Handwritten Digit Classifier

Menna Mohie El-deen Mahmoud.

April 2018.

Introduction

This project aims to apply k-NN classification on the MNIST database and to calculate the accuracy of the classification.

The MNIST database consists of 60,000 handwritten digits (train sets) from '0' to '9', which we are going to use to classify the other 10,000 handwritten digits (test sets).
It can be found here: https://cs.nyu.edu/~roweis/data.html

We are going to use the train sets to find a single feature (image) for each digit, then we are going to compare each test with these features to find the nearest feature to each test, and so we will use that feature to classify the test.

I – Loading and tidying the data.

The data is available in many formats, we chose the *.mat format. In order to load a *.mat file into R, we need to install the "R.matlab" package.

```
install.packages("R.matlab");
data <- R.matlab::readMat("mnist_all.mat");
```

After loading the data, this is how it looks like:

Name	Type	Value
data	list [20]	List of length 20
train0	integer [5923 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
test0	integer [980 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
train1	integer [6742 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
test1	integer [1135 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
train2	integer [5958 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
test2	integer [1032 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
train3	integer [6131 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
test3	integer [1010 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
train4	integer [5842 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
test4	integer [982 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
train5	integer [5421 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
test5	integer [892 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
train6	integer [5918 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
test6	integer [958 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
train7	integer [6265 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
test7	integer [1028 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
train8	integer [5851 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
test8	integer [974 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
train9	integer [5949 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
test9	integer [1009 x 784]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...

The list consists of 20 nx784 matrix, n represents the number of rows/images, 784 represents the number of pixels. Each image is 28x28, hence 784 pixels.

train0/test0 are matrices of handwritten zeroes, train1/test1 are matrices of handwritten ones, and so on.

Now we need to separate the train sets from the test sets.

```
trains <- list(data[[1]], data[[3]], data[[5]], data[[7]], data[[9]], data[[11]], data[[13]],
               data[[15]], data[[17]], data[[19]]);

tests <- list(data[[2]], data[[4]], data[[6]], data[[8]], data[[10]], data[[12]], data[[14]],
              data[[16]], data[[18]], data[[20]]);
```

data[[1]] is the first element in the data list, which is train0. data[[2]] is the second element, which is test0, and so on.

II – Finding features.

To find the feature of zeroes, we are going to calculate the column means of train0, hence we'll have a row or a vector where the first cell is the mean of all the first pixels of the handwritten zeroes, the second cell is the mean of all the second pixels of the handwritten zeroes, and so on. 784 cells.

We do the same thing with train1, train2, train3… train9.

Instead of using a for-loop, we can use the "sapply()" function.

The sapply(x, FUN) takes x, which is a list, applies the function FUN on each element of x, and returns a matrix.

```
patterns <- sapply(trains, function(x) colMeans(x));
```

Now, patterns is a 784x10 matrix.

```
patterns        num [1:784, 1:10] 0 0 0 0 0 0 0 0 0 …
```

For simplicity, we will transform it into a 10x784 matrix, each row representing the feature of a specific digit from '0' to '9'.

```
patterns <- matrix(c(patterns), nrow=10, ncol = 784, byrow=TRUE);
```

This is the result:

```
patterns        num [1:10, 1:784] 0 0 0 0 0 0 0 0 0 …
```

1	0	0	0	0	0	0	0	0	0	0	0	0	0.000000000	0.00000000	0.00000000	0.000000000	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0.000000000	0.00000000	0.00000000	0.000000000	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0.001678416	0.03625378	0.03625378	0.001510574	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0.000000000	0.00000000	0.00000000	0.000000000	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0.000000000	0.00000000	0.00000000	0.000000000	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0.000000000	0.00000000	0.00000000	0.000000000	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0.019601217	0.04291991	0.00000000	0.000000000	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0.000000000	0.00000000	0.00000000	0.000000000	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0.000000000	0.00000000	0.00000000	0.000000000	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0.000000000	0.00000000	0.00000000	0.000000000	0	0	0	0	0	0	0

III – Calculating the correlations.

We need a list of 10 matrices, each matrix is nx10, where n represents the number of rows in the corresponding test matrix, and 10 represents the correlations with the 10 features.

```
correlations = lapply(1:10, function(x) matrix(nrow=nrow(tests[[x]]),ncol=10));
```

The lapply() function works from 1 to 10, each time creating a matrix with the above description, and returns a list containing the 10 matrices.

Name	Type	Value
correlations	list [10]	List of length 10
[[1]]	logical [980 x 10]	NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA...
[[2]]	logical [1135 x 10]	NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA...
[[3]]	logical [1032 x 10]	NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA...
[[4]]	logical [1010 x 10]	NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA...
[[5]]	logical [982 x 10]	NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA...
[[6]]	logical [892 x 10]	NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA...
[[7]]	logical [958 x 10]	NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA...
[[8]]	logical [1028 x 10]	NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA...
[[9]]	logical [974 x 10]	NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA...
[[10]]	logical [1009 x 10]	NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA...

Now we can calculate the correlations.

```
for(i in 1:10)
{
    for(j in 1:10)
    {
        correlations[[i]][,j] = apply(tests[[i]], 1, function(x) cor(x,patterns[j,]));
    }
}
```

The correlations list needs two indices, i, specifying the index of the matrix corresponding to the index of the test-set, if we are calculating the correlations of tests[[1]], we are going to put them in correlations[[1]]. The second index, j, specifying the index of the column corresponding to the index of the pattern that is used in the correlation. We have 10 matrices in correlations (i in 1:10), and 10 rows in patterns (j in 1:10).

The apply(y, margin, FUN), takes a margin of 1 to indicate rows, or 2 to indicate columns. So, here, apply() is going to apply function(x) on each row of tests[[i]].

The result:

Name	Type	Value
correlations	list [10]	List of length 10
[[1]]	double [980 x 10]	0.810 0.702 0.723 0.855 0.763 0.612 0.288 0.103 0.113 0.249 0.252 0.206 0.545 0. ...
[[2]]	double [1135 x 10]	0.170 0.193 0.246 0.192 0.174 0.192 0.780 0.851 0.798 0.586 0.516 0.700 0.411 0. ...
[[3]]	double [1032 x 10]	0.339 0.387 0.427 0.213 0.202 0.468 0.358 0.196 0.448 0.428 0.306 0.178 0.476 0. ...
[[4]]	double [1010 x 10]	0.248 0.376 0.279 0.285 0.449 0.350 0.308 0.470 0.315 0.460 0.377 0.467 0.330 0. ...
[[5]]	double [982 x 10]	0.3387 0.2268 0.2838 0.2312 0.3930 0.4445 0.0426 0.2154 0.1786 0.1550 0.1295 0.0 ...
[[6]]	double [892 x 10]	0.451 0.330 0.429 0.458 0.499 0.318 0.367 0.148 0.315 0.271 0.280 0.124 0.507 0. ...
[[7]]	double [958 x 10]	0.472 0.368 0.325 0.420 0.285 0.436 0.318 0.232 0.377 0.313 0.239 0.342 0.448 0. ...
[[8]]	double [1028 x 10]	0.3097 0.3716 0.2440 0.3538 0.3130 0.3185 0.2111 0.1756 0.0774 0.4472 0.1928 0.3 ...
[[9]]	double [974 x 10]	0.508 0.359 0.456 0.422 0.404 0.472 0.404 0.441 0.533 0.512 0.490 0.527 0.616 0. ...
[[10]]	double [1009 x 10]	0.233 0.445 0.387 0.385 0.410 0.429 0.341 0.429 0.306 0.162 0.520 0.219 0.336 0. ...

IV – Final classification.

We create a list of 10 vectors, each vector is of size n, where n is the number of rows in the corresponding test-set.

```
results <- lapply(1:10, function(x) vector(mode="integer", length=nrow(tests[[x]])));
```

Name	Type	Value
results	list [10]	List of length 10
[[1]]	integer [980]	0 0 0 0 0 0 ...
[[2]]	integer [1135]	0 0 0 0 0 0 ...
[[3]]	integer [1032]	0 0 0 0 0 0 ...
[[4]]	integer [1010]	0 0 0 0 0 0 ...
[[5]]	integer [982]	0 0 0 0 0 0 ...
[[6]]	integer [892]	0 0 0 0 0 0 ...
[[7]]	integer [958]	0 0 0 0 0 0 ...
[[8]]	integer [1028]	0 0 0 0 0 0 ...
[[9]]	integer [974]	0 0 0 0 0 0 ...
[[10]]	integer [1009]	0 0 0 0 0 0 ...

To find the results:

```
for(i in 1:10)
{
    for(j in 1:nrow(correlations[[i]]))
    {
        results[[i]][j]=which.max(correlations[[i]][j,])-1
    }
};
```

which.max() returns the index of the largest correlation, so which.max(correlations[[1]][1,]) returns 1, which is the index of the zero pattern, so we subtract 1 to get 0, and that's how it is with all the other patterns.

We get:

Name	Type	Value
results	list [10]	List of length 10
[[1]]	double [980]	0 0 0 0 0 5 ...
[[2]]	double [1135]	1 1 1 1 1 1 ...
[[3]]	double [1032]	2 2 3 2 2 2 ...
[[4]]	double [1010]	3 3 3 3 3 2 ...
[[5]]	double [982]	4 4 4 4 4 0 ...
[[6]]	double [892]	2 3 5 5 5 3 ...
[[7]]	double [958]	0 6 6 6 2 2 ...
[[8]]	double [1028]	7 7 7 7 7 7 ...
[[9]]	double [974]	8 8 8 8 8 8 ...
[[10]]	double [1009]	9 9 9 9 7 9 ...

V – Calculating the accuracy.

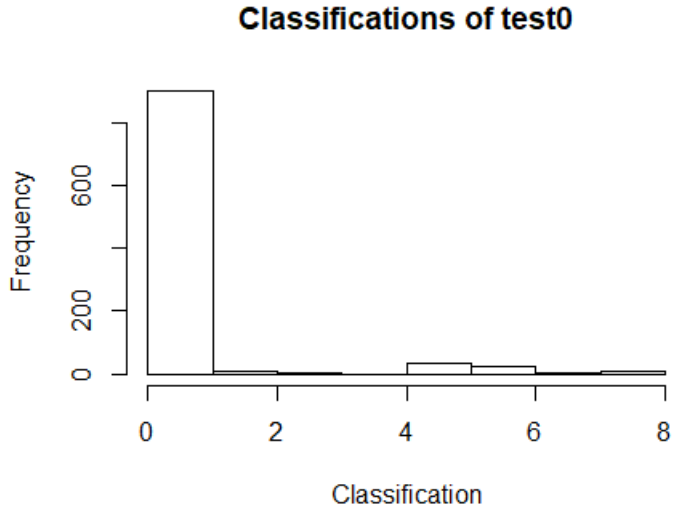
```
correct <- 0;
for(i in 1:10)
        correct<- correct + length(which(results[[i]]==(i-1)));
accuracy <- (correct/10000)*100
```

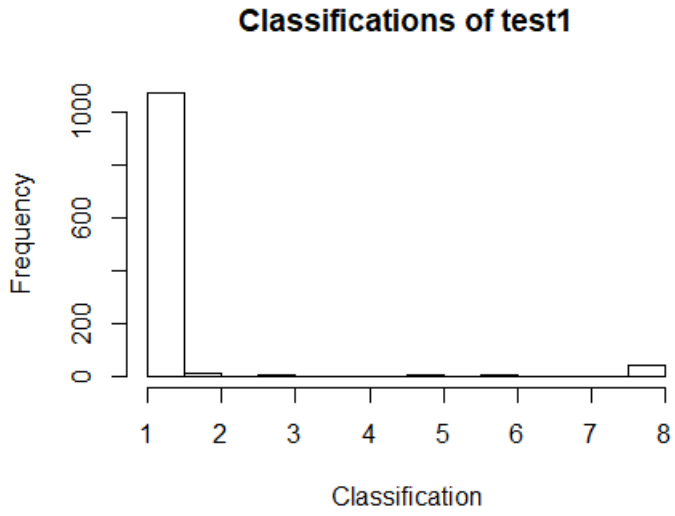
We get:

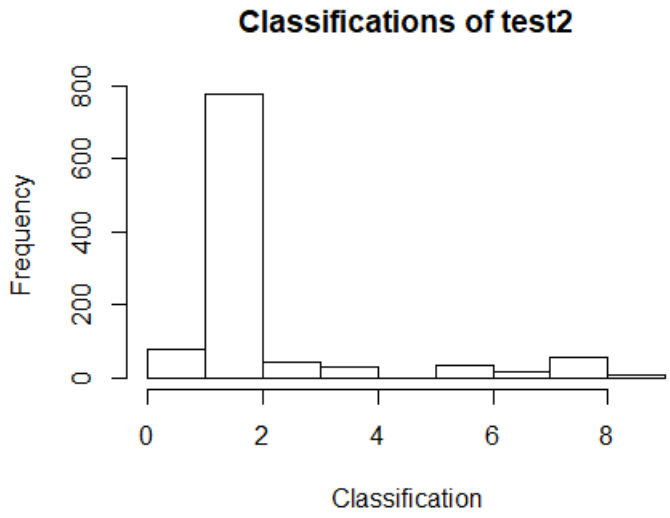
accuracy	82.08
correct	8208

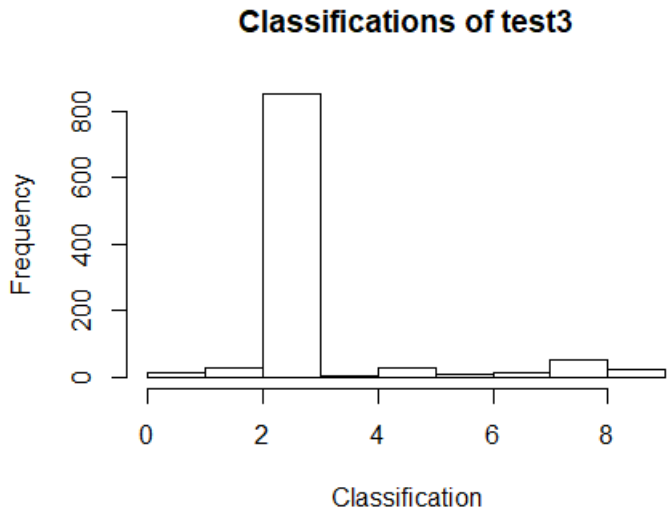
Useful histograms:

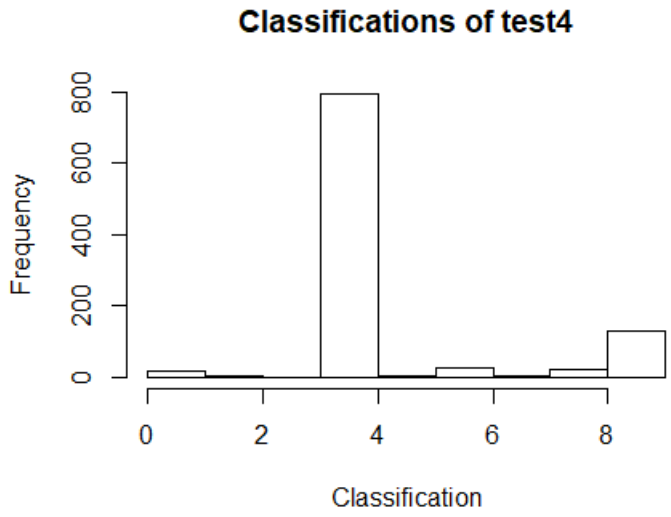
hist(results[[i]], xlab="Classification", main="Classifications of test(i)")

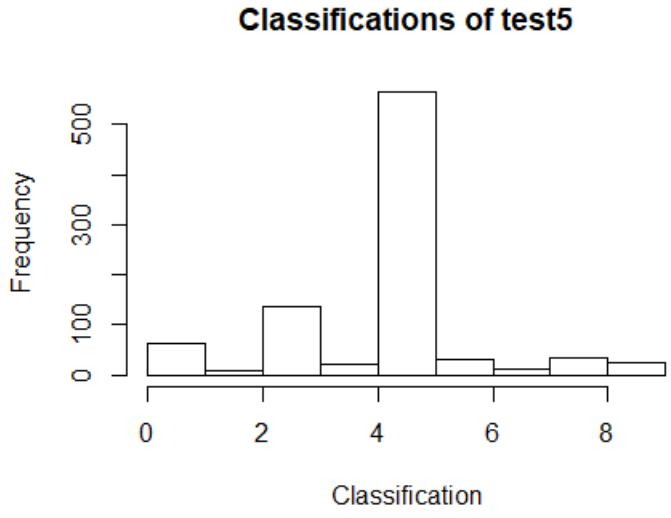


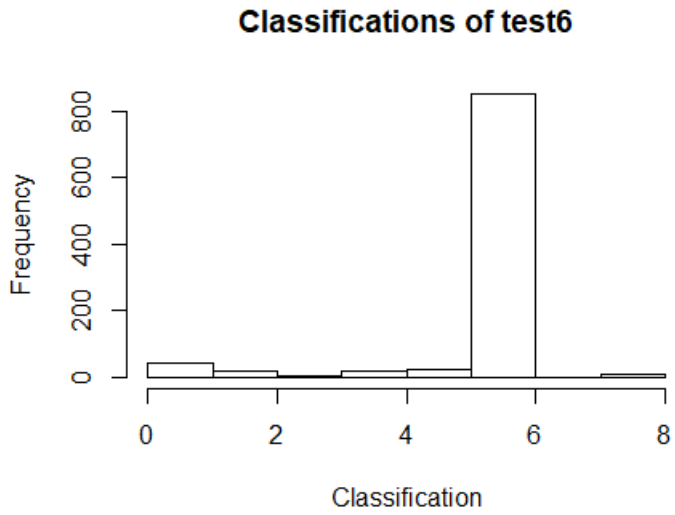


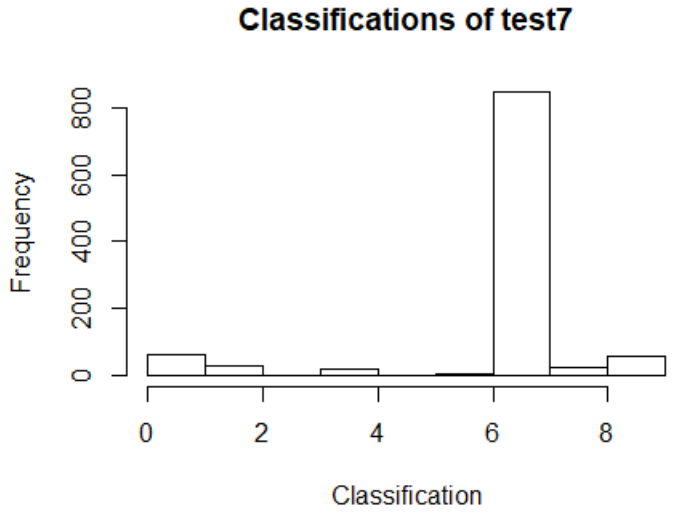

Classifications of test2
Frequency
0
200
400
600
800
0
2
4
6
8
Classification

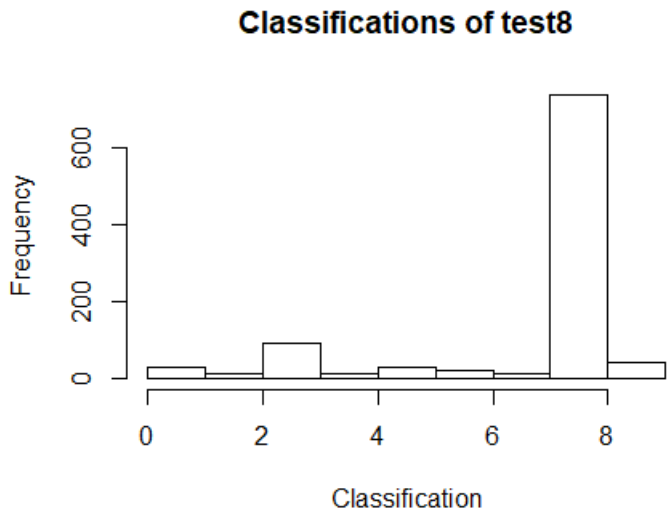

Classifications of test3
Frequency
0
200
400
600
800
0
2
4
6
8
Classification

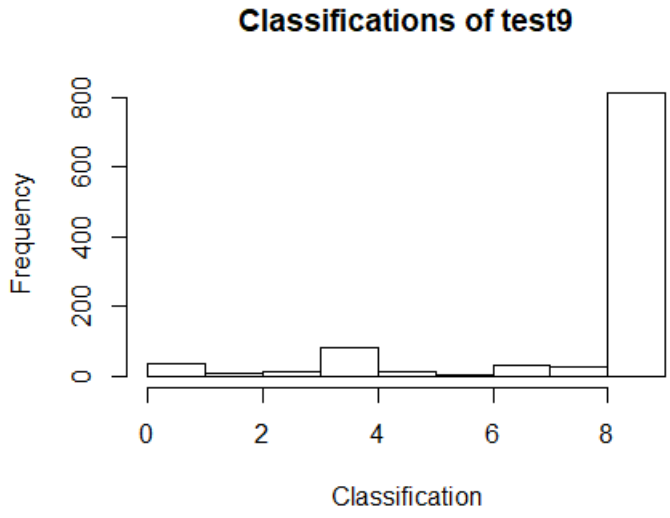

Classifications of test4
Frequency
0
200
400
600
800
0
2
4
6
8
Classification


Classifications of test5
Frequency
0
100
300
500
0
2
4
6
8
Classification


Classifications of test6
Frequency
0
200
400
600
800
0
2
4
6
8
Classification


Classifications of test7
Frequency
0
200
400
600
800
0
2
4
6
8
Classification


Classifications of test8
Frequency
0
200
400
600
0
2
4
6
8
Classification


Classifications of test9
Frequency
0
200
400
600
800
0
2
4
6
8
Classification

Conclusion

By applying correlational k-NN classification, we got an accuracy of 82.08%.