STAT4051_HW4

Ruwiada Al Harrasi

2023-11-03

R Markdown

```
## For .rdata type, use load() function,
## and use its name to refer the dataset
load(file='~/Downloads/Digits.Rda')
## the dimension of the dataset
dim(digit.dt)
```

```
## [1] 360 256
```

(a) Apply PCA to the data set. Select 8 PCs. Note that each PC correspond to a 256-dim loading vector, as the linear combination coefficient for each of the 256 pixels (variables). Thus we can plot each loading vector as a 16 × 16 headmap image, with the pixel positions corresponding to the variable position in the image. Again, by using different colors for different values, you will be able to visualize the 8 loadings by eight pictures. Generate these 8 pictures.

```
# 16×16 matrix
# nrow is the desired number of rows, you can use ncol as well.
a <- c(1,2,3,4)
matrix(data=a, nrow=2, byrow=TRUE)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

```
matrix(data=a, nrow=2, byrow=FALSE)
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
## create the image matrix of the third image
image1 <- matrix(data=digit.dt[3, ], byrow=FALSE, nrow=16)
image1
```

```
##         [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]  [,10]
##  [1,] -1.000 -1.000 -1.000 -1.000 -1.000 -0.965 -0.230  0.127  0.381  0.636
```

```
##  [2,] -1.000 -1.000 -1.000 -0.906  0.107  0.749  0.996  1.000  1.000  1.000
##  [3,] -1.000 -1.000 -0.746  0.684  1.000  1.000  1.000  1.000  1.000  1.000
##  [4,] -1.000 -1.000 -0.915  0.696  1.000  1.000  0.987  0.832  0.625  0.371
##  [5,] -1.000 -1.000 -1.000 -0.067  1.000  1.000  0.945 -0.762 -0.999 -1.000
##  [6,] -1.000 -1.000 -1.000 -0.849  0.749  1.000  1.000  0.329 -1.000 -1.000
##  [7,] -1.000 -1.000 -1.000 -1.000 -0.608  0.280  0.616  0.085 -1.000 -1.000
##  [8,] -0.937 -0.517 -0.962 -1.000 -1.000 -1.000 -0.999 -1.000 -1.000 -1.000
##  [9,]  0.252  1.000  0.321 -0.871 -1.000 -1.000 -1.000 -1.000 -1.000 -1.000
## [10,] -0.073  1.000  0.947 -0.598 -1.000 -1.000 -1.000 -1.000 -1.000 -1.000
## [11,] -0.430  1.000  0.593 -1.000 -1.000 -1.000 -1.000 -1.000 -1.000 -1.000
## [12,] -0.875  0.656  0.951  0.330 -0.387 -0.955 -1.000 -1.000 -1.000 -0.994
## [13,] -1.000 -0.625  0.959  1.000  1.000  0.851  0.560  0.295 -0.110  0.245
## [14,] -1.000 -1.000 -0.674  0.706  1.000  1.000  1.000  1.000  1.000  1.000
## [15,] -1.000 -1.000 -1.000 -0.732  0.215  0.873  1.000  1.000  1.000  1.000
## [16,] -1.000 -1.000 -1.000 -1.000 -1.000 -0.941 -0.351  0.366  0.424  0.515
##        [,11]  [,12]  [,13]  [,14]  [,15]  [,16]
##  [1,] -0.047 -0.753 -1.000 -1.000 -1.000 -1.000
##  [2,]  1.000  0.952 -0.399 -1.000 -1.000 -1.000
##  [3,]  1.000  1.000  0.909 -0.524 -1.000 -1.000
##  [4,]  0.704  1.000  1.000  0.998 -0.540 -1.000
##  [5,] -0.997  0.173  1.000  1.000  0.657 -1.000
##  [6,] -1.000 -0.943  0.782  1.000  0.993 -0.487
##  [7,] -1.000 -1.000  0.258  1.000  1.000  0.352
##  [8,] -1.000 -1.000  0.199  1.000  1.000  0.403
##  [9,] -1.000 -1.000 -0.575  1.000  1.000  0.403
## [10,] -1.000 -1.000 -0.237  1.000  1.000  0.403
## [11,] -1.000 -0.893  0.788  1.000  1.000  0.402
## [12,] -0.690  0.673  1.000  1.000  1.000 -0.383
## [13,]  0.821  1.000  1.000  1.000  0.765 -0.898
## [14,]  1.000  1.000  1.000  1.000 -0.281 -1.000
## [15,]  1.000  0.943  0.123 -0.139 -0.980 -1.000
## [16,]  0.607 -0.397 -1.000 -1.000 -1.000 -1.000
```

```
dim(image1)
```

```
## [1] 16 16
```

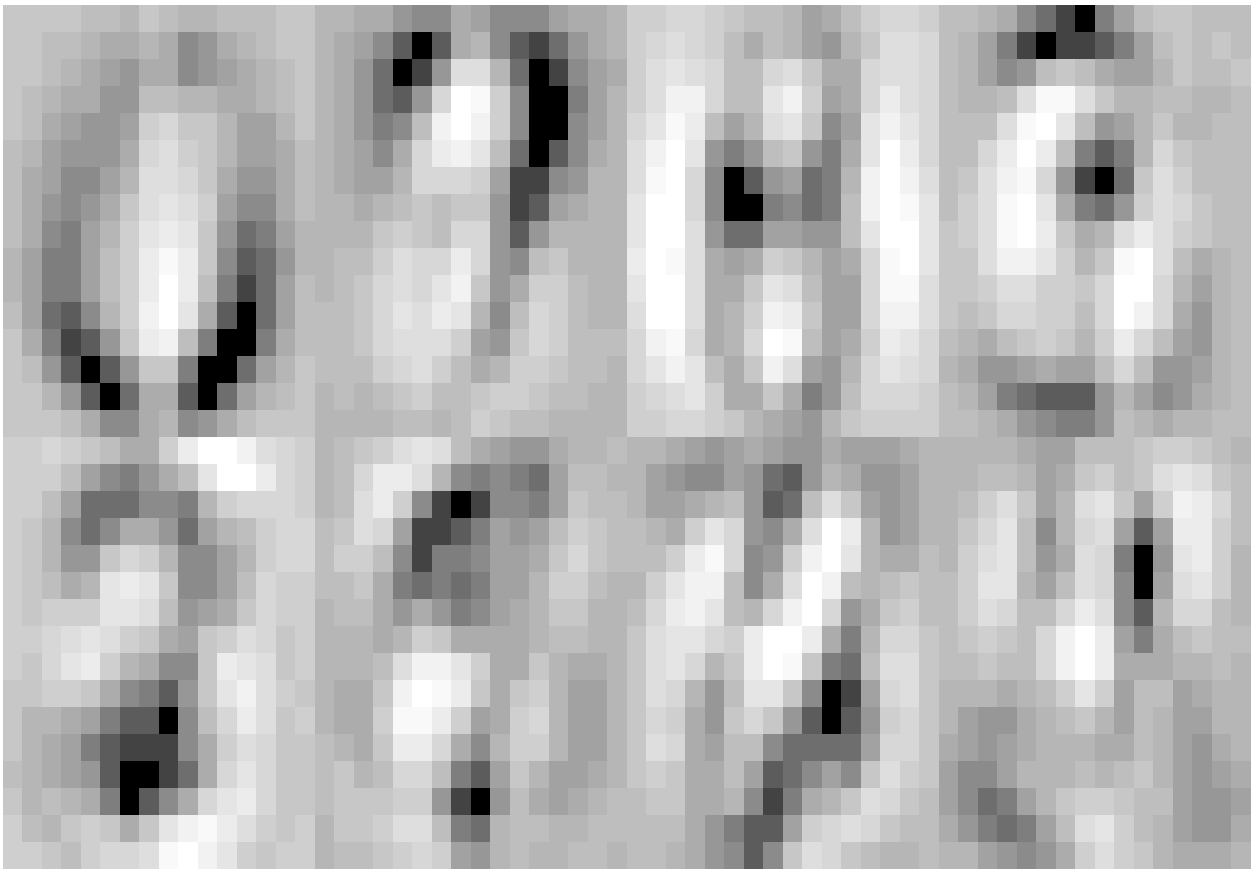
```
# Assuming you have already performed PCA
pca <- prcomp(x=digit.dt, center=TRUE)
pca$x[1, 1:8]
```

```
##        PC1        PC2        PC3        PC4        PC5        PC6        PC7
## -2.5600948  1.8749549 -3.6258291  1.8671183  1.0647800  0.9030784 -1.6483135
##        PC8
## -2.3180435
```

```
par(mfrow=c(4,4), mar=c(0.2,0.2,0.2,0.2))
for (j in 1:8){
  image=matrix(data=digit.dt[j, ], ncol=16, byrow=F)
  image(image[, ncol(image):1], col=grey.colors(n=20, start=0, end=1), axes=FALSE)
}
```

```
# Assuming you have already performed PCA
pca <- prcomp(x=digit.dt, center=TRUE)
par(mfrow=c(2,4), mar=c(0,0,0,0))
for (j in 1:8){
image=matrix(pca$rotation[, j], ncol=16, byrow=F)
  image(image[, ncol(image):1], col=grey.colors(n=20, start=0, end=1), axes=FALSE)

}
```



#NMF

```
library(registry)
library(rngtools)
library(cluster)
library(Biobase)
```

```
## Loading required package: BiocGenerics
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unsplit, which.max, which.min
```

```
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
```

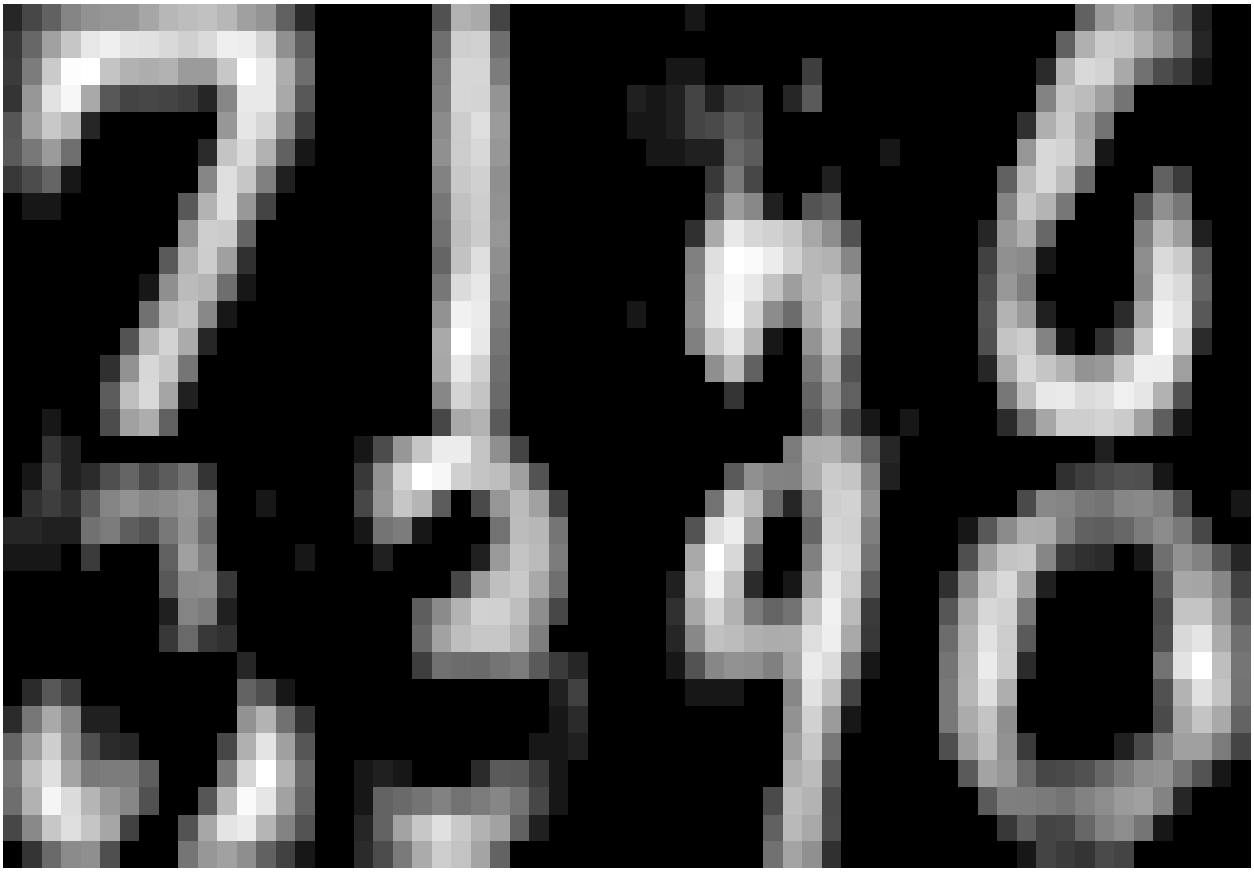
```
library(NMF)
```

```
## NMF - BioConductor layer [OK] | Shared memory capabilities [NO: bigmemory] | Cores 2/2

##   To enable shared memory capabilities, try: install.extras('
## NMF
## ')
```

```
set.seed(4051)
digit.dt = digit.dt + 1.01 ## make this matrix non-negative
res <- nmf(x = digit.dt, rank=8)
W = basis(res)
H = coef(res)
```

```
par(mfrow=c(2,4) , mar=c(0,0,0,0))
for(j in 1:8){
  basis.image <- matrix(data=H[j, ],ncol=16, byrow=T)
  image(t(basis.image[nrow(basis.image):1, ]), col = gray.colors(n = 200, start = 0, end = 1), axes = F
}
```



I've noticed that NMF performs exceptionally well on this dataset compared to PCA. This advantage is

primarily due to NMF's ability to leverage non-negative values within the data, including the abundance of zeros in the dataset. NMF's inherent constraint on non-negativity allows it to represent zero values as black, which significantly enhances the interpretability of figures within the image. NMF excels in learning the fundamental components or features of the data, and this constraint results in a parts-based representation that can be far more interpretable and visually pleasing than the output of PCA.