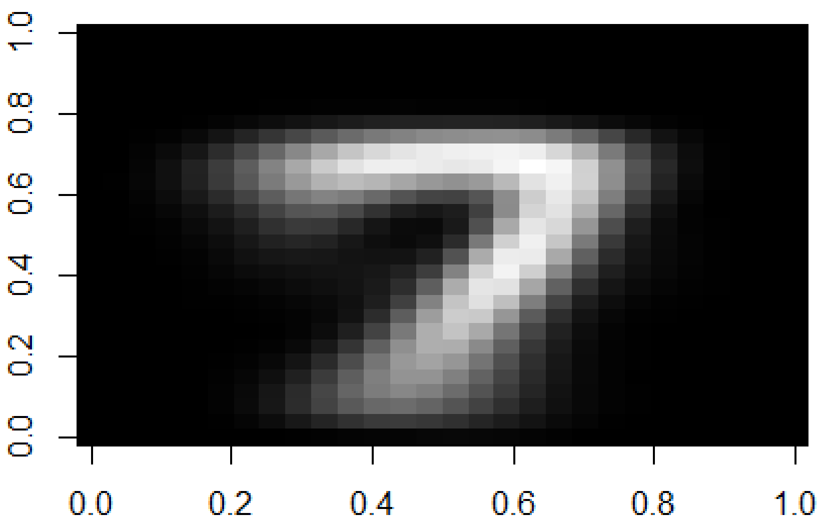

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

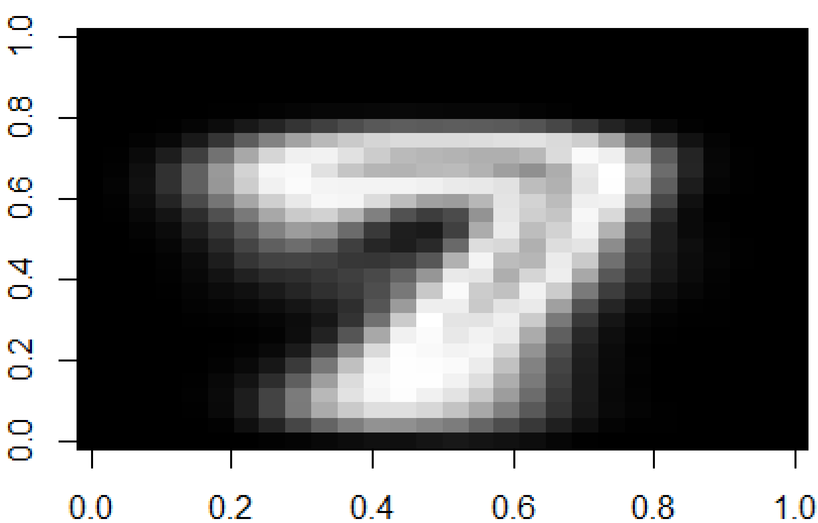

0.0 0.2 0.4 0.6 0.8 1.0
0.0 0.2 0.4 0.6 0.8 1.0

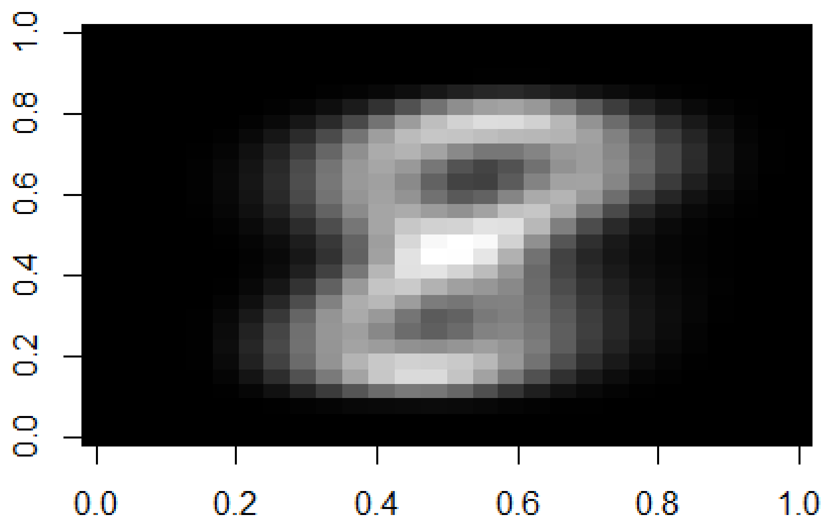

0.0 0.2 0.4 0.6 0.8 1.0
0.0 0.2 0.4 0.6 0.8 1.0

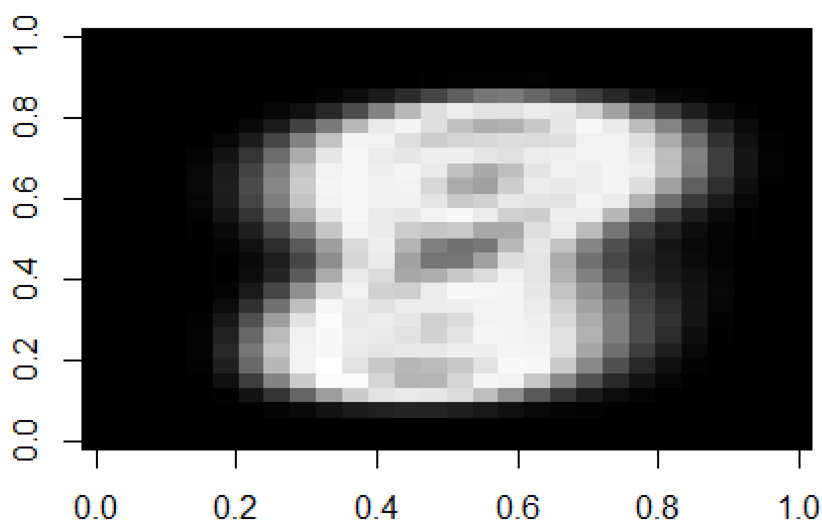

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

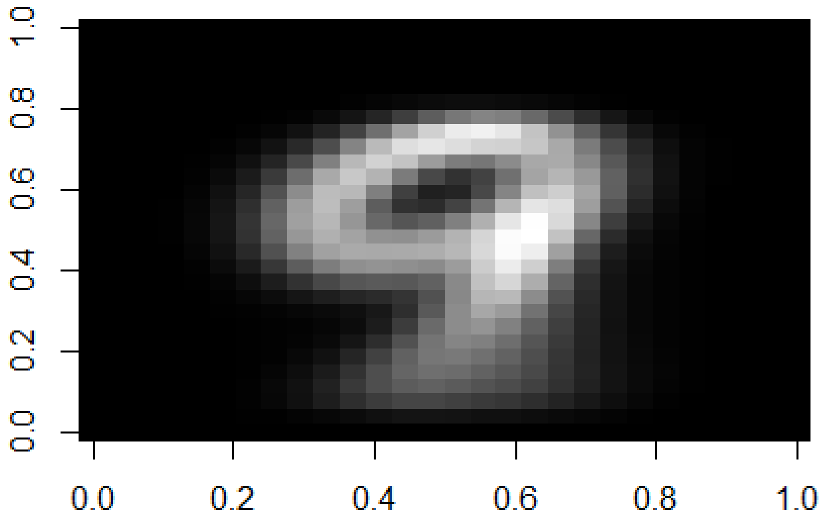

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

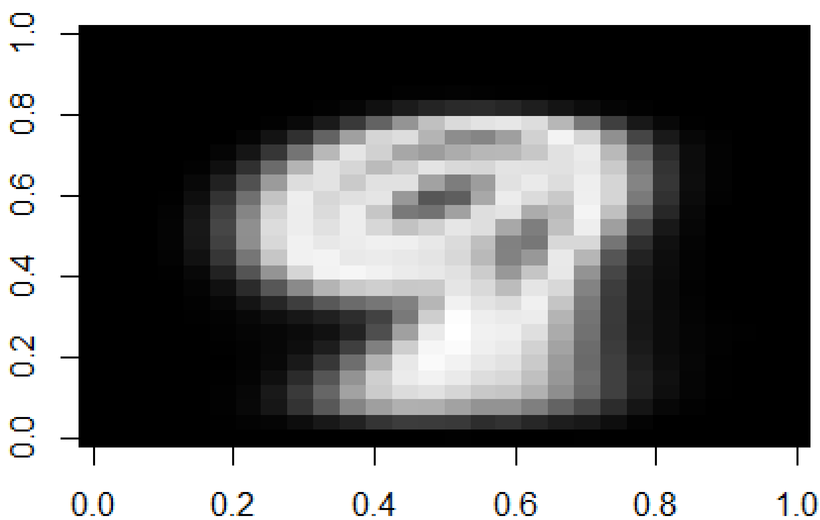

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0 0.2 0.4 0.6 0.8 1.0
0.0 0.2 0.4 0.6 0.8 1.0


0.0 0.2 0.4 0.6 0.8 1.0
0.0 0.2 0.4 0.6 0.8 1.0





```
summary(model)

##              Length Class      Mode
## priors        10    table      numeric
```

```
## means      785    data.frame list
## variances 785    data.frame list
```

```r
str(model$priors)
```

```
##  'table' num [1:10(1d)] 0.0987 0.1124 0.0993 0.1022 0.0974 ...
##  - attr(*, "dimnames")=List of 1
##   ..$ : chr [1:10] "0" "1" "2" "3" ...
```

```r
model$priors %>% dim()
```

```
## [1] 10
```

```r
model$means %>% dim()
```

```
## [1]  10 785
```

```r
model$variances %>% dim()
```

```
## [1]  10 785
```

```r
library(parallel)
#detectCores()
```

```r
predictLabel_model <- naive_bayes_testing(test_set,model,type="class")
```

```r
table(predictLabel_model, test_set$V1)
```

```
##
## predictLabel_model   0   1   2   3   4   5   6   7   8   9
##                  0 462   0  22  34   8  51  20  11  15   5
##                  1   0 636  15  21   0  16  11   8  34   6
##                  2  53   4 294   5   6   1  16   1   7   1
##                  3  10   6 113 393   7  63   0  15  30   7
##                  4  31   0   4   4 230  14   6   3   5  14
##                  5   1   0   0   6   4  90   8   2  11   0
##                  6  13   9  75  16  36  18 494   1   7   1
##                  7   2   0   7  27  18   7   7 361  12  25
##                  8  13  23  84  72  29 225  12  19 407  12
##                  9   3   3   5  28 251  50   1 196  56 534
```

```r
confusionMatrix(factor(predictLabel_model), factor(test_set$V1))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 462   0  22  34   8  51  20  11  15   5
##          1   0 636  15  21   0  16  11   8  34   6
##          2  53   4 294   5   6   1  16   1   7   1
##          3  10   6 113 393   7  63   0  15  30   7
##          4  31   0   4   4 230  14   6   3   5  14
##          5   1   0   0   6   4  90   8   2  11   0
```

```
##          6  13   9  75  16  36  18 494   1   7   1
##          7   2   0   7  27  18   7   7 361  12  25
##          8  13  23  84  72  29 225  12  19 407  12
##          9   3   3   5  28 251  50   1 196  56 534
##
## Overall Statistics
##
##                Accuracy : 0.6503
##                  95% CI : (0.6381, 0.6624)
##     No Information Rate : 0.1135
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.611
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.78571   0.9339  0.47496  0.64851  0.39049  0.16822
## Specificity           0.96932   0.9791  0.98253  0.95346  0.98503  0.99414
## Pos Pred Value        0.73567   0.8514  0.75773  0.61025  0.73955  0.73770
## Neg Pred Value        0.97654   0.9914  0.94208  0.96022  0.93688  0.92428
## Prevalence            0.09802   0.1135  0.10318  0.10102  0.09818  0.08918
## Detection Rate        0.07701   0.1060  0.04901  0.06551  0.03834  0.01500
## Detection Prevalence  0.10468   0.1245  0.06468  0.10735  0.05184  0.02034
## Balanced Accuracy     0.87752   0.9565  0.72874  0.80099  0.68776  0.58118
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.85913  0.58509  0.69692  0.88264
## Specificity           0.96755  0.98049  0.90970  0.89006
## Pos Pred Value        0.73731  0.77468  0.45424  0.47382
## Neg Pred Value        0.98480  0.95373  0.96531  0.98543
## Prevalence            0.09585  0.10285  0.09735  0.10085
## Detection Rate        0.08235  0.06018  0.06784  0.08901
## Detection Prevalence  0.11169  0.07768  0.14936  0.18786
## Balanced Accuracy     0.91334  0.78279  0.80331  0.88635
```

```
prob_Train_nb <- naive_bayes_testing(training_set,model, type="prob")
```

```
prob_Test_nb <- naive_bayes_testing(test_set,model,type="prob")
```

```
head(prob_Train_nb)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]     [,7]
## [1,]      -Inf      -Inf      -Inf -2901.226      -Inf      -Inf     -Inf
## [2,] -2966.781 -1999.752 -2941.981 -2769.921 -2968.601 -2792.121 -3365.28
## [3,]      -Inf      -Inf -3744.383 -3888.410      -Inf -3337.638     -Inf
## [4,] -3057.916      -Inf -2833.903 -2876.888      -Inf -2893.728     -Inf
## [5,] -3080.578 -2034.425 -2890.820 -2752.005 -2830.899 -2775.781 -2649.94
## [6,] -3202.360 -6179.648 -3102.941 -2701.192      -Inf -2947.348     -Inf
##          [,8]      [,9]     [,10]
## [1,]     -Inf      -Inf      -Inf
```

```
## [2,]      -Inf -2617.653 -2876.190
## [3,] -2742.04      -Inf -2479.999
## [4,]      -Inf -3035.145      -Inf
## [5,]      -Inf -2637.187 -3317.226
## [6,]      -Inf -2970.433      -Inf
```

```
prob_Test_nb%>% exp()%>% head(3)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    0    0    0    0    0    0    0    0     0
## [2,]    0    0    0    0    0    0    0    0    0     0
## [3,]    0    0    0    0    0    0    0    0    0     0
```

Due to higher negative values, exponential of the numbers are becoming zero

The Below function finds the relative probabilites of the posteriors and it is based on the idea that the sum of probabilities of the 10 labels for each record is 1.

```
prob <- function(matrix_){
  matrix_[matrix_== -Inf]<- -3.4e38
  matrix_<- 1/sweep(matrix_, 1, rowSums(matrix_), FUN = "/")
  return(sweep(matrix_,1,rowSums(matrix_), FUN = "/"))

}

prob_train1 <- prob(prob_Train_nb)
prob_test1 <- prob(prob_Test_nb)
prob_train1%>% head(2)
```

```
##              [,1]         [,2]         [,3]      [,4]         [,5]
## [1,] 8.533019e-36 8.533019e-36 8.533019e-36 1.0000000 8.533019e-36
## [2,] 1.033969e-01 1.533969e-01 1.042685e-01 0.1107454 1.033335e-01
##              [,6]         [,7]         [,8]         [,9]        [,10]
## [1,] 8.533019e-36 8.533019e-36 8.533019e-36 8.533019e-36 8.533019e-36
## [2,] 1.098648e-01 9.115314e-02 9.022232e-37 1.171874e-01 1.066535e-01
```

Find the Error input terms for gradient boosting model

```
error_1_train <- train_ideal_prob -prob_train1
error_1_test <-  test_ideal_prob - prob_test1

error_1_train %>% head(2)
```

```
##               [,1]          [,2]          [,3]       [,4]          [,5]
## [1,] -8.533019e-36 -8.533019e-36 -8.533019e-36 -1.0000000 -8.533019e-36
## [2,] -1.033969e-01  8.466031e-01 -1.042685e-01 -0.1107454 -1.033335e-01
##            [,6]          [,7]          [,8]          [,9]         [,10]
## [1,]  1.0000000 -8.533019e-36 -8.533019e-36 -8.533019e-36 -8.533019e-36
## [2,] -0.1098648 -9.115314e-02 -9.022232e-37 -1.171874e-01 -1.066535e-01
```

```
error_1_test %>% head(2)
```

```
##                [,1]          [,2]          [,3]          [,4]          [,5]
## [1,] -1.565112e-01 -1.893303e-36 -1.893303e-36 -1.893303e-36 -1.754969e-01
## [2,] -5.591733e-36 -5.591733e-36  3.595227e-01 -5.591733e-36 -5.591733e-36
##             [,6]          [,7]          [,8]          [,9]         [,10]
## [1,] -0.1810021 -1.893303e-36  7.439268e-01 -1.893303e-36 -2.309167e-01
## [2,] -0.3595227 -5.591733e-36 -5.591733e-36 -5.591733e-36 -5.591733e-36
```

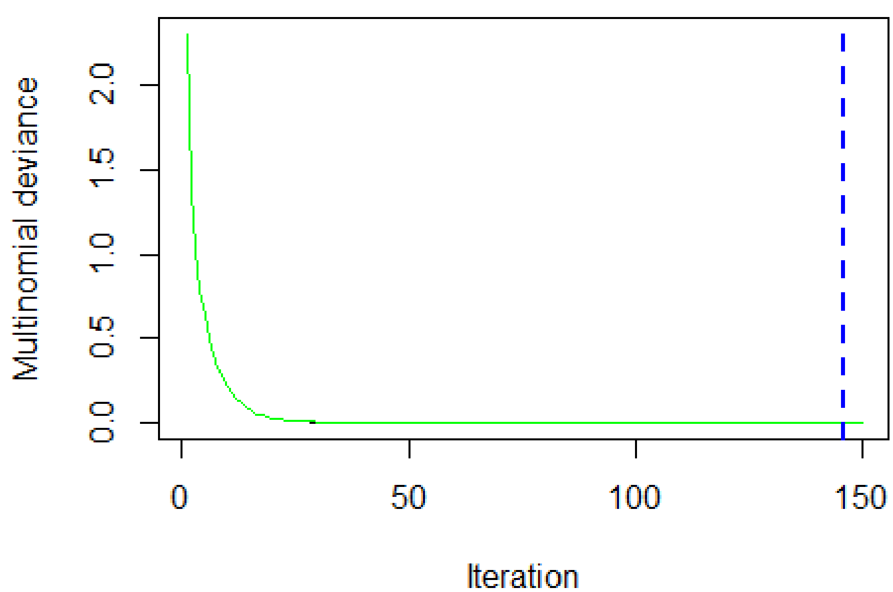
Combine the error terms with the training and test data set to create the inputs for the second model

```
column_Error_names <- paste0("E",0:9)
column_Error_names

##  [1] "E0" "E1" "E2" "E3" "E4" "E5" "E6" "E7" "E8" "E9"

colnames(error_1_train) <- column_Error_names
colnames(error_1_test) <- column_Error_names

error_1_train %>% head(2)

##                 E0            E1            E2         E3            E4
## [1,] -8.533019e-36 -8.533019e-36 -8.533019e-36 -1.0000000 -8.533019e-36
## [2,] -1.033969e-01  8.466031e-01 -1.042685e-01 -0.1107454 -1.033335e-01
##              E5            E6            E7            E8            E9
## [1,]  1.0000000 -8.533019e-36 -8.533019e-36 -8.533019e-36 -8.533019e-36
## [2,] -0.1098648 -9.115314e-02 -9.022232e-37 -1.171874e-01 -1.066535e-01

training_set<- training_set %>% cbind(error_1_train)
training_set[1,784:795]

##   V784 V785            E0            E1            E2 E3            E4 E5
## 1    0    0 -8.533019e-36 -8.533019e-36 -8.533019e-36 -1 -8.533019e-36  1
##              E6            E7            E8            E9
## 1 -8.533019e-36 -8.533019e-36 -8.533019e-36 -8.533019e-36

test_set<- test_set %>% cbind(error_1_test)
test_set[1,784:795]

##   V784 V785         E0            E1            E2            E3
## 1    0    0 -0.1565112 -1.893303e-36 -1.893303e-36 -1.893303e-36
##           E4         E5            E6        E7            E8         E9
## 1 -0.1754969 -0.1810021 -1.893303e-36 0.7439268 -1.893303e-36 -0.2309167

rm(error_1_test)
rm(error_1_train)
rm(i,j)
```

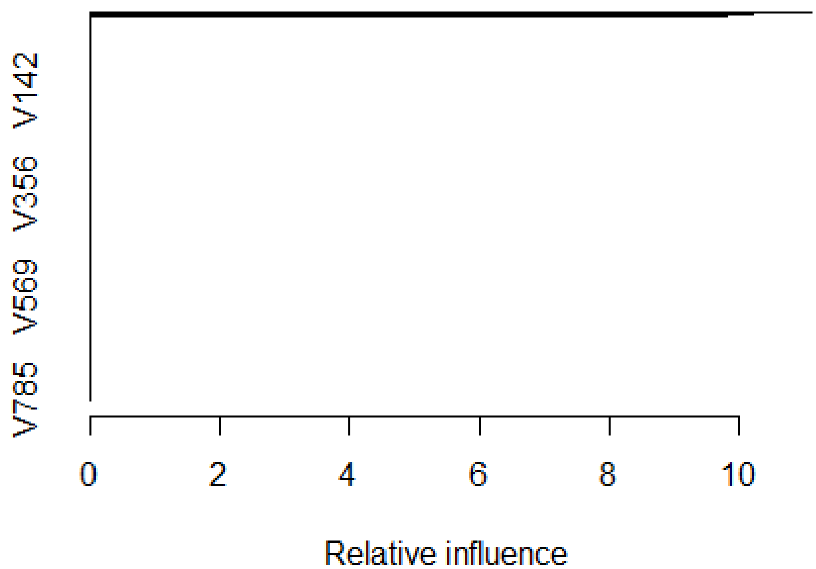
gbm

Made the labels as factor for the classification model

```r
training_set$V1 <-as.factor(training_set$V1)
test_set$V1 <-as.factor(test_set$V1)

#gbm1 <- gbm(V1~. , data=training_set,n.trees = 1000, cv.folds=3)
#ntree_opt_cv <- gbm.perf(gbm1, method = "cv")

gbm1 <- gbm(V1~. , data=training_set,n.trees = 150, cv.folds = 3)

## Distribution not specified, assuming multinomial ...

## Warning in gbm.fit(x = x, y = y, offset = offset, distribution =
## distribution, : variable 1: V2 has no variation.

## Warning in gbm.fit(x = x, y = y, offset = offset, distribution =
## distribution, : variable 2: V3 has no variation.

## Warning in gbm.fit(x = x, y = y, offset = offset, distribution =
## distribution, : variable 3: V4 has no variation.

## Warning in gbm.fit(x = x, y = y, offset = offset, distribution =
## distribution, : variable 4: V5 has no variation.

## Warning in gbm.fit(x = x, y = y, offset = offset, distribution =
## distribution, : variable 784: V785 has no variation.

ntree_opt_cv <- gbm.perf(gbm1, method = "cv")
```




```r
gbm1 %>% summary()
```




```
##        var      rel.inf
## E1      E1 1.111857e+01
## E7      E7 1.046770e+01
## E3      E3 1.019824e+01
## E6      E6 9.953675e+00
## E2      E2 9.943944e+00
## E9      E9 9.896836e+00
## E0      E0 9.804355e+00
## E4      E4 9.767943e+00
## E8      E8 9.750968e+00
## E5      E5 9.097533e+00
## V752 V752 7.954944e-05
## V724 V724 7.038047e-05
## V336 V336 4.107935e-05
## V697 V697 2.678317e-05
## V335 V335 8.786782e-06
## V725 V725 4.777435e-06
## V364 V364 2.216127e-06
## V753 V753 6.639918e-09
## V307 V307 9.166533e-14
## V363 V363 2.871935e-15
## V2      V2 0.000000e+00
## V3      V3 0.000000e+00
## V4      V4 0.000000e+00
## V5      V5 0.000000e+00
## V6      V6 0.000000e+00
## V7      V7 0.000000e+00
```

```
## V8      V8 0.000000e+00
## V9      V9 0.000000e+00
## V10   V10 0.000000e+00
## V11   V11 0.000000e+00
## V12   V12 0.000000e+00
## V13   V13 0.000000e+00
## V14   V14 0.000000e+00
## V15   V15 0.000000e+00
## V784 V784 0.000000e+00
## V785 V785 0.000000e+00
```

Observation - The GBM model is only using the 10 error terms and 10 other pixels to classify the labels.

```
final_predict<-predict(gbm1,test_set,type = "response", n.tree =ntree_opt_cv)

final_predict

## , , 146
##
##                    0            1            2            3            4
##    [1,] 1.891662e-11 2.021505e-11 1.889776e-11 1.922813e-11 1.884202e-11
##    [2,] 2.156387e-11 2.304402e-11 1.000000e+00 2.191898e-11 2.147884e-11
##    [3,] 2.015827e-11 1.000000e+00 2.013817e-11 2.049023e-11 2.007877e-11
##    [4,] 8.587716e-10 9.137254e-10 8.585725e-10 8.726087e-10 1.151412e-09
## [5998,] 2.030585e-09 2.167827e-11 2.016494e-11 5.179715e-11 2.345097e-09
## [5999,] 9.999997e-01 2.262058e-09 2.104146e-09 4.449491e-09 2.447033e-07

# reducing 1 from which max as label stsrts from 0 , but column no starts fro
m 1
pred<- final_predict %>% apply(1,which.max)-1

pred %>% head()

## [1] 7 2 1 9 5 9

confusionMatrix(data = factor(pred),reference = test_set$V1)

## Confusion Matrix and Statistics
##
##             Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 588   0   0   0   0   0   0   0   0   0
##          1   0 681   0   0   0   0   0   0   0   0
##          2   0   0 619   0   0   0   0   0   0   0
##          3   0   0   0 606   0   0   0   0   0   0
##          4   0   0   0   0 589   0   0   0   0   0
##          5   0   0   0   0   0 535   0   0   0   0
##          6   0   0   0   0   0   0 575   0   0   0
##          7   0   0   0   0   0   0   0 617   0   0
##          8   0   0   0   0   0   0   0   0 584   0
##          9   0   0   0   0   0   0   0   0   0 605
```

```
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9994, 1)
##     No Information Rate : 0.1135
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           1.00000   1.0000   1.0000    1.000  1.00000  1.00000
## Specificity           1.00000   1.0000   1.0000    1.000  1.00000  1.00000
## Pos Pred Value        1.00000   1.0000   1.0000    1.000  1.00000  1.00000
## Neg Pred Value        1.00000   1.0000   1.0000    1.000  1.00000  1.00000
## Prevalence            0.09802   0.1135   0.1032    0.101  0.09818  0.08918
## Detection Rate        0.09802   0.1135   0.1032    0.101  0.09818  0.08918
## Detection Prevalence  0.09802   0.1135   0.1032    0.101  0.09818  0.08918
## Balanced Accuracy     1.00000   1.0000   1.0000    1.000  1.00000  1.00000
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           1.00000   1.0000  1.00000   1.0000
## Specificity           1.00000   1.0000  1.00000   1.0000
## Pos Pred Value        1.00000   1.0000  1.00000   1.0000
## Neg Pred Value        1.00000   1.0000  1.00000   1.0000
## Prevalence            0.09585   0.1029  0.09735   0.1009
## Detection Rate        0.09585   0.1029  0.09735   0.1009
## Detection Prevalence  0.09585   0.1029  0.09735   0.1009
## Balanced Accuracy     1.00000   1.0000  1.00000   1.0000
```

By using the errors form the naive bayes model as input for gradiantboosting model , the testset prediction has improved to 100% from 65.03% accuracy of naive bayes algorithm. The gbm model may be over fitting in this case or the way I has penalised the errors from the naive bayes model is helping the second models to predict with much higher accuracy. Earlier I had used randomforest, which gave an accurarcy of 97.XX%.