Zip Digit Solution

Matt Richey

5/7/2020

Introduction

Most of this is just a reworking of the Cats/Dogs classification method using SVD and the "hat" matrix. The one big difference is that now we have 10 different categories to classify

Libraries and Data

```
library(tidyverse)
```

```
## -- Attaching packages ------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

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

Training data, pull off the digit values.

```
dataDir <- "~/Dropbox/COURSES/ADM/DATA/ZIP_DIGITS_ESL/"
dataTrainFile <- "zipTrainSmall.csv"
zipTrain.df <- read.csv(file.path(dataDir,dataTrainFile),
                        header=T)
zipTrain.vals <- zipTrain.df[,1]
```

Check the distributio

```
## Should be...
## 250 x 257
dim(zipTrain.df)
```

```
## [1] 250 257
```

```
## Should be something like....
##  0  1  2  3  4  5  6  7  8  9
## 31 35 34 28 18 16 17 29 18 24
table(zipTrain.vals)
```

```
## zipTrain.vals
##  0  1  2  3  4  5  6  7  8  9
## 38 41 26 17 27 17 19 21 21 23
```

Do the same for the Testing data

```
dataTestFile <- "zipTestSmall.csv"
zipTest.df <- read.csv(file.path(dataDir,dataTestFile),
                       header=T)

## Should be
## 1000 x 257
dim(zipTest.df)
```

```
## [1] 1000  257
```

```
zipTest.vals <- round(zipTest.df[,1],0)
```

A couple helper functions to show the images

```
imageConv <- function(flatImage,size=64){
    matrix(flatImage,nrow=size,byrow=T)
}
```

This plots the image, we will use size=16

```
flatImage <- function(dat,size=64) {
  img <- imageConv(dat,size)
  image(img,col=grey.colors(256))
}
```

Shape the data

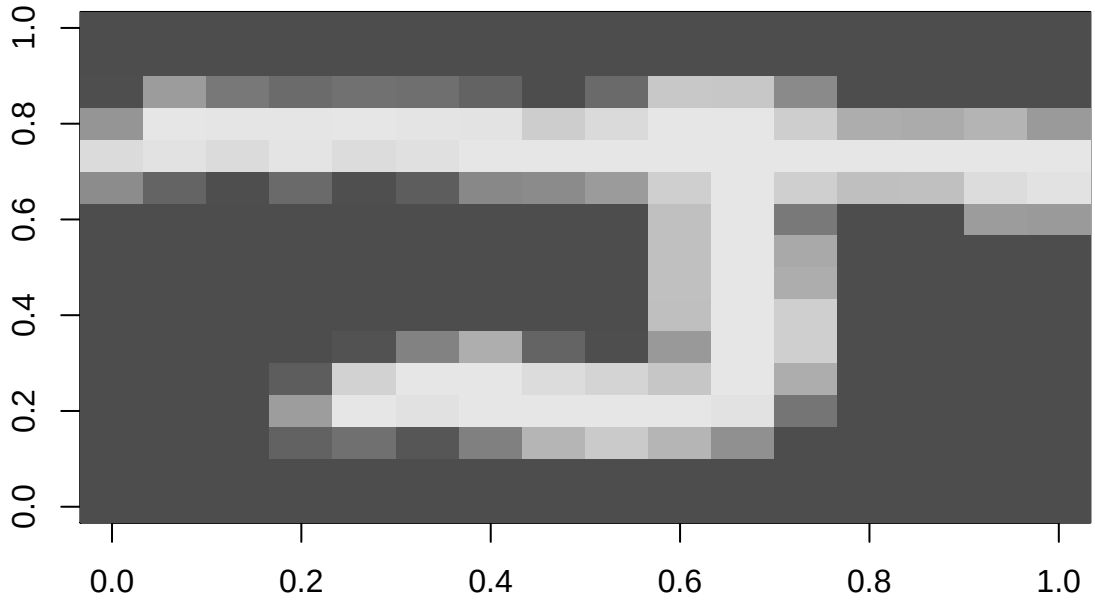
Create matrices of data

```
zipTrain.mat <- t(data.matrix(zipTrain.df[,2:257]))
dim(zipTrain.mat)
```

```
## [1] 256 250
```

Check. . . try this a few times to make sure the digit labels are correct. Note the images are rotate 90 degrees

```
n <- sample(1:ncol(zipTrain.mat),1)
digit <- zipTrain.mat[,n]
flatImage(digit,16)
```




```
zipTrain.vals[n]
```

```
## [1] 4
```

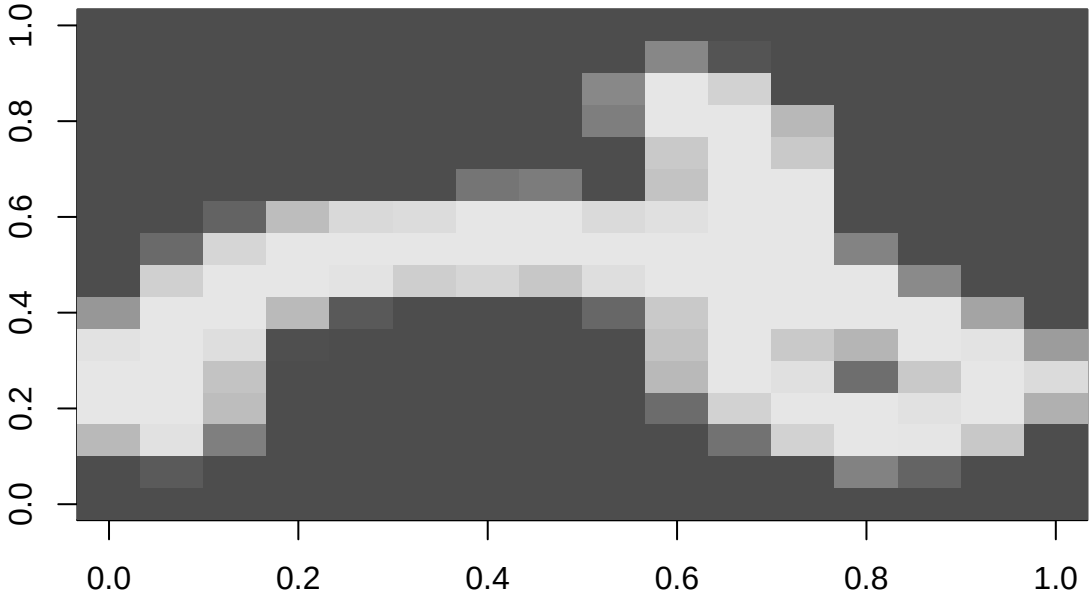
Same for test data

```
zipTest.mat <- t(data.matrix(zipTest.df[,2:257]))
zipTest.vals <- zipTest.df[,1]
dim(zipTest.mat)
```

```
## [1]  256 1000
```

Checking...

```
n <- sample(1:ncol(zipTest.mat),1)
digit <- zipTest.mat[,n]
flatImage(digit,16)
```




```
zipTest.vals[n]
```

```
## [1] 2
```

Hat Matrices

We need one hat matrix for each digit. We could build 10 different matrices. A better way is to use a three-dimensional array for the hat matrices. The first two dimensions are the hat matrices. The third dimension indexes the digit.

```
hatMatrix.mat <- array(dim=c(256,256,10))
```

Now run through the digits and create the spaces and hat matrices.

```
for(i in 0:9){
    zipVals <- zipTrain.vals == i
    zipTrainVal.mat <- zipTrain.mat[,zipVals]
    svdA <- svd(zipTrainVal.mat)
    u <- svdA$u
    hat <- u %*% t(u)
    hatMatrix.mat[,,i+1] <- hat
}
```

Check on Training Data

Run throught the training digits in for each one determine the subspace to which it is closest. It had better be closest to its own subspace. In fact, it should be **in its own subspace**, that is, the distance should be zero.

```
N <- ncol(zipTrain.mat)
dist.mat <- numeric(10)
confusionTrain.mat <- matrix(0,nrow=10, ncol=10)
for(n in 1:N){
    digit <- zipTrain.mat[,n]
    ##flatImage(digit,16)
    (true.val <- zipTrain.vals[n])
    for(i in 1:10){
        dist.mat[i] <-
            log(sum((digit - hatMatrix.mat[,,i] %*% digit)^2))
    }
    (pred.val <- which.min(dist.mat)-1)
    ##    print(c(true.val,pred.val))
    tot <- confusionTrain.mat[true.val+1,pred.val+1]
    confusionTrain.mat[true.val+1,pred.val+1] <- tot + 1
}
```

How's it look?

Do we have the right number of classifications?

```
sum(confusionTrain.mat)
```

```
## [1] 250
```

And...

```
confusionTrain.mat
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]   38    0    0    0    0    0    0    0    0     0
##  [2,]    0   41    0    0    0    0    0    0    0     0
##  [3,]    0    0   26    0    0    0    0    0    0     0
##  [4,]    0    0    0   17    0    0    0    0    0     0
##  [5,]    0    0    0    0   27    0    0    0    0     0
##  [6,]    0    0    0    0    0   17    0    0    0     0
##  [7,]    0    0    0    0    0    0   19    0    0     0
##  [8,]    0    0    0    0    0    0    0   21    0     0
##  [9,]    0    0    0    0    0    0    0    0   21     0
## [10,]    0    0    0    0    0    0    0    0    0    23
```

Perfect!

If wanted to compute the error rate, here's one way to do it. Just sum the diagonal entries (that is the entries that are correct).

```
1-sum(diag(confusionTrain.mat))/sum(confusionTrain.mat)
```

```
## [1] 0
```

Test Digits

Same idea Run throught the test digits in the test set for each one determine the subspace its closest to.

```
N <- ncol(zipTest.mat)
##N <- 100
confusionTest.mat <- matrix(0,nrow=10, ncol=10)
for(n in 1:N){
    digit <- zipTest.mat[,n]
    true.val <- zipTest.vals[n]
    if( true.val %in% c(0:9)){
    for(i in 1:10){
        dist.mat[i] <-
            log(sum((digit - hatMatrix.mat[,,i] %*% digit)^2))
    }
    pred.val <- which.min(dist.mat)-1
    ##    print(c(true.val,pred.val))
    tot <- confusionTest.mat[true.val+1,pred.val+1]
        confusionTest.mat[true.val+1,pred.val+1] <- tot + 1
    }
}
```

Checking..The Confusion Matrix

```
sum(confusionTest.mat)
```

```
## [1] 1000
```

```
confusionTest.mat
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]  164    4    5    0    0    0    0    0    0     0
##  [2,]    0  123    0    0    0    0    0    0    0     0
##  [3,]   13    1   89    0    2    0    0    1    2     0
##  [4,]   14    3    6   61    0    2    0    0    2     0
##  [5,]    0    6    0    0   86    0    2    1    0     4
##  [6,]   23    2    0    2    0   41    1    0    2     4
##  [7,]   12    6    0    0    2    0   66    0    0     0
##  [8,]    0    4    0    0    4    0    0   67    0     5
##  [9,]    7    7    0    3    0    0    0    0   52     6
## [10,]    0    2    0    0    1    0    0    3    1    86
```

Error rates...

Overall error rrate

```r
1-sum(diag(confusionTest.mat))/sum(confusionTest.mat)
```

```
## [1] 0.165
```

Per digit error rate.

```r
errs <- 1-diag(confusionTest.mat)/apply(confusionTest.mat,1,sum)
for(i in 0:9){
    print(sprintf("Digit: %s Error rate: %0.3f",i,round(errs[i+1],3)))
}
```

```
## [1] "Digit: 0 Error rate: 0.052"
## [1] "Digit: 1 Error rate: 0.000"
## [1] "Digit: 2 Error rate: 0.176"
## [1] "Digit: 3 Error rate: 0.307"
## [1] "Digit: 4 Error rate: 0.131"
## [1] "Digit: 5 Error rate: 0.453"
## [1] "Digit: 6 Error rate: 0.233"
## [1] "Digit: 7 Error rate: 0.162"
## [1] "Digit: 8 Error rate: 0.307"
## [1] "Digit: 9 Error rate: 0.075"
```

The overall error rate.

```r
1-sum(diag(confusionTest.mat))/sum(confusionTest.mat)
```

```
## [1] 0.165
```

Conclusion

Here's a summary of the results.

- The method does a good job with digits such as 0, 1, and 2. The digit 1 is classified perfectly.
- The digit 3 is problematic, it often gets confused with 0.
- The digit 5 the hardest digit (error rate of 0.453), it is also confused with 0.
- The digit 8 is troublesome, it gets confused with 0, 1, and 9.