Marley Akonnnor HW 8

Marley Akonnor

12/3/2021

```r
#install.packages("readr")
#install.packages("randomForest")
#install.packages("nnet")
library(readr)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```r
library(nnet)
```

```r
setwd("/Users/m/Documents/M.S Syracuse Data Science/Courses/IST 707 - Data Mining - Machine Learning/Ho
train_orig <- read_csv("digit_train.csv")
```

```
## Rows: 42000 Columns: 785
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (785): label, pixel0, pixel1, pixel2, pixel3, pixel4, pixel5, pixel6, pi...
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
test_orig <- read_csv("digit_test.csv")
```

```
## Rows: 28000 Columns: 784
```

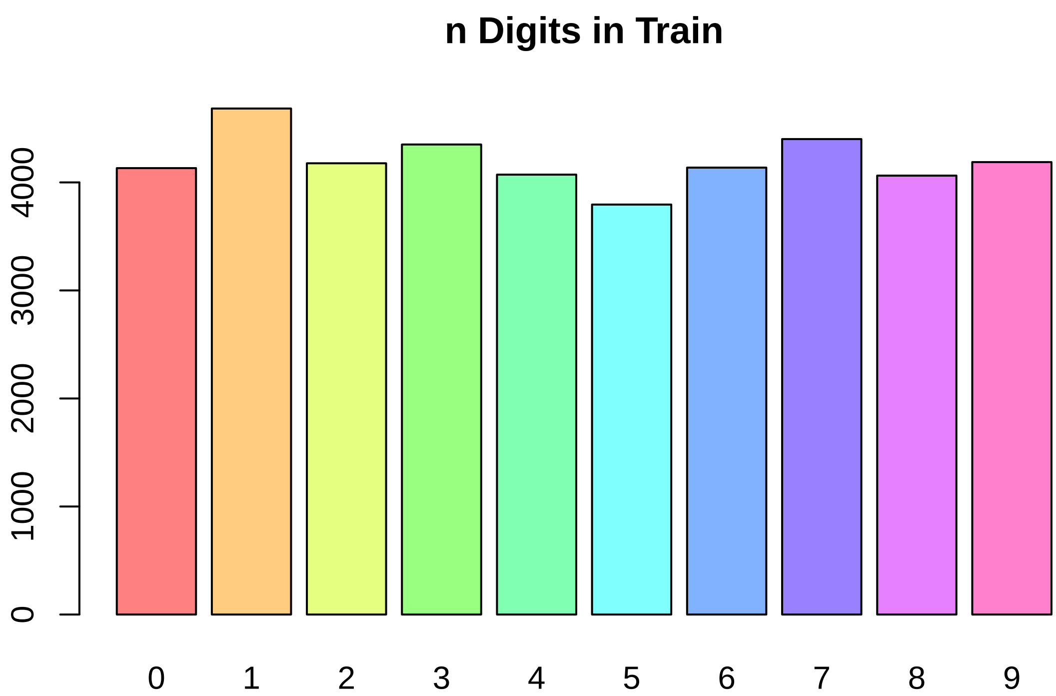
```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (784): pixel0, pixel1, pixel2, pixel3, pixel4, pixel5, pixel6, pixel7, p...
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# save the training labels
train_orig_labels <- train_orig[, 1]
train_orig_labels <- as.factor(train_orig_labels$label)
summary(train_orig_labels)
```

```
##    0    1    2    3    4    5    6    7    8    9
## 4132 4684 4177 4351 4072 3795 4137 4401 4063 4188
```

```r
barplot(table(train_orig[,1]), col=rainbow(10, 0.5), main="n Digits in Train")
```



There is around 4000 observations for each digit. Each row has 784 columns

(pixels) which form a 28x28 image. Let's see what the handwritten digits look

like by plotting them. Here is a function to plot a selection of digits from

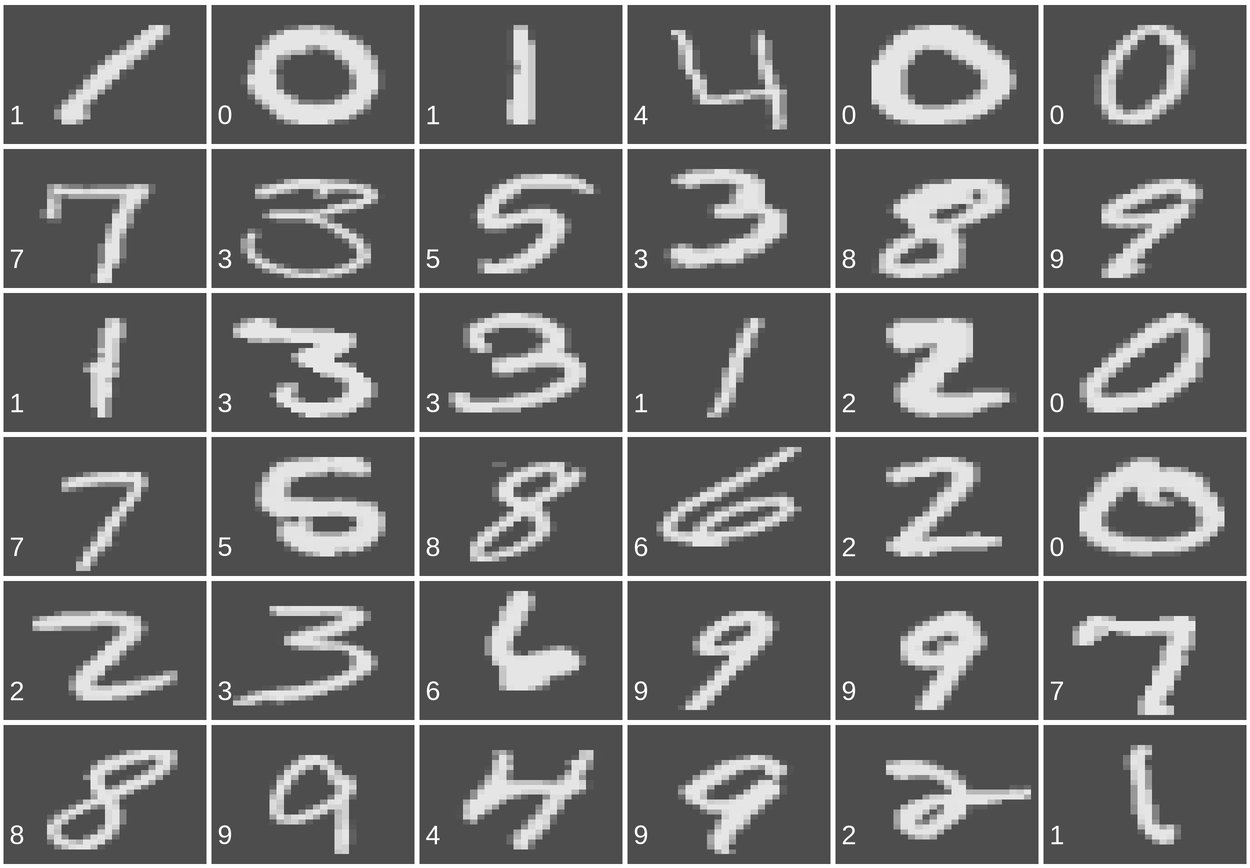
the train dataset.

```
plotTrain <- function(images, ds, labels){
  op <- par(no.readonly=TRUE)
  x <- ceiling(sqrt(length(images)))
  par(mfrow=c(x, x), mar=c(.1, .1, .1, .1))

  for (i in images){ #reverse and transpose each matrix to rotate images
    m <- matrix(data.matrix(ds[i,-1]), nrow=28, byrow=TRUE)
    m <- apply(m, 2, rev)
    image(t(m), col=grey.colors(255), axes=FALSE)
    text(0.05, 0.2, col="white", cex=1.2, labels[i])
  }
  par(op) #reset the original graphics parameters
}

test_orig_36 <- test_orig[1:36,]
test_orig_36$label <- c(2,0,9,0,3,7,0,3,0,3,5,7,4,0,4,3,3,1,9,0,9,1,1,5,7,4,2,7,4,7,7,5,4,2,6,2)
```

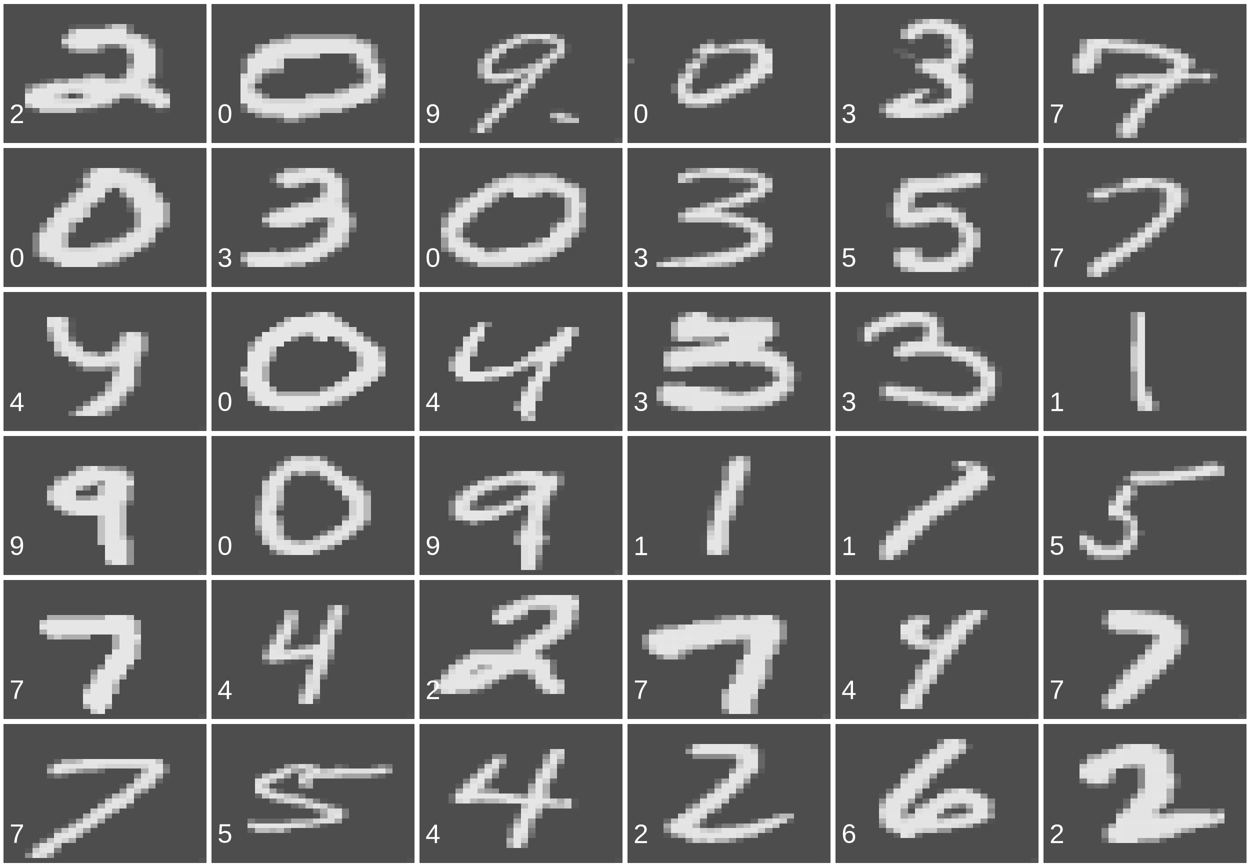
Now let's use this function to look at the first 36 images. You can look at

many images if you wanted too, e.g., plotTrain(1001:1100)

```
plotTrain(1:36, train_orig, train_orig$label)
```




```
plotTrain(1:36, test_orig_36, test_orig_36$label)
```



first we are going to try a random forest

```
numTrees <- 25
```

Train on entire training dataset and predict on the test

```
startTime <- proc.time()
rf <- randomForest(train_orig[-1], train_orig_labels, xtest=test_orig, ntree=numTrees)
proc.time() - startTime
```

```
##    user  system elapsed
## 142.996   1.097 144.275
```

user system elapsed

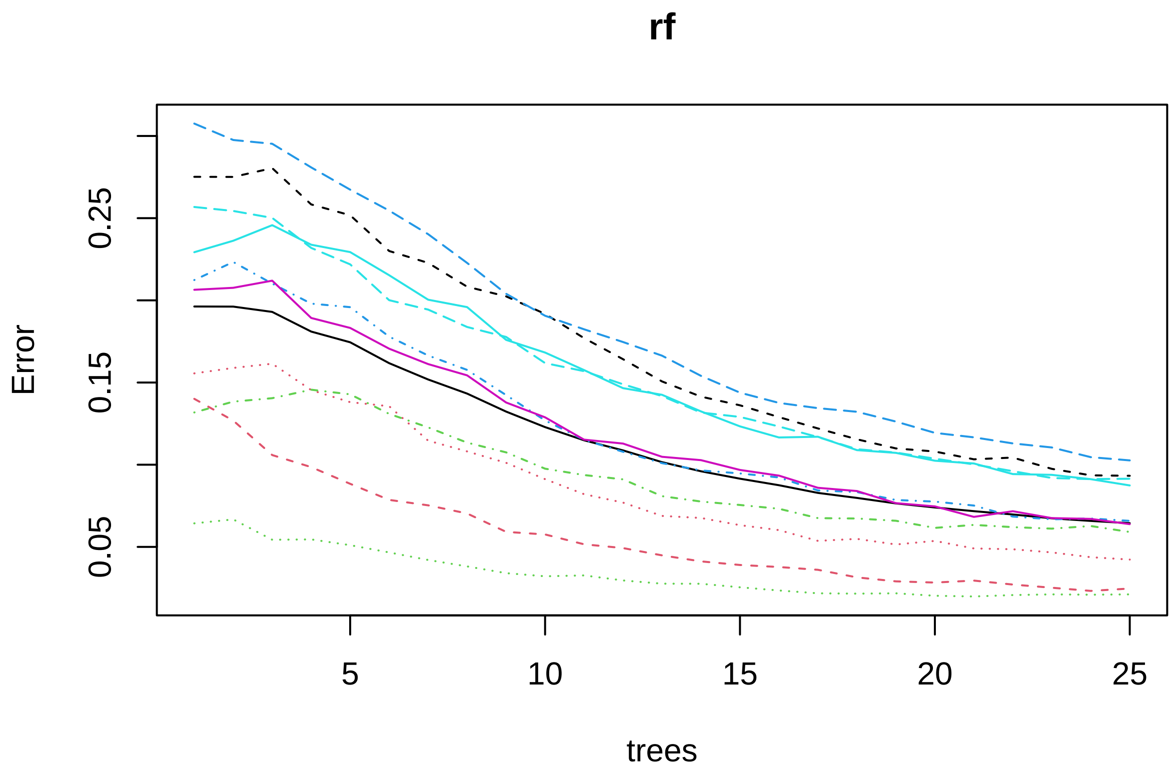
165.52 6.52 207.74

```
print(rf)
```

```
##
## Call:
##  randomForest(x = train_orig[-1], y = train_orig_labels, xtest = test_orig,      ntree = numTrees)
##                Type of random forest: classification
##                      Number of trees: 25
```

```
## No. of variables tried at each split: 28
##
##         OOB estimate of  error rate: 6.44%
## Confusion matrix:
##      0    1    2    3    4    5    6    7    8    9 class.error
## 0 4030    0    8    7    7    9   31    2   29    9  0.02468538
## 1    0 4585   29   15    9    9    8   10   16    3  0.02113578
## 2   33   19 3902   41   34   20   22   55   34   17  0.06583672
## 3   12   10  101 3953   10  111   14   38   70   32  0.09147322
## 4    5   11   20    6 3812   14   27   16   21  140  0.06385069
## 5   23    9   15  146   22 3441   42    8   55   34  0.09328063
## 6   41    7   18    8   19   53 3962    1   24    4  0.04230118
## 7    3   19   67   18   32    4    3 4141   15   99  0.05907748
## 8   22   33   57   95   36   62   29   13 3646   70  0.10263352
## 9   20   10   23   58   96   43    7   62   47 3822  0.08739255
```

```
plot(rf,type="l")
```



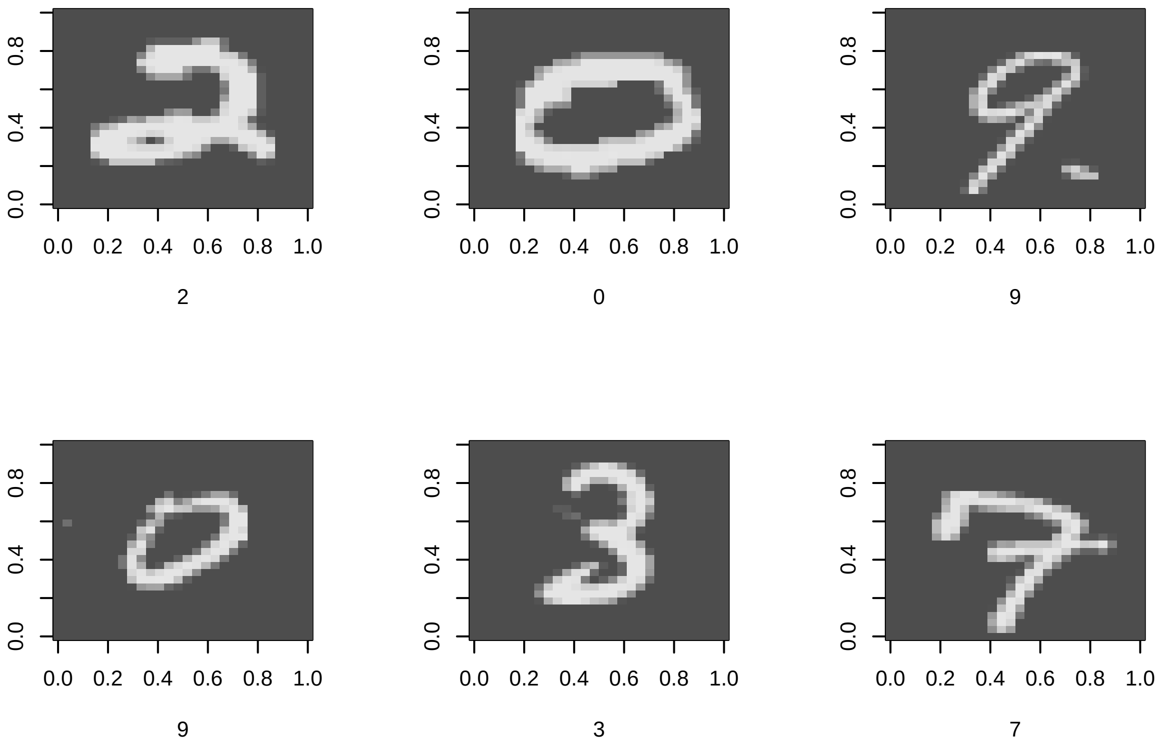
output predictions for submission

```
predictions <- data.frame(ImageId=1:nrow(test_orig),
  Label=levels(train_orig_labels)[rf$test$predicted])
head(predictions)
```

```
##   ImageId Label
## 1       1     2
## 2       2     0
## 3       3     9
## 4       4     9
```

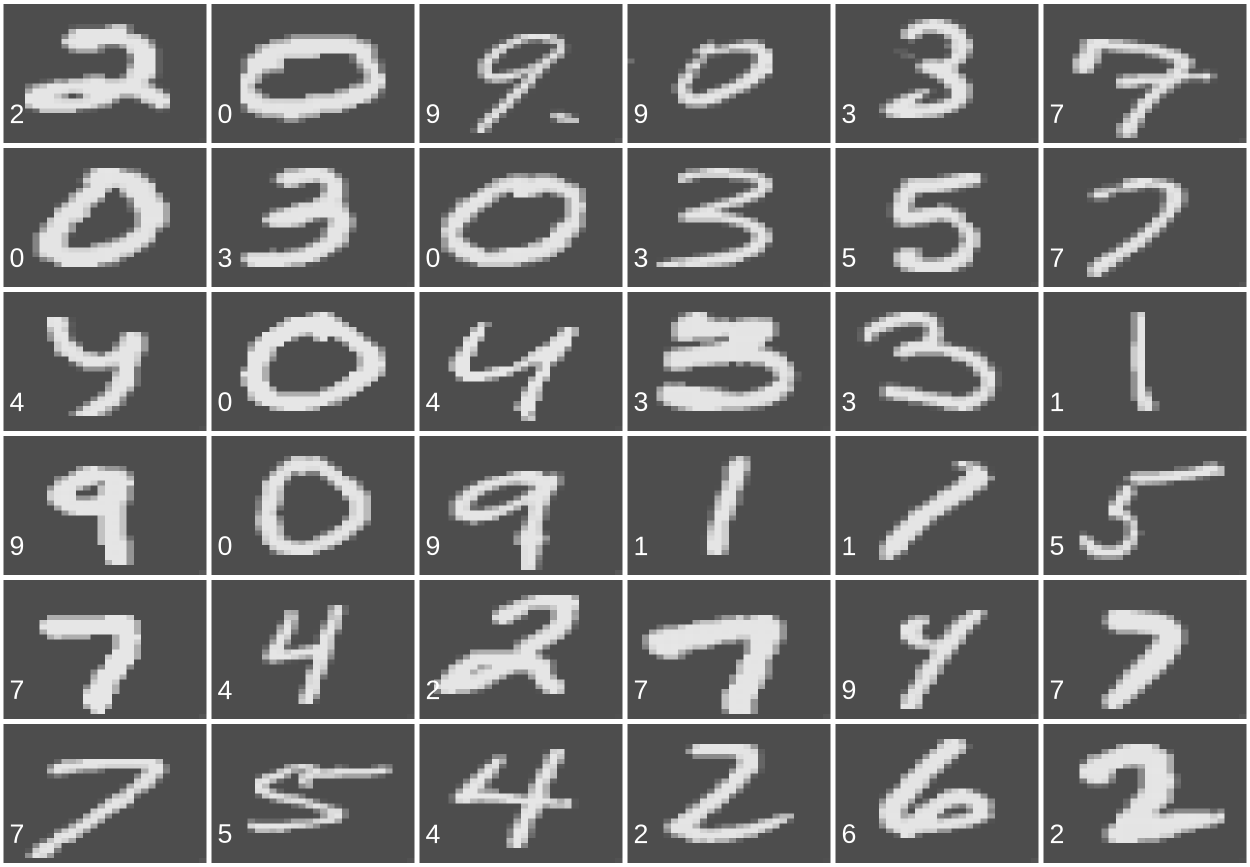
```
## 5        5      3
## 6        6      7
```

```
rotate <- function(x) t(apply(x, 2, rev)) # reverses (rotates the matrix)

par(mfrow=c(2,3)) # Plotting in 2*3 format (random forest)
lapply(1:6,
       function(x) image( #norow = 28 because this is 28 pixel image
         rotate(matrix(unlist(test_orig[x,]),nrow = 28,byrow = T)),
         col=grey.colors(255),
         xlab=predictions[x,2]
       )
)
```



```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
```

```
plotTrain(1:36, test_orig_36, predictions$Label)
```




```
predictions_36 <- predictions[1:36,2]
accuracy <- mean(predictions_36 == test_orig_36$label)
print(paste('Accuracy:', accuracy))
```

```
## [1] "Accuracy: 0.944444444444444"
```

split the training data into train and test to do local evaluation

```
set.seed(123)
rows <- sample(1:nrow(train_orig), as.integer(0.7*nrow(train_orig)))
```

Get train and test labels

```
train_labels <- train_orig[rows, 1]
test_labels <- train_orig[-rows, 1]
```

convert the labels to factors

```
train_labels <- as.factor(train_labels$label)
```

custom normalization function

```
normalize <- function(x) {
  return(x / 255)
}
```

create the train and test datasets and apply normalization

```
train_norm <- as.data.frame(lapply(train_orig[rows, -1], normalize))
test_norm <- as.data.frame(lapply(train_orig[-rows,-1], normalize))
```

check a random pixel to see if the normalization worked

```
summary(train_orig$pixel350)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    0.00    0.00   89.51  228.00  255.00
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    0.00    0.00   89.51  228.00  255.00

summary(train_norm$pixel350)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.3487  0.8902  1.0000
```

```
##     Min.   1st Qu.   Median     Mean   3rd Qu.     Max.
## 0.000000 0.000000 0.003922 0.350500 0.890200 1.000000

summary(test_norm$pixel350)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.01176 0.35629 0.90196 1.00000
```

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.0000 0.0000 0.0000 0.3521 0.9059 1.0000

create the class indicator matrix

```
train_labels_matrix = class.ind(train_labels)
head(train_labels)
```

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

[1] 2 2 9 2 8 7

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

```
head(train_labels_matrix)
```

```
##      0 1 2 3 4 5 6 7 8 9
```

```
## [1,] 0 0 0 0 0 0 0 1 0 0
## [2,] 0 0 0 0 0 0 1 0 0 0
## [3,] 0 0 0 1 0 0 0 0 0 0
## [4,] 0 0 0 1 0 0 0 0 0 0
## [5,] 0 1 0 0 0 0 0 0 0 0
## [6,] 0 0 0 0 1 0 0 0 0 0
```

0 1 2 3 4 5 6 7 8 9

[1,] 0 0 1 0 0 0 0 0 0 0

[2,] 0 0 1 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 0 0 0 0 0 0 0 1 0

[6,] 0 0 0 0 0 0 0 1 0 0

train model

```
set.seed(123)
startTime <- proc.time()
nn = nnet(train_norm, train_labels_matrix, size = 1, softmax = TRUE)
```

```
## # weights:  805
## initial  value 71562.488017
## iter  10 value 67404.101410
## iter  20 value 64669.742937
## iter  30 value 59259.471988
## iter  40 value 54208.857464
## iter  50 value 52919.684509
## iter  60 value 52486.942418
## iter  70 value 52188.376159
## iter  80 value 52043.260352
## iter  90 value 51942.104544
## iter 100 value 51873.801939
## final  value 51873.801939
## stopped after 100 iterations
```

weights: 805

initial value 71631.780715

iter 10 value 64946.553191

iter 20 value 57294.824640

iter 30 value 55912.804141

iter 40 value 54648.757612

iter 50 value 53950.781576

iter 60 value 52927.199756

iter 70 value 52291.634751

iter 80 value 51967.602466

iter 90 value 51774.654787

iter 100 value 51643.951402

final value 51643.951402

stopped after 100 iterations

```
proc.time() - startTime
```

```
##    user  system elapsed
##  38.722   0.352  39.115
```

user system elapsed

46.97 0.13 47.54

nn

a 784-1-10 network with 805 weights

options were - softmax modelling

This is just to try out the nnet function. One hidden node is mostly likely

not enough for the model. "softmax" should be set to TRUE when performing

classification. The default maximum number of iterations is 100. The algorithm

did not converge before reaching the maximum. It ran for 46 seconds.

get predictions

```
pred = predict(nn, test_norm, type="class")
cbind(head(pred), head(test_labels))
```

```
##   head(pred) label
## 1          7     9
## 2          1     2
## 3          1     0
## 4          1     6
## 5          7     9
## 6          1     1
```

head(pred) label

1 1 8

2 1 3

3 1 8

4 1 0

5 1 3

6 7 4

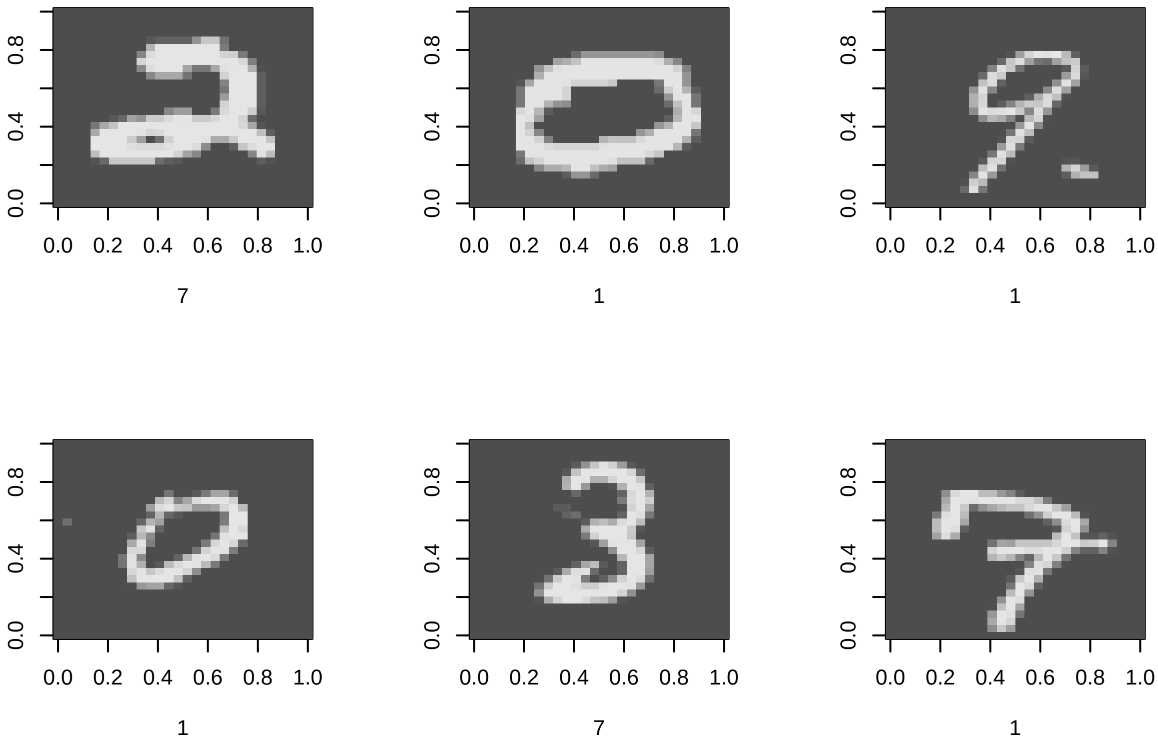
The first six predictions do not look very good.

evaluate the model

```
accuracy <- mean(pred == test_labels)
print(paste('Accuracy:', accuracy))
```

```
## [1] "Accuracy: 0.207681930005555"
```

```
## [1] "Accuracy: 0.206174113165622"

par(mfrow=c(2,3)) # Plotting in 2*3 format (neural net)
lapply(1:6,
       function(x) image( #norow = 28 because this is 28 pixel image
         rotate(matrix(unlist(test_orig[x,]),nrow = 28,byrow = T)),
         col=grey.colors(255),
         xlab=pred[x]
       )
)
```




```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
```