PCA HW

Kristiyan Dimitrov, Srividya Ganapathi, Shreyashi Ganguly, Greesham Simon, Joe Zhang

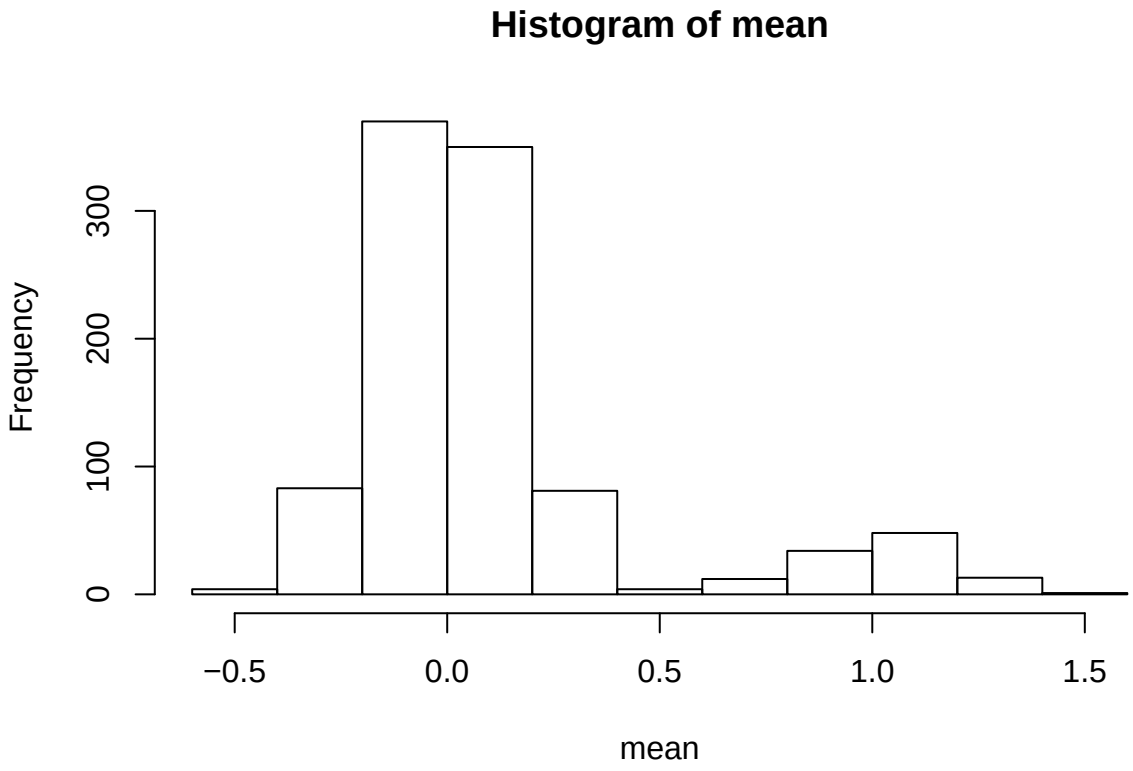
3/8/2020

Problem 1

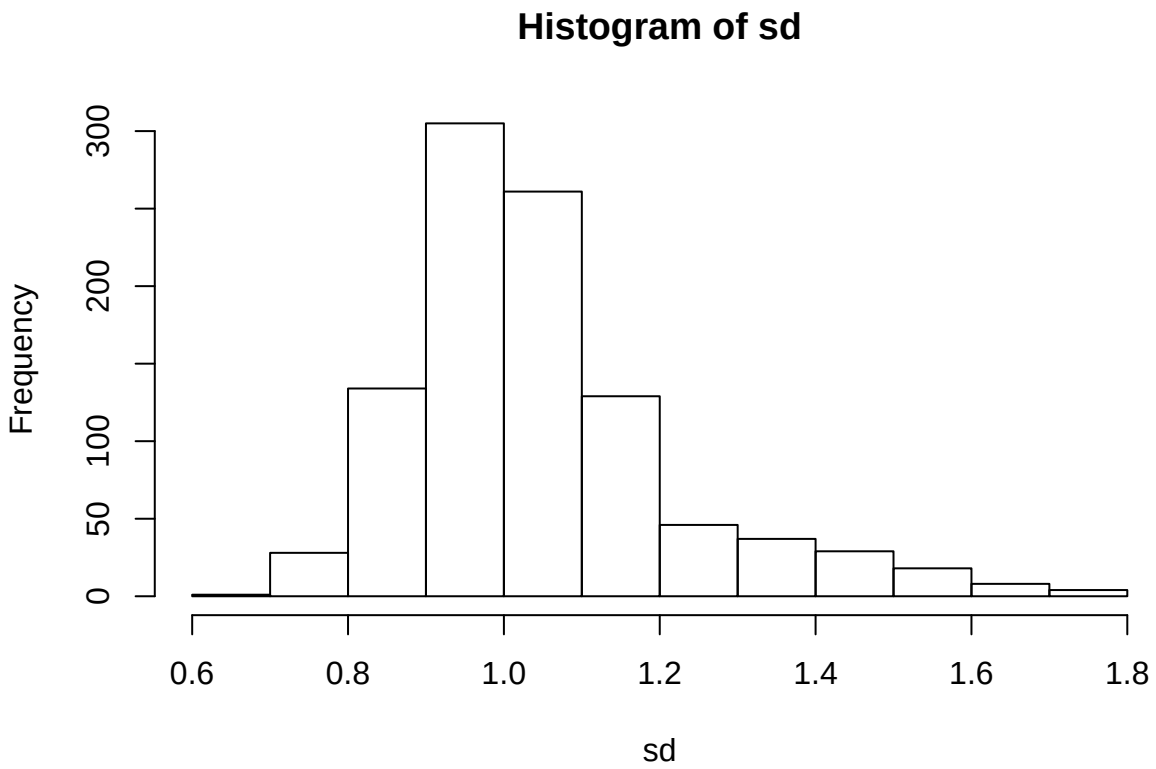
```
gene = read.csv('gene.csv')
dim(gene)
```

```
## [1]   40 1001
```

```
#Checking to see if all fields are measured on comparable scales
mean <- apply(gene[,-1001],2,mean)
hist(mean)
```




```
sd <- apply(gene[,-1001],2,sd)
hist(sd)
```



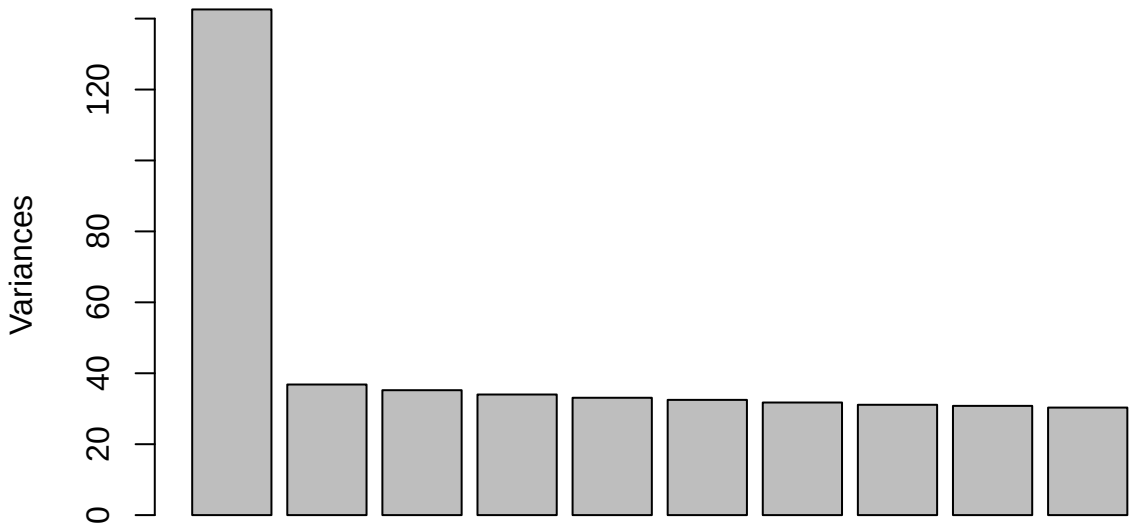
All the X variables are measurements of comparable units. Also, almost all the fields have mean 0 and standard deviation 1, hence not standardizing.

a) PCA on data

Note that we are looking at only the first 1000 columns, not including response variable sick. Also, we're not scaling the data, because we assume all the variables are using the same units.

```
fitPrComp = prcomp(gene[,1:1000], scale. = FALSE)
```

```
plot(fitPrComp)
```




```
summary(fitPrComp)
```

```
## Importance of components:
##                             PC1     PC2     PC3     PC4     PC5     PC6
## Standard deviation     11.9409 6.06818 5.93476 5.83115 5.75209 5.70031
## Proportion of Variance  0.1267 0.03271 0.03129 0.03021 0.02939 0.02887
## Cumulative Proportion   0.1267 0.15939 0.19068 0.22089 0.25029 0.27915
##                             PC7     PC8     PC9    PC10    PC11    PC12
## Standard deviation     5.63448 5.57726 5.54943 5.50625 5.48852 5.46025
## Proportion of Variance 0.02821 0.02764 0.02736 0.02694 0.02676 0.02649
## Cumulative Proportion  0.30736 0.33499 0.36236 0.38929 0.41605 0.44254
##                            PC13    PC14    PC15    PC16    PC17    PC18
## Standard deviation     5.40230 5.33441 5.27756 5.21594 5.20000 5.15140
## Proportion of Variance 0.02593 0.02528 0.02475 0.02417 0.02402 0.02358
## Cumulative Proportion  0.46847 0.49375 0.51850 0.54267 0.56669 0.59027
##                            PC19    PC20    PC21    PC22    PC23    PC24
## Standard deviation     5.11600 5.05591 5.03836 5.01868 4.95965 4.91393
## Proportion of Variance 0.02325 0.02271 0.02255 0.02238 0.02185 0.02145
## Cumulative Proportion  0.61352 0.63623 0.65878 0.68116 0.70301 0.72447
##                            PC25    PC26    PC27    PC28    PC29    PC30
## Standard deviation     4.86397 4.81796 4.80811 4.73485 4.70098 4.65564
## Proportion of Variance 0.02102 0.02062 0.02054 0.01992 0.01963 0.01926
## Cumulative Proportion  0.74548 0.76611 0.78665 0.80656 0.82620 0.84545
##                            PC31    PC32    PC33    PC34    PC35    PC36
## Standard deviation     4.61621 4.56733 4.53032 4.49528 4.36502 4.35858
## Proportion of Variance 0.01893 0.01853 0.01823 0.01795 0.01693 0.01688
## Cumulative Proportion  0.86439 0.88292 0.90115 0.91910 0.93603 0.95291
##                            PC37    PC38    PC39      PC40
## Standard deviation     4.26700 4.20277 4.13922 5.251e-15
## Proportion of Variance 0.01618 0.01569 0.01522 0.000e+00
## Cumulative Proportion  0.96909 0.98478 1.00000 1.000e+00
```

Looking at the first 2 Principal Components, it appears that PC1 & PC2 together explain 15.94% of the variance in the data.

b) Generating a Scatter Plot of the first 2 Principal components

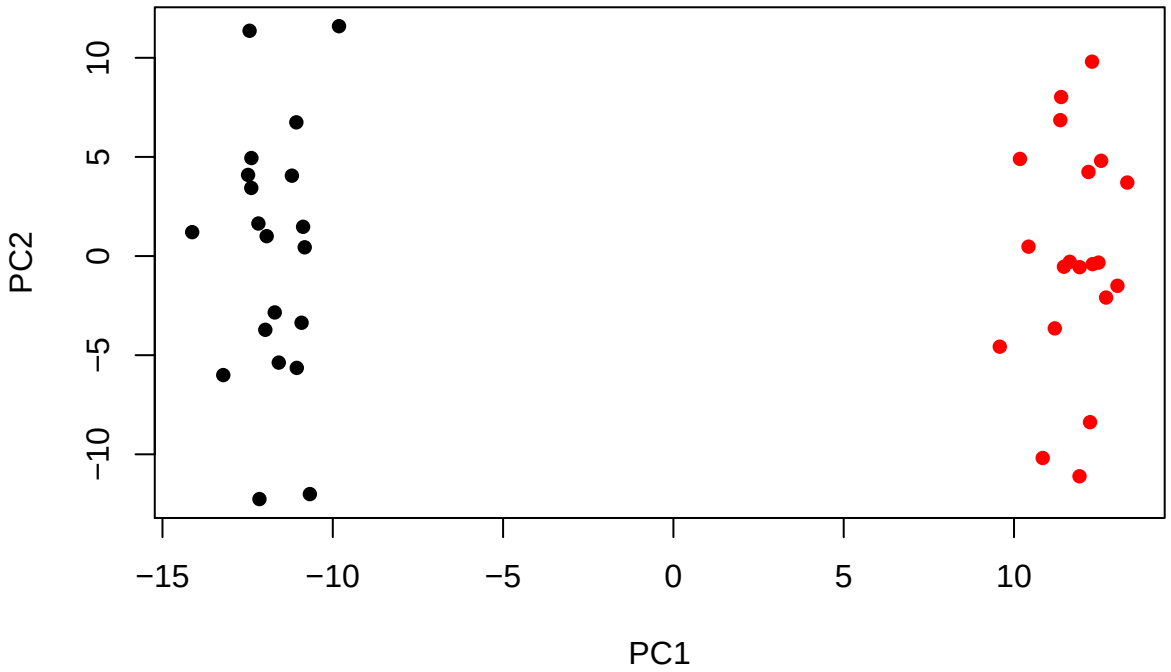
First we take out the scores for the 40 points in PC1 & PC2

```
first2PC = fitPrComp$x[,1:2]
head(first2PC)
```

```
##           PC1        PC2
## V1 -12.48740   4.092325
## V2 -11.20092   4.055472
## V3 -12.39022   4.943398
## V4  -9.81745  11.594869
## V5 -12.15385 -12.257361
## V6 -11.58635  -5.373251
```

Even with just two Principal components, we can see very nice separation in the data. In other words, first two principal components can adequately capture the distinction between sick and healthy tissue samples.

```
plot(fitPrComp$x[,1:2], col = gene[,1001]+1, pch = 16)
```



Using this data for prediction with a Logictic Regression or a Tree model would probably be better. First, Logistic Regression wouldn't even run with more variables than observations. And moreover, using the PC results would be more computationally efficient.

Problem 2

```
X=matrix(c(1:4, 1,4,9,16), nrow=4)
X
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    4
## [3,]    3    9
## [4,]    4   16
```

a)

Finding X_t * X

```
first = t(X) %*% X
first
```

```
##      [,1] [,2]
## [1,]   30  100
## [2,]  100  354
```

Finding X * X_t

```
second = X %*% t(X)
second
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    6   12   20
## [2,]    6   20   42   72
## [3,]   12   42   90  156
## [4,]   20   72  156  272
```

b) Eigenvalues & Eigenvectors of X_t * X

```
eigenFirst = eigen(first, symmetric = T)
eigenFirst
```

```
## eigen() decomposition
## $values
## [1] 382.37857   1.62143
##
## $vectors
##           [,1]       [,2]
## [1,] 0.2730054 -0.9620125
## [2,] 0.9620125  0.2730054
```

c) Eigenvalues & Eigenvectors of X * X_t

```
eigenSecond = eigen(second, symmetric = F)
eigenSecond
```

```
## eigen() decomposition
## $values
## [1]  3.823786e+02  1.621430e+00  8.579295e-15 -6.814541e-15
##
## $vectors
##            [,1]       [,2]       [,3]       [,4]
## [1,] 0.06315773  0.5410964  0.7869830 -0.2031339
## [2,] 0.22470839  0.6533954 -0.2130558 -0.4513541
## [3,] 0.48465200  0.3368969 -0.5029085  0.8049394
## [4,] 0.84298854 -0.4083989  0.2869636 -0.3272440
```

d) Commenting on the above results

We note that the first two eigen values of X * X_t are the same as those of X_t * X. X * X_t has two additional eigen values (owing to it's 4x4 structure), but they are approximately 0.

We also see that the first eigen vector for X_t * X is much bigger than the second. This suggests that most of the variance will be explained with the first PC.

```
eigenFirst$values
```

```
## [1] 382.37857   1.62143
```

```
eigenSecond$values
```

```
## [1]  3.823786e+02  1.621430e+00  8.579295e-15 -6.814541e-15
```

Problem 3

```
X = matrix(c(1,3,5,0,1, 2,4,4,2,3, 3,5,3,4,5), nrow=5)
X
```

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

a)

Finding X_t * X

```r
first = t(X) %*% X
first
```

```
##      [,1] [,2] [,3]
## [1,]   36   37   38
## [2,]   37   49   61
## [3,]   38   61   84
```

Finding X * X_t

```r
second = X %*% t(X)
second
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   14   26   22   16   22
## [2,]   26   50   46   28   40
## [3,]   22   46   50   20   32
## [4,]   16   28   20   20   26
## [5,]   22   40   32   26   35
```

b) Eigenvalues & Eigenvectors

of X_t * X

```r
eigenFirst = eigen(first, symmetric = T)
eigenFirst
```

```
## eigen() decomposition
## $values
## [1] 1.535670e+02 1.543300e+01 1.421085e-14
##
## $vectors
##            [,1]       [,2]       [,3]
## [1,] -0.4092828  0.8159785  0.4082483
## [2,] -0.5634593  0.1258846 -0.8164966
## [3,] -0.7176358 -0.5642094  0.4082483
```

of X * X_t

```r
eigenSecond = eigen(second, symmetric = F)
eigenSecond
```

```
## eigen() decomposition
## $values
## [1]  1.535670e+02+0.00000e+00i  1.543300e+01+0.00000e+00i
## [3] -2.293820e-15+7.35969e-16i -2.293820e-15-7.35969e-16i
```

```
## [5] -1.860586e-15+0.00000e+00i
##
## $vectors
##              [,1]          [,2]                    [,3]
## [1,] 0.2976957+0i  0.1590639+0i  0.71148512+0.00000000i
## [2,] 0.5705086+0i -0.0332003+0i -0.17609014-0.03281437i
## [3,] 0.5207430+0i -0.7358566+0i  0.06368045-0.03218228i
## [4,] 0.3225785+0i  0.5103921+0i  0.26575971-0.25903848i
## [5,] 0.4589849+0i  0.4142600+0i -0.50161697+0.25935453i
##                          [,4]           [,5]
## [1,]  0.71148512+0.00000000i  0.94131607+0i
## [2,] -0.17609014+0.03281437i -0.17481584+0i
## [3,]  0.06368045+0.03218228i -0.04034212+0i
## [4,]  0.26575971+0.25903848i -0.18826321+0i
## [5,] -0.50161697-0.25935453i -0.21515796+0i
```

c) Confirming X_t * X == Gamma * Lambda * Gamma_Transpose

Creating LAMBDA (Eigenvalues diagonal matrix)

```
LAMBDA = Diagonal(3, eigenFirst$values)
LAMBDA
```

```
## 3 x 3 diagonal matrix of class "ddiMatrix"
##      [,1]    [,2]   [,3]
## [1,] 153.567      .            .
## [2,]       . 15.433            .
## [3,]       .      . 1.421085e-14
```

Creating GAMMA (EigenVector Matrix)

```
GAMMA = eigenFirst$vectors
GAMMA
```

```
##            [,1]       [,2]       [,3]
## [1,] -0.4092828  0.8159785  0.4082483
## [2,] -0.5634593  0.1258846 -0.8164966
## [3,] -0.7176358 -0.5642094  0.4082483
```

Checking equation

```
round(GAMMA %*% LAMBDA %*% t(GAMMA),5) == first
```

```
## 3 x 3 Matrix of class "lgeMatrix"
##      [,1] [,2] [,3]
## [1,] TRUE TRUE TRUE
## [2,] TRUE TRUE TRUE
## [3,] TRUE TRUE TRUE
```

Yes! The equation is confirmed.

d) SVD

```
SVD = svd(X)
```

We see that the Singular Value Decomposition gives us the same results as b) i.e. the eigen vectors of X_t * X. Moreover, as expected, they differ only up to a sign difference.

```
SVD$v
```

```
##            [,1]       [,2]       [,3]
## [1,] -0.4092828 -0.8159785 -0.4082483
## [2,] -0.5634593 -0.1258846  0.8164966
## [3,] -0.7176358  0.5642094 -0.4082483
```

```
eigenFirst$vectors
```

```
##            [,1]       [,2]       [,3]
## [1,] -0.4092828  0.8159785  0.4082483
## [2,] -0.5634593  0.1258846 -0.8164966
## [3,] -0.7176358 -0.5642094  0.4082483
```

e) Confirming SVD Equation

We perform the calculations below and observe that they are equal to X!

```
round(SVD$u %*% Diagonal(3, SVD$d) %*% t(SVD$v), 2)
```

```
## 5 x 3 Matrix of class "dgeMatrix"
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    3    4    5
## [3,]    5    4    3
## [4,]    0    2    4
## [5,]    1    3    5
```

```
round(SVD$u %*% Diagonal(3, SVD$d) %*% t(SVD$v), 2) == X
```

```
## 5 x 3 Matrix of class "lgeMatrix"
##      [,1] [,2] [,3]
## [1,] TRUE TRUE TRUE
## [2,] TRUE TRUE TRUE
## [3,] TRUE TRUE TRUE
## [4,] TRUE TRUE TRUE
## [5,] TRUE TRUE TRUE
```

f)

Setting the smallest singular value to 0. Note that the 3rd Singular value is already 0, because the rank of our matrix is 2 i.e. only 2 of the columns are linearly independent (more specifically two times the second minus the third gives us the first)

```
dModified = diag(c(SVD$d[1],0,0))
dModified
```

```
##          [,1] [,2] [,3]
## [1,] 12.39222    0    0
## [2,]  0.00000    0    0
## [3,]  0.00000    0    0
```

```
Xhat = SVD$u %*% dModified %*% t(SVD$v)
Xhat
```

```
##          [,1]     [,2]     [,3]
## [1,] 1.509889 2.078663 2.647437
## [2,] 2.893574 3.983581 5.073588
## [3,] 2.641167 3.636093 4.631018
## [4,] 1.636093 2.252407 2.868722
## [5,] 2.327935 3.204866 4.081797
```

Why is this a one-dimensional representation of X? Well, we still see that two times the second column of Xhat minus the 3rd column gives us the 1st column

```
Xhat[,2]*2 - Xhat[,3]
```

```
## [1] 1.509889 2.893574 2.641167 1.636093 2.327935
```

```
Xhat[,1]
```

```
## [1] 1.509889 2.893574 2.641167 1.636093 2.327935
```

Furthermore, we see that the 2nd column is an exact scalar multiple of the 1st column. In fact, all three columns are scalar multiples of each other, therefore the rank of Xhat is 1 and the span is a 1-dimensional line.

```
Xhat[,3] / Xhat[,2]
```

```
## [1] 1.273625 1.273625 1.273625 1.273625 1.273625
```

```
Xhat[,2] / Xhat[,1]
```

```
## [1] 1.376699 1.376699 1.376699 1.376699 1.376699
```

Furthermore, we can calculate the values of Xhat on this one dimensional space in two ways. In other words, these would be the principal component scores (the matrix Y as defined on page 24 of the notes) Calculating the one-dimensional representation of X in two ways:

First way: X * V[1] i.e. projecting X on the first principal component

```
Xhat1 =  X %*% SVD$v[,1]
Xhat1
```

```
##           [,1]
## [1,] -3.689109
## [2,] -7.069865
## [3,] -6.453159
## [4,] -3.997462
## [5,] -5.687840
```

Second Way: U * D_modified, where D_modified is the diagonal matrix which contains only the first singular value.

```
Xhat2 = SVD$u %*% dModified
Xhat2
```

```
##           [,1] [,2] [,3]
## [1,] -3.689109    0    0
## [2,] -7.069865    0    0
## [3,] -6.453159    0    0
## [4,] -3.997462    0    0
## [5,] -5.687840    0    0
```

We confirm we get the same result!

g) Sum of Squared Values of Xhat

The sum of the squared values of Xhat is precisely the square of the singular value.

```
sum(Xhat^2)
```

```
## [1] 153.567
```

```
sum(dModified^2)
```

```
## [1] 153.567
```

It is also precisely the square sum of the principal component scores!

```
sum(Xhat1^2)
```

```
## [1] 153.567
```

In other words the variance of the principal component scores is precisely equal to the square value of the singular value and equal to the eigen vector along that principal component. This makes sense, the squared singular value represents the variance along a given principal component. So when we project our data onto that principal component, we would expect precisely that amount of variance.

h) X - Xhat, square the values & add them up

We observe that the 'sum of squared errors' X-Xhat is precisely the variance represented by the squared value of the 2nd singular value (the one we set to 0 in part (f))

```
sum((X-Xhat)^2)
```

```
## [1] 15.433
```

```
SVD$d[2]^2
```

```
## [1] 15.433
```

This makes sense, that's precisely the amount of variance we lost by ignoring that singular value and projecting into a lower-dimensional subspace.

i) How much energy is preserved with the first singular value?

We assume this means what proportion of the squared singular values i.e. what percent of the variance is explained, because the squared singular values are the eigenvectors of X_t * X. Below we compute in two different, but equivalent ways.

```
(SVD$d[1])^2 / sum(SVD$d^2)
```

```
## [1] 0.9086805
```

```
sum(Xhat1^2) / (sum(Xhat1^2) + sum((X-Xhat)^2))
```

```
## [1] 0.9086805
```

It appears that 90.86% of the 'energy' i.e. variance is preserved with the first singular value.

Problem 4

a) Commenting on the Correlation Matrix R

```
R = matrix(c(
1, .4919, .2636, .4653, -.2277, .0652,
.4919, 1, .3127, .3506, -.1917, .2045,
.2635, .3127, 1, .4108, .0647, .2493,
.4653, .3506, .4108, 1, -.2249, .2293,
-.2277, -.1917, .0647, -.2249, 1, -.2144,
.0652, .2045, .2493, .2293, -.2144, 1),
byrow=T, ncol=6)

fishNames = c('Bluegill','Black crappie', 'Smallmouth bass',
'Largemouth bass', 'Walleye', 'Northern pike')
rownames(R) = fishNames
colnames(R) = fishNames
R
```

```
##                 Bluegill Black crappie Smallmouth bass Largemouth bass
## Bluegill          1.0000        0.4919          0.2636          0.4653
## Black crappie     0.4919        1.0000          0.3127          0.3506
## Smallmouth bass   0.2635        0.3127          1.0000          0.4108
## Largemouth bass   0.4653        0.3506          0.4108          1.0000
## Walleye          -0.2277       -0.1917          0.0647         -0.2249
## Northern pike     0.0652        0.2045          0.2493          0.2293
##                 Walleye Northern pike
## Bluegill        -0.2277        0.0652
## Black crappie   -0.1917        0.2045
## Smallmouth bass  0.0647        0.2493
## Largemouth bass -0.2249        0.2293
## Walleye          1.0000       -0.2144
## Northern pike   -0.2144        1.0000
```

Let's look at the first 4 rows & columns, the centrarchids family

```
R[1:4,1:4]
```

```
##                 Bluegill Black crappie Smallmouth bass Largemouth bass
## Bluegill          1.0000        0.4919          0.2636          0.4653
## Black crappie     0.4919        1.0000          0.3127          0.3506
## Smallmouth bass   0.2635        0.3127          1.0000          0.4108
## Largemouth bass   0.4653        0.3506          0.4108          1.0000
```

We observe that the correlations within the centrarchids family is much greater than with the other types of fish (walleye and Northern pike). In fact, Walleye is negatively correlated with most of the centrarchids family.

b) PCA on the centrarchid family

```
eigen(R[1:4,1:4])
```

```
## eigen() decomposition
## $values
## [1] 2.1539189 0.7875498 0.6156590 0.4428723
##
## $vectors
##            [,1]       [,2]       [,3]       [,4]
## [1,] -0.5265384 -0.4571582  0.2491110 -0.6719808
## [2,] -0.5033080 -0.4119833 -0.6142538  0.4467865
## [3,] -0.4427711  0.7584337 -0.3680028 -0.3056057
## [4,] -0.5228691  0.2146031  0.6520812  0.5053996
```

PC1 (corresponding to the first eigenvector) has more or less equal scores for all variables; In other words, it is a sort of average and simply suggests how much fish fishermen catch.

PC2 separates feshermen into two groups: those that catch more Bluegill & Black crappie vs. those that catch more Small & Large mouth bass.

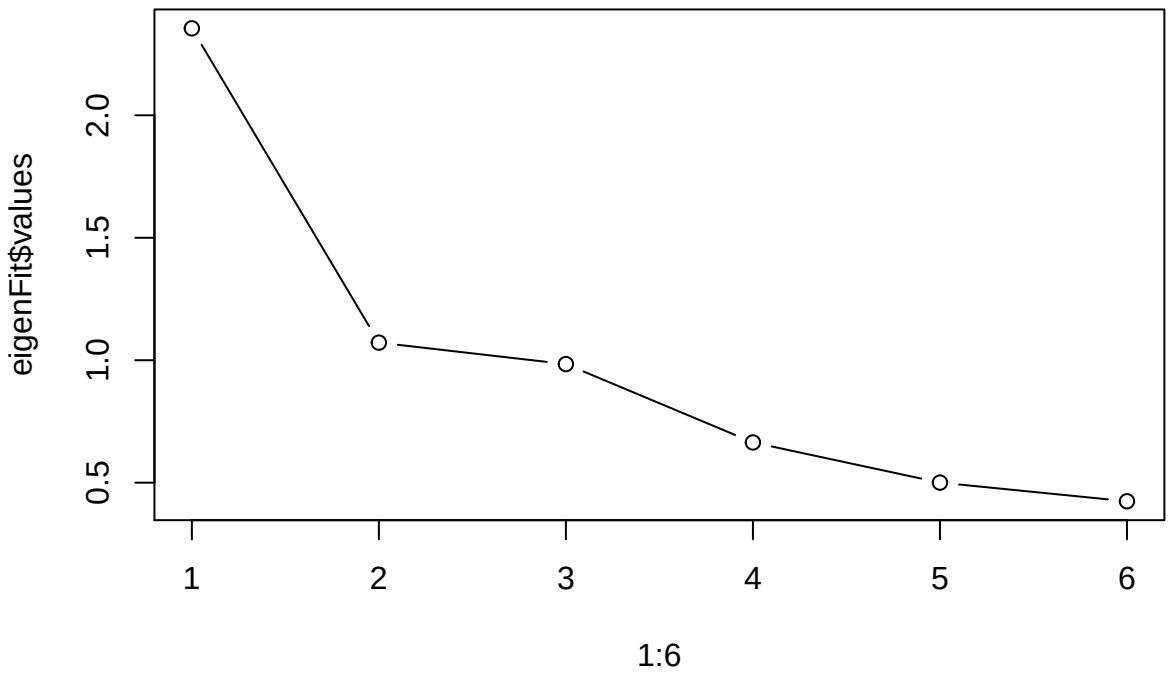
c) PCA on entire correlation matrix R

```
eigenFit = eigen(R)
eigenFit
```

```
## eigen() decomposition
## $values
## [1] 2.3549250 1.0718567 0.9842486 0.6643858 0.5003897 0.4241942
##
## $vectors
##            [,1]        [,2]        [,3]        [,4]       [,5]       [,6]
## [1,] -0.4753543  0.02188910  0.47987482  0.04564932  0.3578888 -0.6425051
## [2,] -0.4719328 -0.01924243  0.20906455  0.70296351 -0.1771358  0.4556533
## [3,] -0.3931540 -0.56061775 -0.26440849 -0.17551515 -0.5973982 -0.2713733
## [4,] -0.4963538 -0.07723024  0.03226116 -0.60426514  0.3238962  0.5260914
## [5,]  0.2563177 -0.80502150  0.01294351  0.21817123  0.4823355  0.0768220
## [6,] -0.2910014  0.17559671 -0.80925398  0.24539324  0.3822272 -0.1524794
```

Creating a scree plot

```
plot(1:6, eigenFit$values, type = 'b')
```



d)

```
sum(eigenFit$values[1:2]) / sum(eigenFit$values)
```

```
## [1] 0.5711303
```

We observe that 57.11% of the variation is accounted for by the first two PCs in (c)

e) Interpreting first loading vector of part (c)

```
rownames(eigenFit$vectors) <- fishNames
colnames(eigenFit$vectors) <- fishNames
eigenFit$vectors
```

```
##                   Bluegill Black crappie Smallmouth bass Largemouth bass
## Bluegill        -0.4753543    0.02188910      0.47987482      0.04564932
## Black crappie   -0.4719328   -0.01924243      0.20906455      0.70296351
## Smallmouth bass -0.3931540   -0.56061775     -0.26440849     -0.17551515
## Largemouth bass -0.4963538   -0.07723024      0.03226116     -0.60426514
## Walleye          0.2563177   -0.80502150      0.01294351      0.21817123
## Northern pike   -0.2910014    0.17559671     -0.80925398      0.24539324
##                     Walleye Northern pike
## Bluegill          0.3578888    -0.6425051
## Black crappie    -0.1771358     0.4556533
## Smallmouth bass  -0.5973982    -0.2713733
## Largemouth bass   0.3238962     0.5260914
## Walleye           0.4823355     0.0768220
## Northern pike     0.3822272    -0.1524794
```

The first loading vector clearly separates all the fishermen into those that catch Walleye vs. those that catch all the other types of fish. The problem states that Walleye is the most popular fish to eat. Perhaps this means there are two kinds of fishermen - those who fish for the fun of it and don't care what they catch and those who are specifically fishing to catch the tastiest fish!