MATH 6350: Statistical Learning and Data Mining

Homework 2

**Contributions of Co-Authors**

Stephanie Dinh ([sdinh@central.uh.edu](mailto:sdinh@central.uh.edu)): Wrote interpretation for 50% of the report.

Ying-Yu Huang ([yingyu010365@gmail.com](mailto:yingyu010365@gmail.com)): Wrote interpretation for 50% of the report.

Patricia Sieng ([patricia.sieng@yahoo.com](mailto:patricia.sieng@yahoo.com)): Wrote the code for all parts of the report.

[**Preliminary Treatment of the Data Set**](#p9ou0eh6n3st)

**Respective Sizes of CL1, CL2, CL3, and Data**

n1 = 4262

n2 = 4768

n3 = 4805

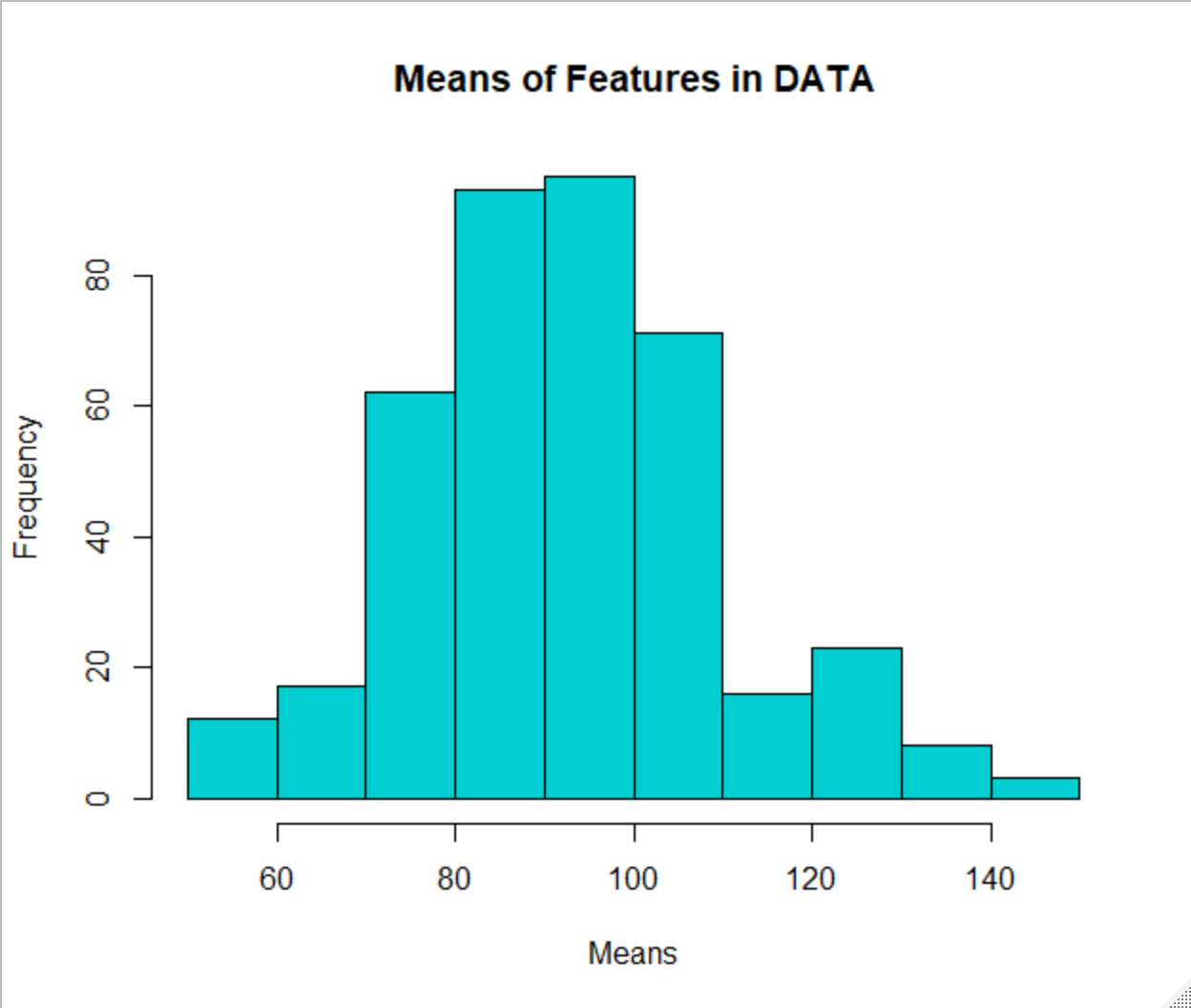
N = 13835

[**Part 0**](#p9ou0eh6n3st)

Click on the links below to access the computed means and standard deviations of the 400 features in the DATA data set

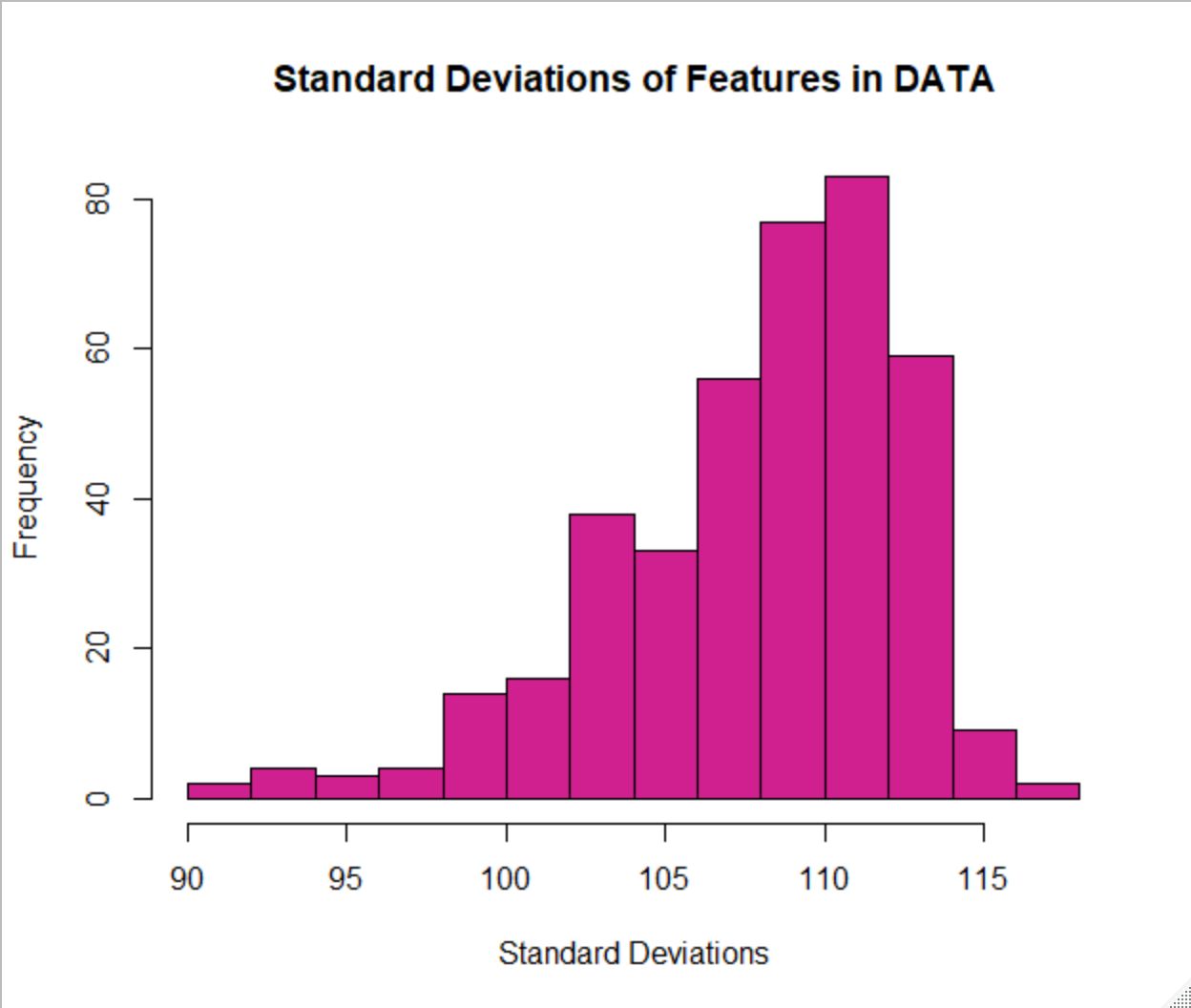
[Means](https://docs.google.com/spreadsheets/d/10d8bnuBToU2gU850iO1x-iSUGrQSfINXDxY3iiK8O-Q/edit?usp=sharing)

[Standard deviations](https://docs.google.com/spreadsheets/d/1jmWmKXdcED3kju7WeaSpgW6Jd5KqlGjpb_oryH43n0s/edit?usp=sharing)

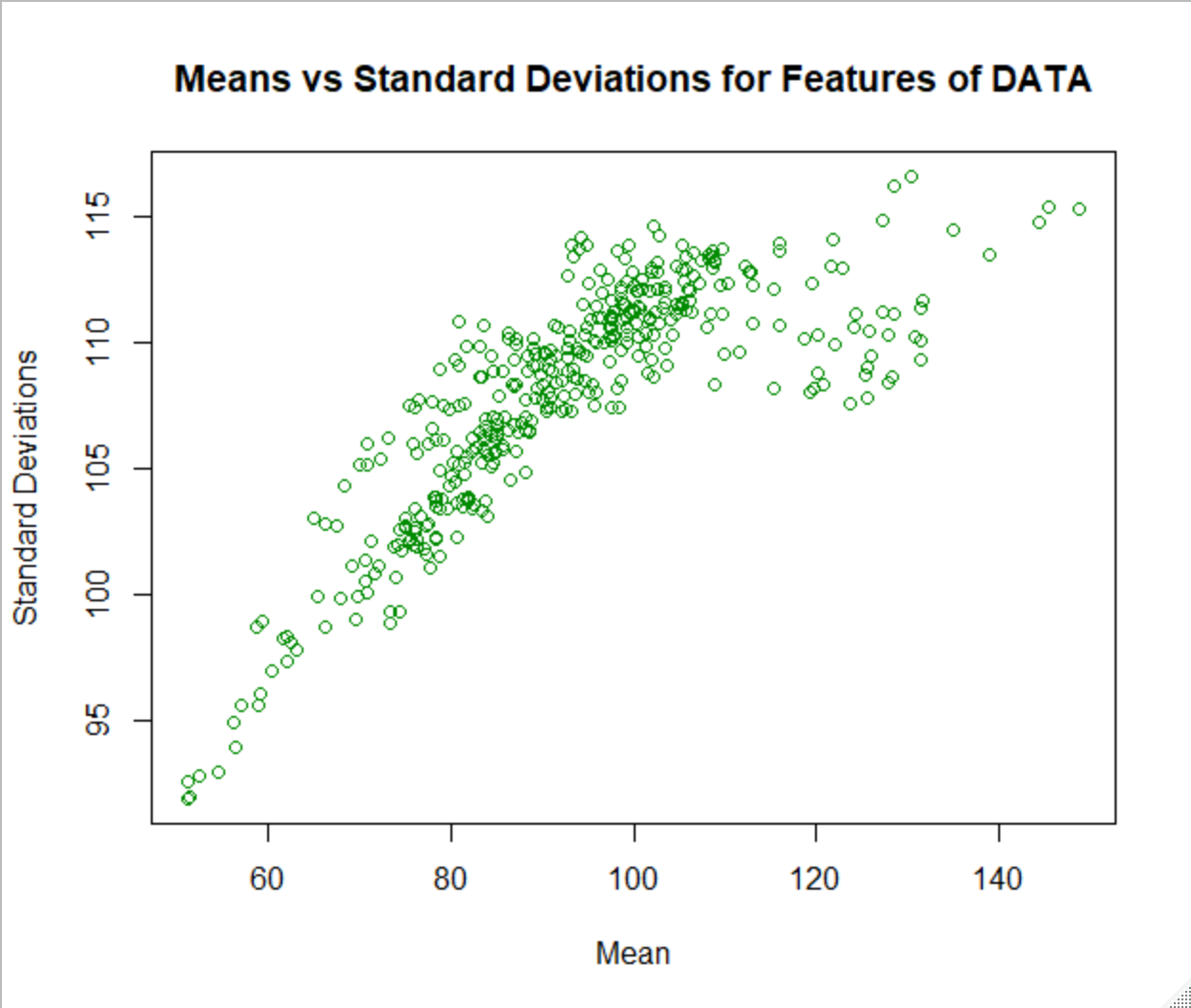


The range of means is approximately between 50 to 150, and its distribution appears to be

fairly symmetric and unimodal with few outliers.

****

The range of standard deviations is approximately in between 90 and 120, and it's distribution appears to be left-skewed with a few outliers, indicating that most of the standard deviations fall in between 108 and 115.

****

There is a strong, increasing trend between the mean and standard deviations of the features of Data. This indicates that the mean and standard deviation are closely correlated to one another and could be used to predict one another. In addition, the median of the mean is 91.92, indicating that most points are clustered around a mean of 91 with high standard deviations between 108 and 115.

[**Part 1 Results**](#5ysde581mght)

[1.1) Correlation matrix COR of the 400 random variables Y1,..., Y400](#citaf7masydl)

Click the link below to access the correlation matrix COR.

[COR](https://docs.google.com/spreadsheets/d/1PS3AhJ5owsahR5fOmLuFMjvj8ymigXhBh70C5ibKAyo/edit?usp=sharing)

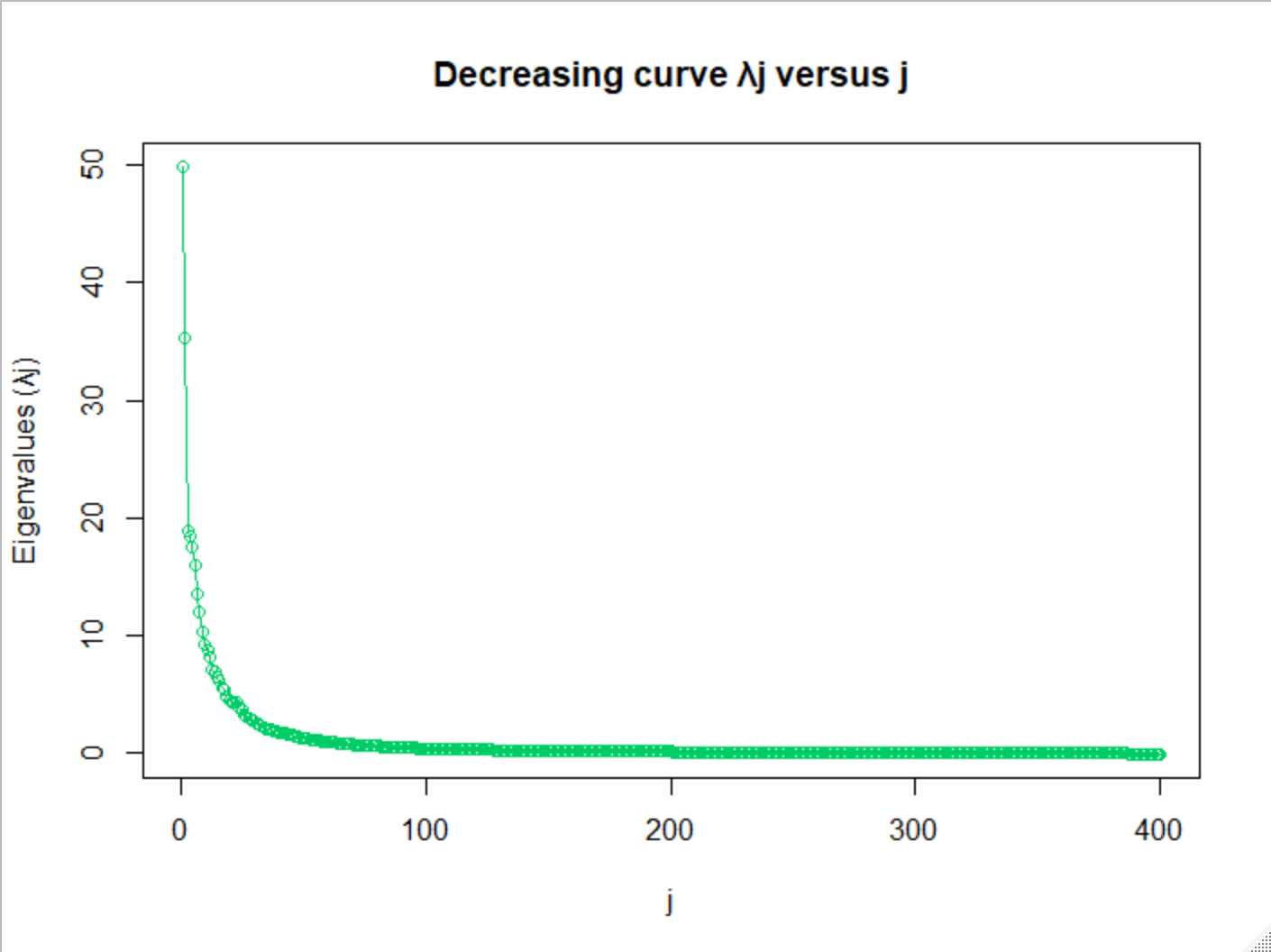
[1.2) Eigenvalues λ1 > λ2 > ... > λ400 > 0 , and Eigenvectors v1, v2, ..., v400 for matrix Cor](#wxpf5xgj6foy)

Click the links below to access the eigenvalues and eigenvectors.

[Eigenvalues](https://docs.google.com/spreadsheets/d/1uA5ae21Stfwzm6s-bAjVgFqT9Zs2KoWqq-2ON-PLTHM/edit?usp=sharing)

[Eigenvectors](https://docs.google.com/spreadsheets/d/1URAG3bIZ52AMm8Y3T7Nol36uzBDOAUzCA_A-NNMPfXM/edit?usp=sharing)

[1.3) The decreasing curve λj versus j for j=1 , 2, ..., 400](#aysftlj4m1du)

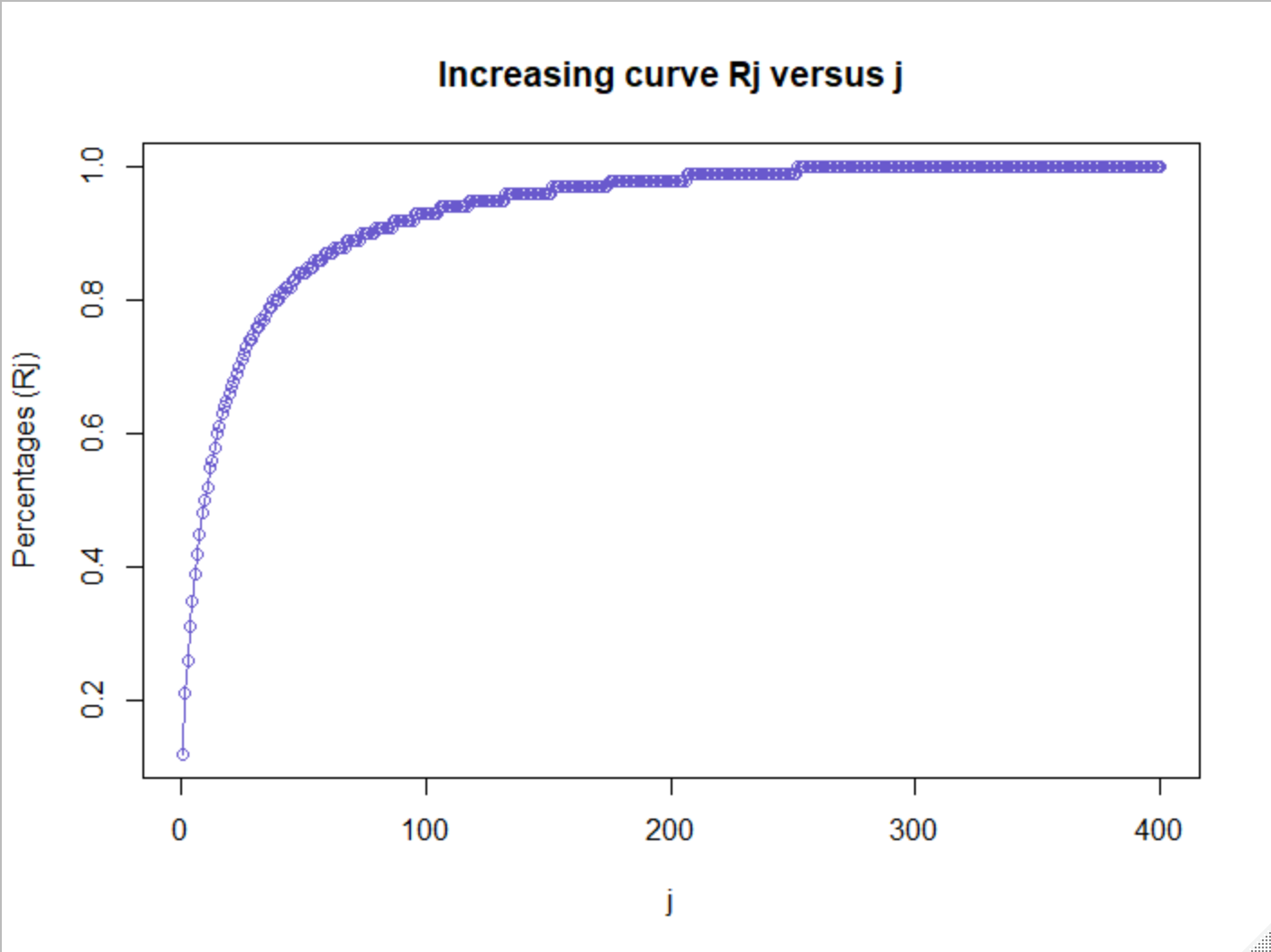


The decreasing curve shown above proves the successive decrease in the value of the eigenvalue as j increases. As j increases, λj decreases to the point of being near, but always greater than, near zero.

[1.4) The successive percentages Rj given by Rj = (λ1 + λ2 + ... + λj)/400 for j =1, 2, ... , 400](#r3tdn1kgb77u)

Click the link below to access the percentages Rj

[Rj](https://docs.google.com/spreadsheets/d/1cm0K9sqlD_bxtLsu-y6JOZREg4NabdBbnKCVpUsdH3c/edit?usp=sharing)

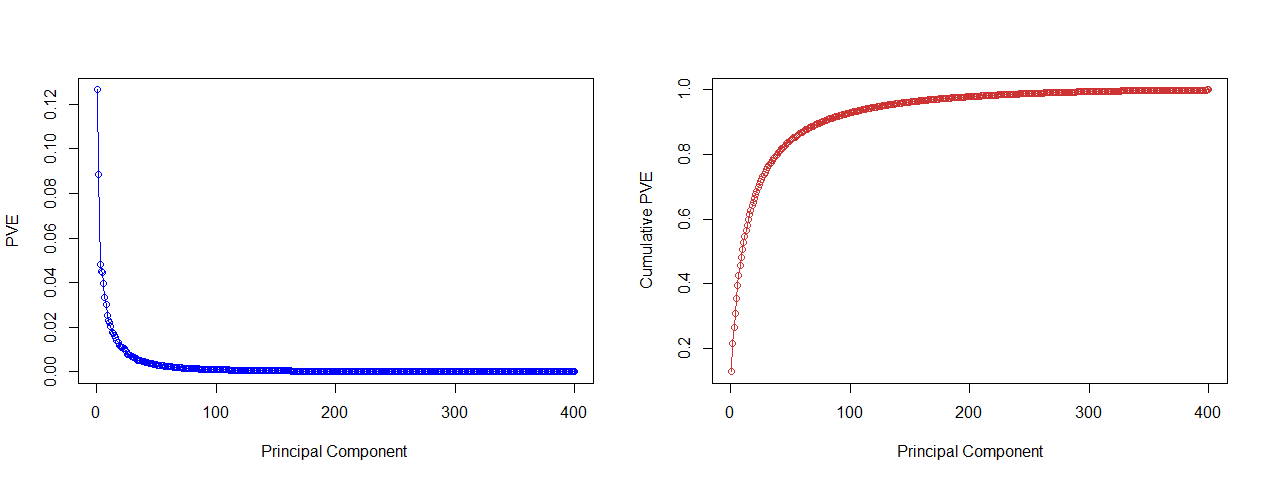
[1.5) The increasing curve Rj versus j for j=1 , 2, ..., 400](#j8ib9nbs6lr1) 

The above graph shows the successive increase in percentage Rj as j increases. In other words, the graph displays the percentage of explained common variance when the corresponding eigenvalue is divided by the total common variance j. Of course, the percentage increases when the number of eigenvalues taken into account increases.

The smallest integer r such that Rr >90% is 77 (R77 = 90.1%).

[1.6) The relationship between these computations and the PCA analysis of the set DATA](#tw70b3wk832h)

Performing PCA analysis on just the set DATA gives us very similar results to that of the previous questions.



In this case scaling the data does not make a big impact as the variables are already on the same scale.

**Correlation** indicates that there is redundancy in the data. To alleviate this redundancy, PCA can be used to reduce the original variables into a smaller number of new variables (i.e. principal components) explaining most of the variance in the original variables.

The diagonal elements are the variances of the different variables. **Large diagonal values correspond to a strong signal**.

The off-diagonal values are the covariances between variables. They reflect distortions in the data (noise, redundancy, …). **Large off-diagonal values correspond to high distortions in our data**.

The aim of PCA is to minimize this distortion and to summarize the essential information in the data.

**Eigenvalues**: The numbers on the diagonal of the diagonalized covariance matrix are called the eigenvalues of the covariance matrix. Large eigenvalues correspond to large variances. **The first (or in other words, the highest) eigenvalues correspond to the first principal components.**

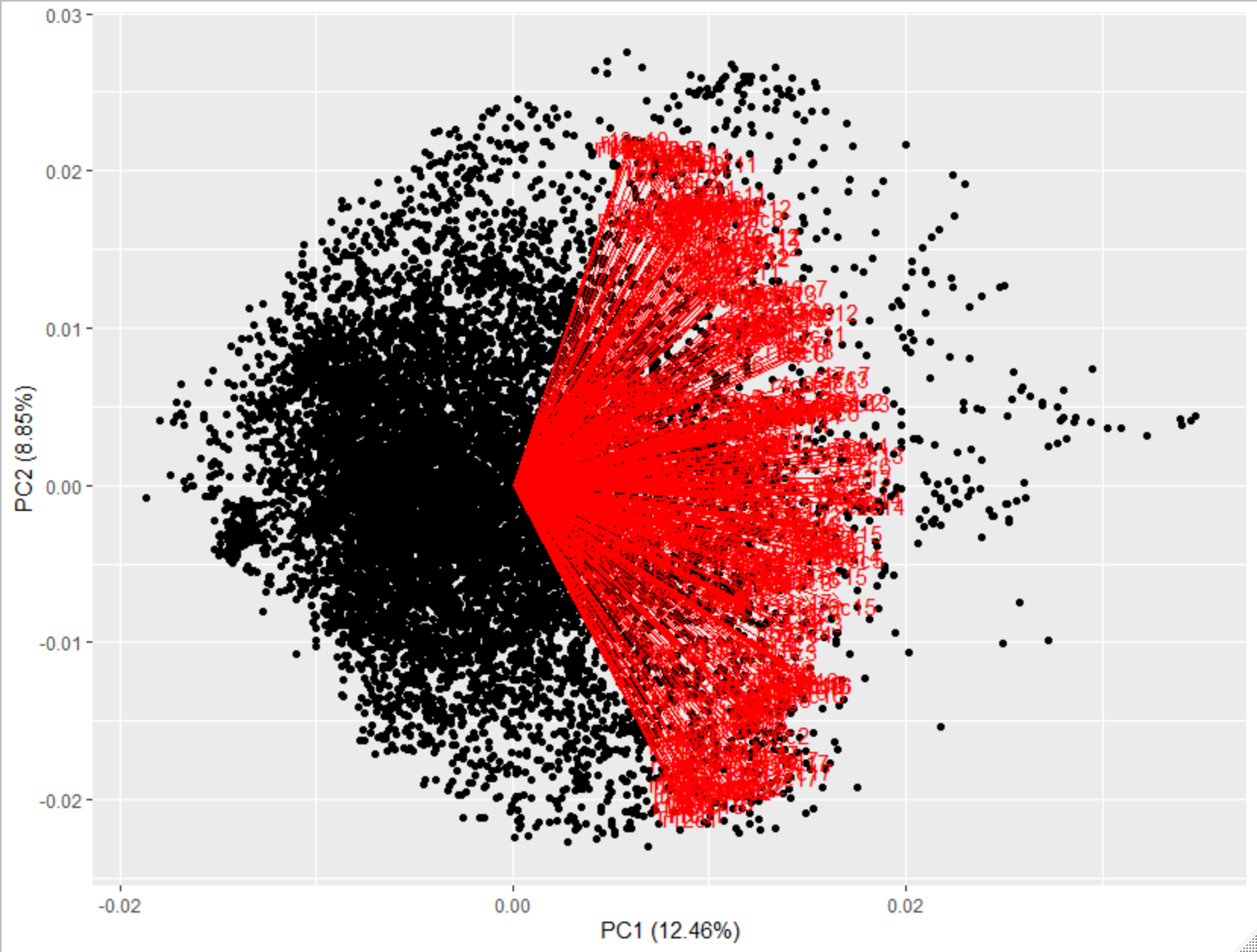
**Eigenvectors**: The directions of the new rotated axes are called the eigenvectors of the covariance matrix.

The first principal components of the data are the first directions explaining maximum variances. This is equivalent to the first eigenvectors of the covariance matrix.

First, we have 400 features. By calculating, we get R=77. It means we use the first 77 eigenvectors as the base to project DATA to display 90.1% of the variety of the data. Then, the features is reduced from 400 dimension to 77 dimension.

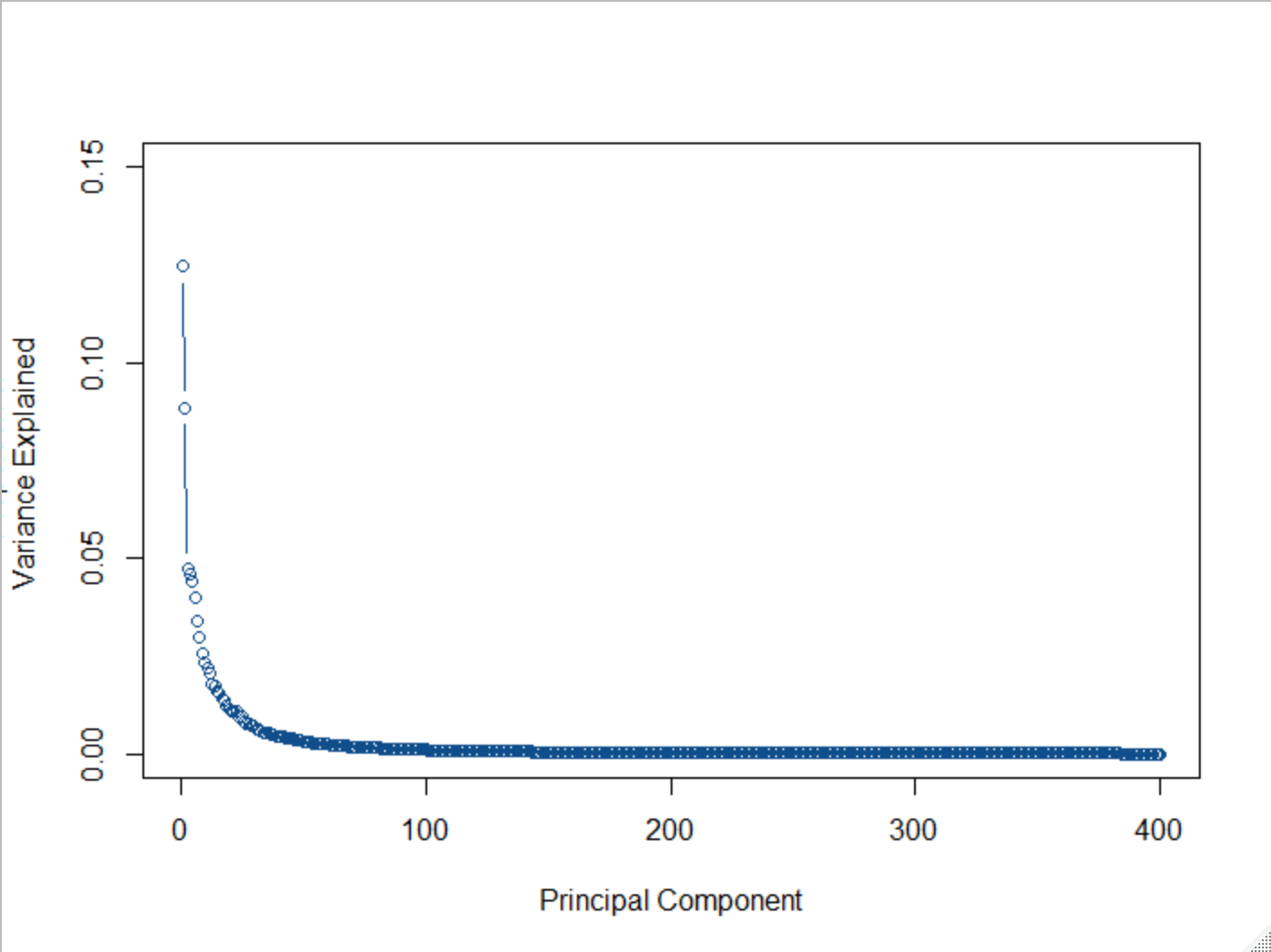
[1.7) PCA analysis of the rescaled data matrix SDATA](#buutgtp3q960)

The prcomp() function creates our PCA object and performs PCA analysis by taking in the input which is the 400 features of SDATA. Center and scale are set to FALSE since we have previously centered and scaled DATA to become the SDATA matrix.

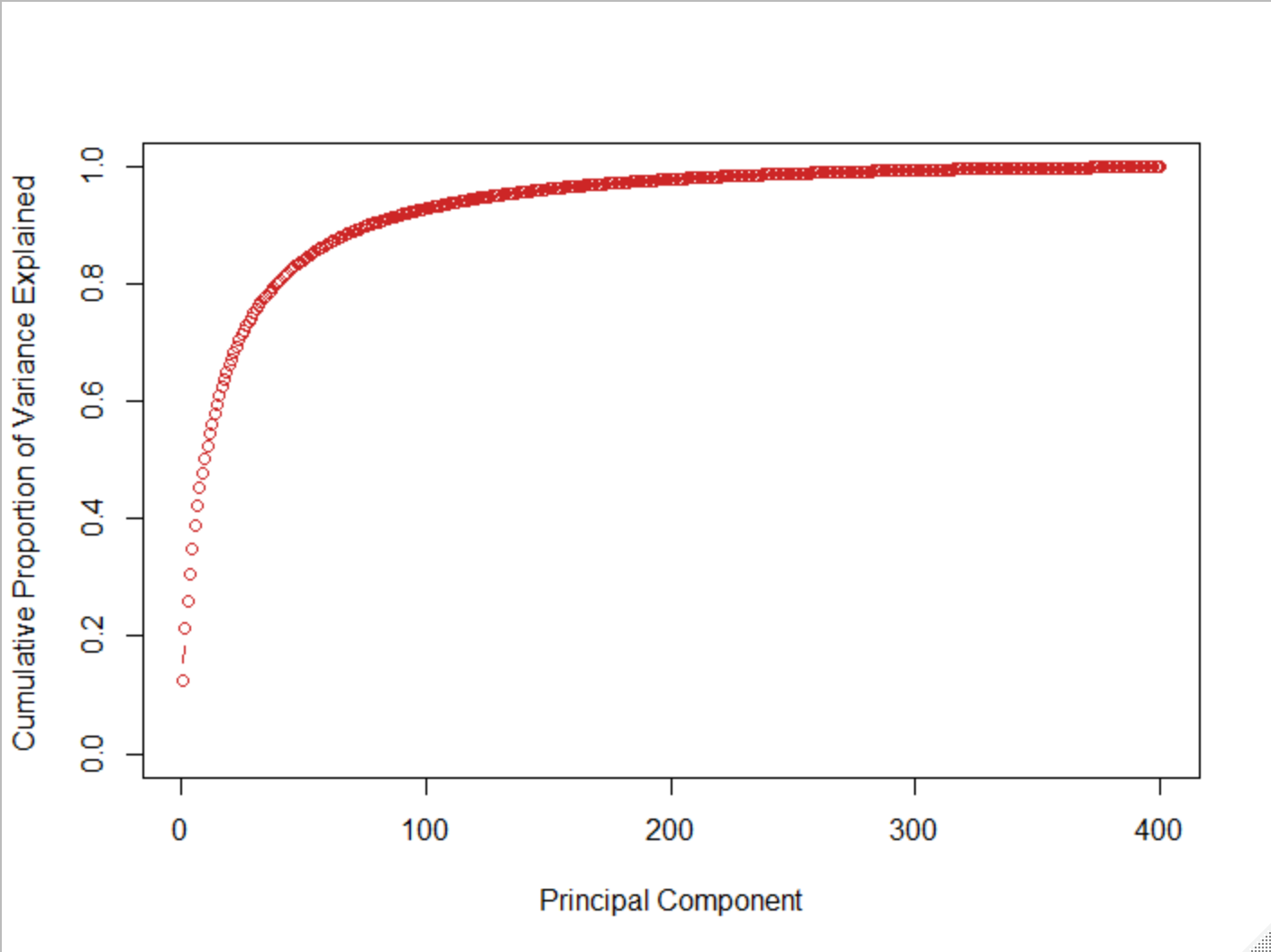


We installed a special library from GitHub, ggfortify (an enhancement library for ggplot2), in order to plot the first two principal components as shown above. We used the function autoplot() from the ggplot2 library which takes our PCA object as input. The red arrows indicate the first two principal component loading vectors. The above graph shows that the first principal component (PC1) explains 12.46% of the variance in the data while the second principal component (PC2) explains 8.85% of the variance in the data. We see that the first loading vector places approximately equal weight on about half of the 400 features of SDATA while the second loading vector places approximately equal weight on the other half of those 400 features. Thus, half of the features of SDATA are correlated to one another while the other half are correlated likewise.

Using the variance as input (which is obtained by squaring the standard deviation), we can calculate the proportion of variance explained by each principal component by dividing the variance explained by each principal component by the total variance explained by all 400 principal component (pca.pve=pca.var/sum(pca.var)).



The plot above displays the proportion of variance explained by each of the 400 components in the SDATA data set. The plot() function takes pca.pve (that we have computed and explained above) as input.



Likewise, we can also plot the cumulative PVE (shown above) to display the cumulative proportion of variance explained by the 400 components in the SDATA data set. The plot() function takes the cumulative sum of of pca.pve which is computed by the function cumsum(). This graph and the previous graph together explain 100% of the variance in the data.

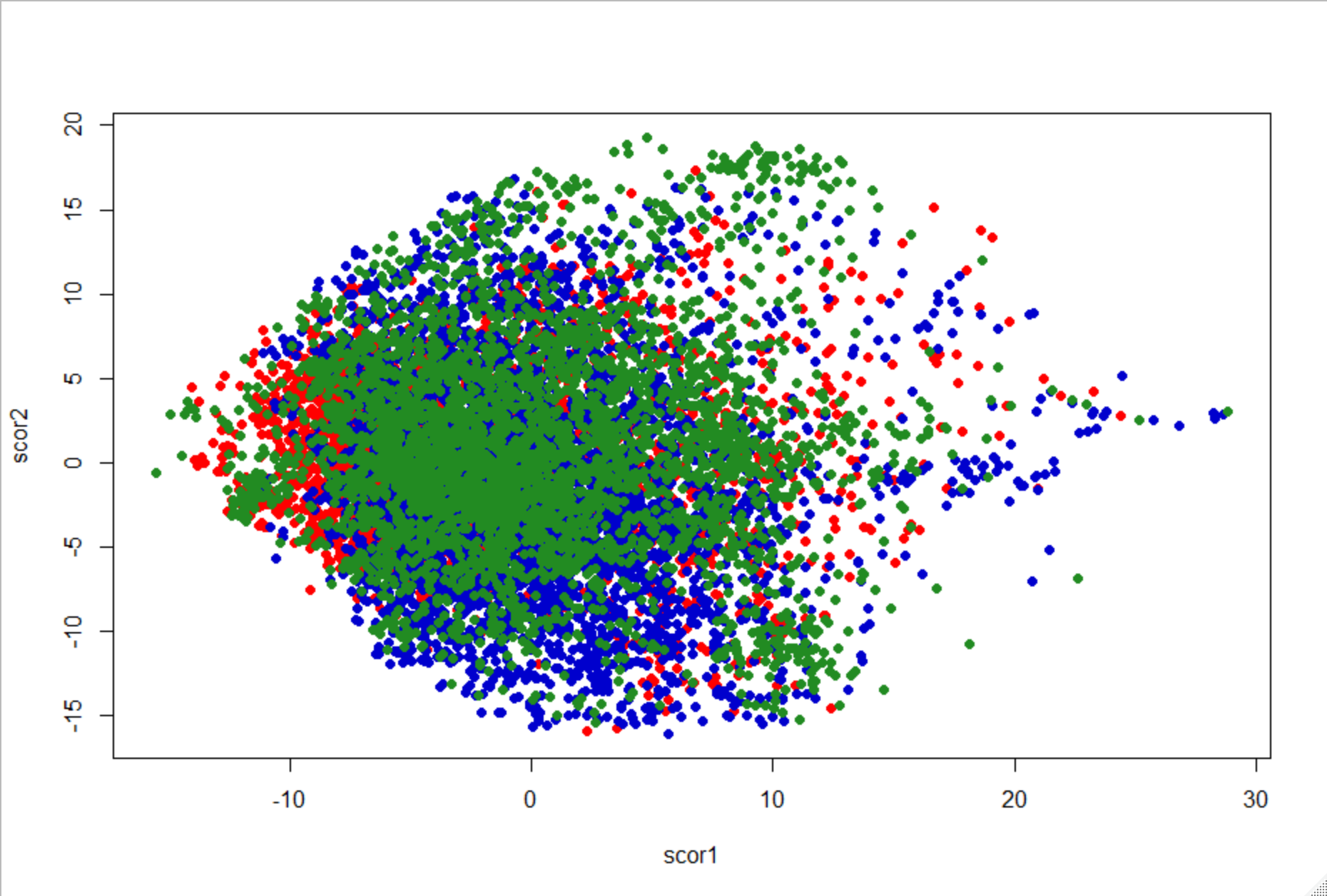
[1.8) The geometric relationship between Ei, Ui, and Wi](#kugkrmgson68)

Ei is the column vector of principal component i scores. It is formed by the linear combination (scalar product) of the values in the columns from the transposed SDATA matrix and the p eigenvectors generated by PCA analysis of the data.

Wi is the projection of Ei on the 2-dimensional space (x, y plane) by the first two principal component scores, scor1(i) and scor2(i).

Ui is the projection of Ei on the 3-dimensional space (x, y, z plane) by the first three principal component scores, scor1(i), scor2(i), and scor3(i).

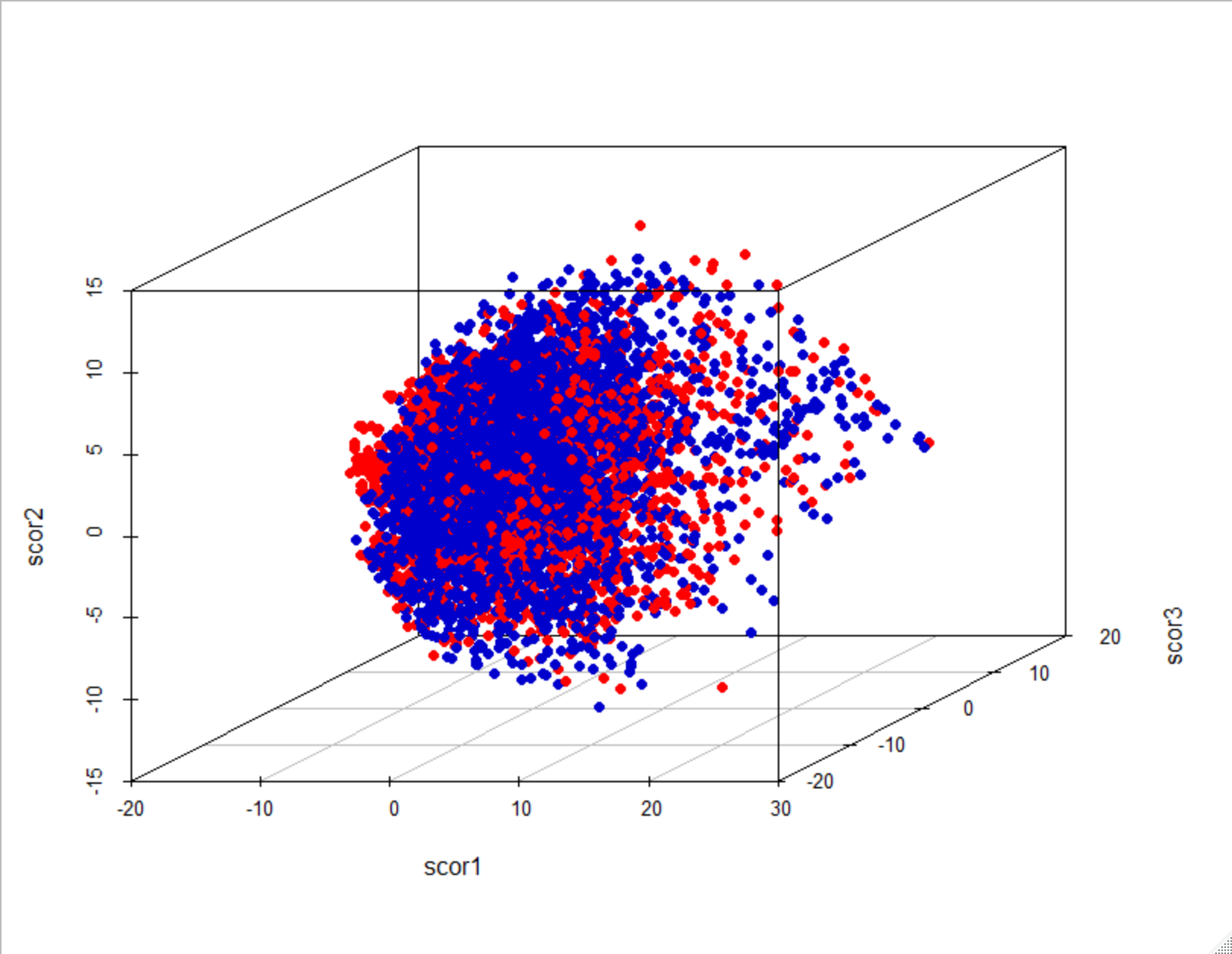
[1.9) 2 dimensional scatterplot of all the Wi , i= 1,2,...,N](#pirt42sisc3g)

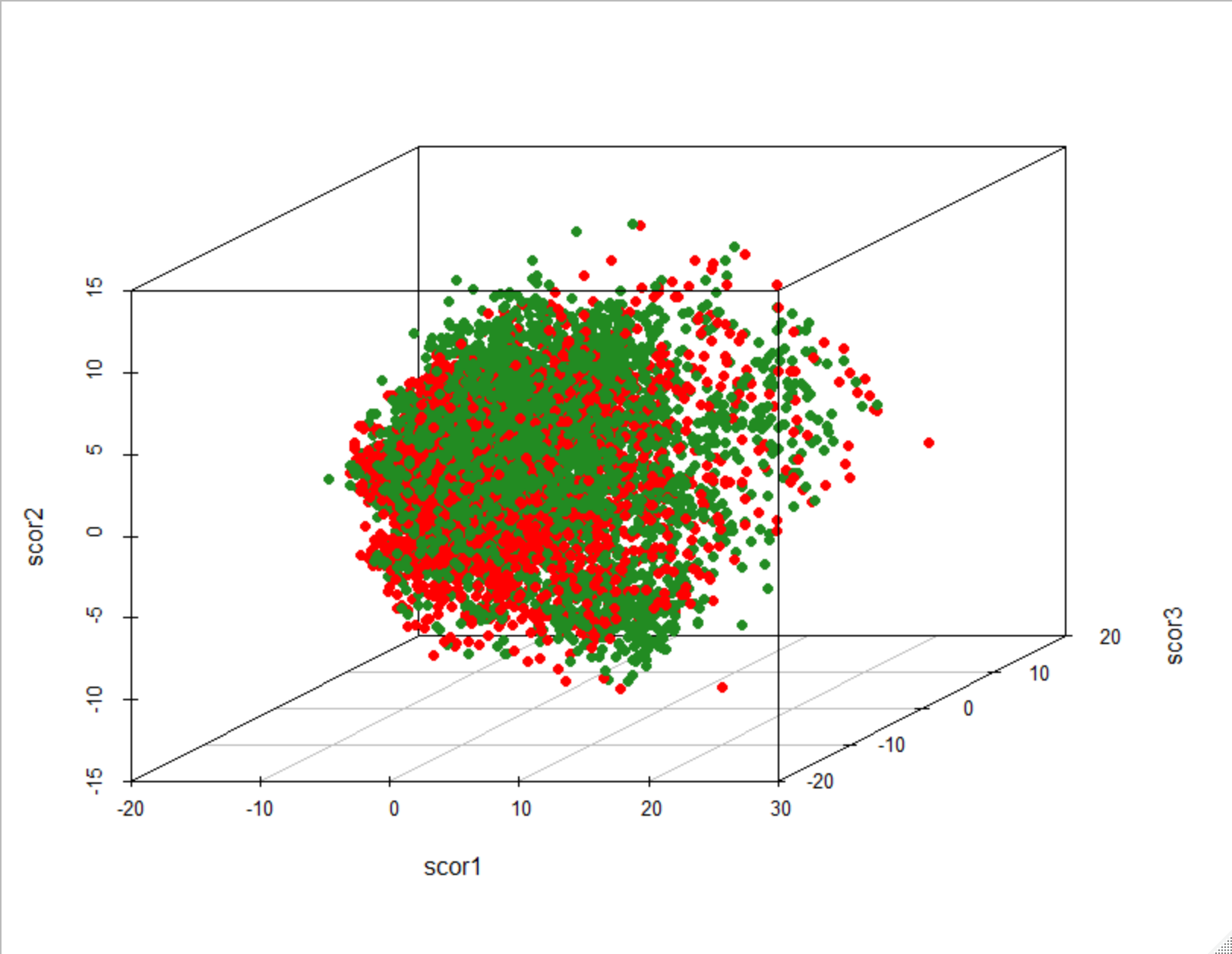
****

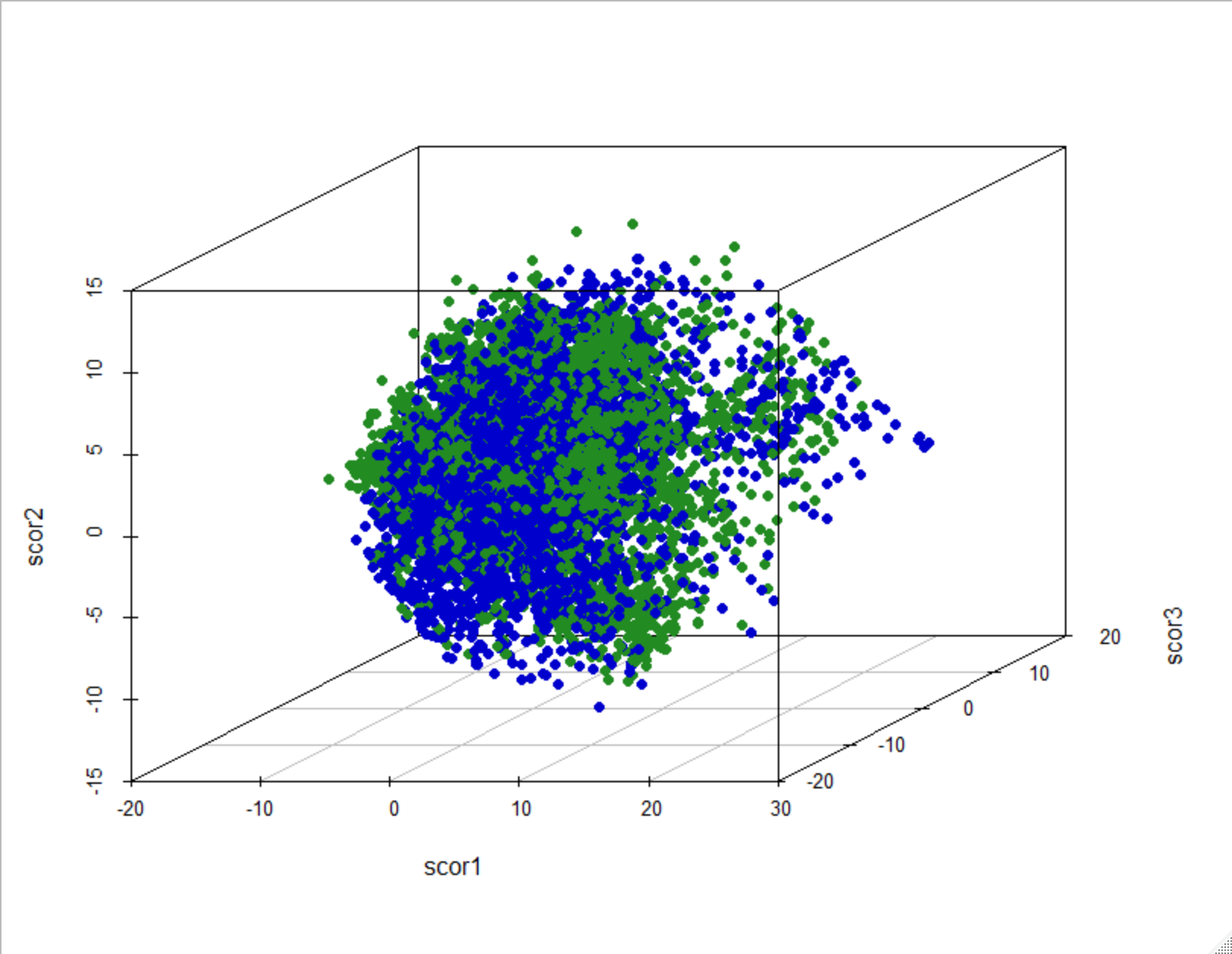
The above graph displays no clear distinction or any visual indication of clustering between the three classes. Although we can not rule out that there is no significant distinction at all between the three classes or that performing PCA on our data set does not reveal any meaningful information, the graph above potentially implies that the largest possible source of variation is similar in all three classes. We will lose too much information when we project them onto a 2-dimension space.

[1.10) 3 dimensional scatterplot of all the Ui , i= 1,2,...,N](#9gi5vok801pa)

**Note: Refer to color key in 1.9**

****

****

****

The graphs above display no clear distinction or any visual indication of clustering between each of the two classes. Plotting with an additional third component score (scor3) for each pair of classes does not improve nor demonstrate any visual proof that there is a significant amount of variance between the three classes. As mentioned previously, this may indicate that the largest possible source of variance is similar for all three classes.

[**Part 2 Results**](#knxgzzad04xx)

**Notation**

k = The number of "nearest" neighbors that kNN considers in the model

per(k) = The percentage of correct classifications using k

[2.1) kNN algorithm on SDATA for k = 15](#qtj23yn1fdh2)

per(15) = 0.7622

k = 15 correctly classifies 76.22% of the observations into one of the three classes.

[2.2) kNN algorithm on SDATA for k = 5, 10, 15, 20, 30, 40, 50, 100, 200](#e2r8mwodamho)

per(5) = 0.7965

k = 5 correctly classifies 79.65% of the observations into one of the three classes.

per(10) = 0.7763

k = 10 correctly classifies 77.63% of the observations into one of the three classes.

Refer to section 2.1 for k = 15 and per(15)

per(20) = 0.7467

k = 20 correctly classifies 74.67% of the observations into one of the three classes.

per(30) = 0.7282

k = 30 correctly classifies 72.82% of the observations into one of the three classes.

per(40) = 0.7091

k = 40 correctly classifies 70.91% of the observations into one of the three classes.

per(50) = 0.6885

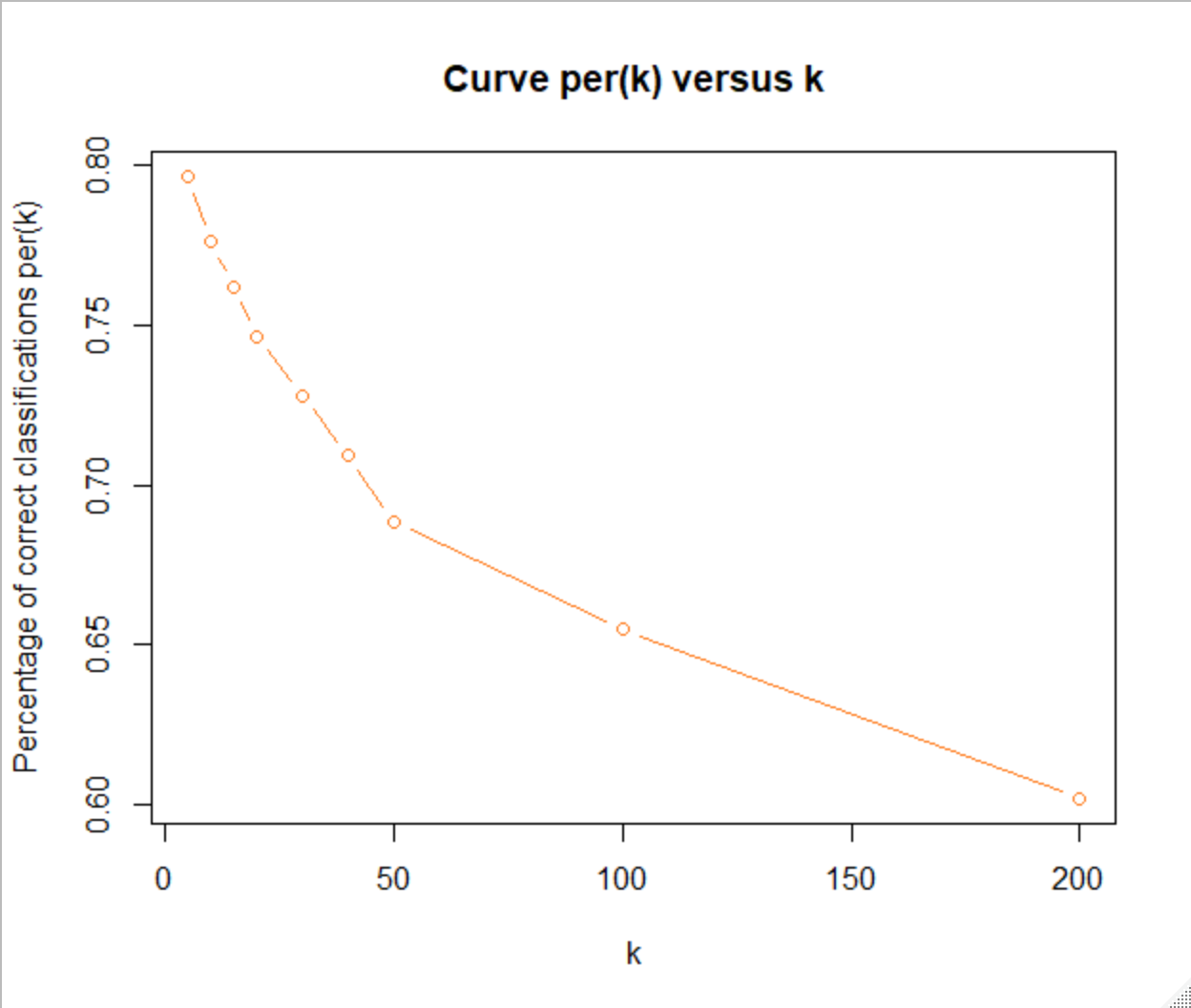
k = 50 correctly classifies 68.85% of the observations into one of the three classes.

per(100) = 0.6549

k = 100 correctly classifies 65.49% of the observations into one of the three classes.

per(200) = 0.6021

k = 200 correctly classifies 60.21% of the observations into one of the three classes.



Increasing k beyond 5 provides no further improvements in predicting the correct classifications. We identified that the best range is for k would be between 1 and 5. [1<k<5]

[2.3) Selecting a "best" value k\* for the integer k](#w80bjftdvjfe)

per(1) = 0.8605

k = 1 correctly classifies 86.05% of the observations into one of the three classes.

per(2) = 0.8218

k = 2 correctly classifies 82.18% of the observations into one of the three classes.

per(3) = 0.8179

k=3 correctly classifies 81.79% of the observations into one of the three classes.

per(4) = 0.8066

k = 4 correctly classifies 80.66% of the observations into one of the three classes.

We conclude that the "best" value for k\* is k = 1. Using k = 1, we obtained an 86.05% success rate in predicting the correct classifications.

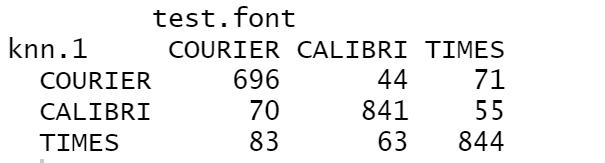
[2.4) 3x3 confusion matrix for kNN classification using the "best" k= k\*](#w80bjftdvjfe)

**Side Note**

COURIER refers to class CL1

CALIBRI refers to class CL2

TIMES refers to class CL3



The 400 features of the standardized data matrix SDATA were used to predict font type which is the unique identifier for each of the 3 classes, CL1, CL2, and CL3. We first randomly rearrange the rows of SDATA and then we split the observations into the training and test set using an 80:20 ratio.

Using k = 1, out of 2767 observations, 86.05% ((696+841+844)/2767) observations were correctly classified into their respective classes.

**#** **Code for Preliminary Treatment of the Data Set**

# Discard unused columns

start\_time <- Sys.time()

courier <- COURIER[,-c(2:3, 6:12)]

calibri <- CALIBRI[,-c(2:3, 6:12)]

times <- TIMES[,-c(2:3, 6:12)]

# Keep only rows where strength and italic is NORMAL

CL1 <- courier[(courier$strength==0.4) & (courier$italic==0),]

CL2 <- calibri[calibri$strength==0.4 & calibri$italic==0,]

CL3 <- times[times$strength==0.4 & times$italic==0,]

# Sizes of CL1, CL2, CL3

n1 <- nrow(CL1)

n1

n2 <- nrow(CL2)

n2

n3 <- nrow(CL3)

n3

# Full dataset (union of CL1, CL2, and CL3) and its size

union1 <- rbind(CL1, CL2)

DATA <- rbind(union1, CL3)

N <- nrow(DATA)

N

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 1.52 secs

**#** **Code for Part 0**

# Calculates the mean and standard deviations of the features (stores them in vectors of size 400)

# Also creates SDATA matrix by standardizing the features matrix DATA

start\_time <- Sys.time()

mean <- 1:400

sd <- 1:400

SDATA <- DATA[,1:3]

for(j in 1:400)

{

feature <- DATA[,j+3]

mean[j] <- mean(feature)

sd[j] <- sd(feature)

SDATA <- cbind(SDATA, (DATA[,j+3, drop=FALSE]-mean[j])/sd[j])

}

# Histograms of mean and standard deviations

hist(mean,

main="Means of Features in DATA",

xlab="Means",

ylab="Frequency",

col="darkturquoise")

hist(sd,

main="Standard Deviations of Features in DATA",

xlab="Standard Deviations",

ylab="Frequency",

col="violetred")

# Scatterplot of means vs standard deviations

plot(mean, sd, main="Means vs Standard Deviations for Features of DATA",

xlab="Mean", ylab="Standard Deviations", col="green4")

median(mean) # 91.92

# Output mean and standard deviation into Excel files

write.csv(round(mean,2), file="mean.csv")

write.csv(round(sd,2), file="sd.csv")

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 2.77 secs

**#** **Code for Part 1**

# 1.1) Correlation Matrix COR

start\_time <- Sys.time()

COR <- cor(SDATA[,4:403])

# Output COR into Excel file

write.csv(round(COR,2), file = "correlation matrix.csv")

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 2.50 secs

# 1.2) Eigenvalues and eigenvectors for matrix COR

start\_time <- Sys.time()

eigen <- eigen(COR)

eigenvalues <- eigen$values

sum(eigenvalues) # Sum of eigenvalues is equal to 400

eigenvectors <- eigen$vectors

# Output eigenvalues and eigenvectors into Excel files

# Each eigenvalue is a nonzero value greater than 0

write.csv(round(eigenvalues,2), file = "eigenvalues.csv")

write.csv(round(eigenvectors,2), file = "eigenvectors.csv")

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 0.65 secs

# 1.3) Plot the decreasing curve λj versus j for j=1 , 2, ..., 400

start\_time <- Sys.time()

plot(eigenvalues, type="o", col="springgreen3", ann="FALSE")

title(main="Decreasing curve λj versus j", xlab="j", ylab="Eigenvalues (λj)")

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 0.056 secs

# 1.4) The successive percentages Rj given by Rj = (λ1 + λ2 + ... + λj)/400 for j =1, 2, ... , 400

start\_time <- Sys.time()

Rj <- 1:400

sum <- 0

for(i in 1:400)

{

sum <- sum + eigenvalues[i]

percentage <- sum/400

Rj[i] = percentage

}

# Output percentages into Excel file

write.csv(round(Rj,2), file = "percentage.csv")

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 0.40 secs

# 1.5) The increasing curve Rj versus j for j=1 , 2, ..., 400

start\_time <- Sys.time()

plot(Rj, type="o", col="slateblue3", ann="FALSE")

title(main="Increasing curve Rj versus j", xlab="j", ylab="Percentages (Rj)")

# The smallest integer "r" such that Rr >90%

smallest <- 0

for(i in 1:400)

{

if(Rj[i] > 0.90)

{

smallest <- i

break

}

}

smallest # Smallest integer of "r" is 77

Rj[smallest] # 0.901 (90.1%)

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 0.088 secs

# 1.6) The relationship between these computations and the PCA analysis of the set DATA

# Perform PCA on DATA

start\_time <- Sys.time()

pDATA=DATA[,4:403]

pr.out=prcomp(pDATA , scale=FALSE)

pr.var=pr.out$sdev^2

summary(pr.out)

pve =pr.out$sdev ^2/sum(pr.out$sdev^2)

par(mfrow=c(1,2))

plot(pve , type="o", ylab="PVE", xlab="Principal Component ", col="blue")

plot(cumsum(pve), type="o", ylab="Cumulative PVE", xlab=" Principal Component ", col="brown3")

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 11.69 secs

# 1.7) Implement PCA Analysis

start\_time <- Sys.time()

pca <- prcomp(SDATA[,c(4:403)], scale=FALSE, center=FALSE)

# Install library ggfortify to plot PCA

devtools::install\_github("sinhrks/ggfortify") # Select option 1

library(ggfortify)

ggplot2::autoplot(pca, label = FALSE, loadings.label = TRUE)

pca.sdev <- pca$sdev

pca.var <- pca.sdev^2

pca.pve <- pca.var/sum(pca.var)

# Plots of pve and cumulative pve

plot(pca.pve, xlab=" Principal Component ", ylab=" Proportion of

Variance Explained ", ylim=c(0,0.15), type="b", col="dodgerblue4")

plot(cumsum(pca.pve), xlab=" Principal Component ", ylab ="

Cumulative Proportion of Variance Explained ", ylim=c(0,1),

type="b", col="firebrick3")

end\_time <- Sys.time()

end\_time - start\_time # 19.0 secs

# 1.8) The first three "scores" of example "i" from the transpose of SDATA

# Transpose the SDATA matrix and create v1, v2, and v3 from the eigenvectors

start\_time <- Sys.time()

TSDATA <- t(SDATA[,4:403])

v1 <- eigenvectors[,1]

v2 <- eigenvectors[,2]

v3 <- eigenvectors[,3]

# Calculate the first three scores

scor1 <- numeric(0)

scor2 <- numeric(0)

scor3 <- numeric(0)

for (i in 1:ncol(TSDATA))

{

scor1[i] = TSDATA[,i] %\*% v1

scor2[i] = TSDATA[,i] %\*% v2

scor3[i] = TSDATA[,i] %\*% v3

}

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 1.06 secs

# 1.9) 2 dimensional scatterplot of all the Wi , i= 1,2,...,N

start\_time <- Sys.time()

SDATA.font <- SDATA$font

colors <- c("red1", "mediumblue", "forestgreen")[SDATA.font]

plot(pca$x[,1:2], col=colors, pch=19, xlab ="scor1", ylab="scor2")

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 1.24 secs

# 1.10) 3 dimensional scatterplot of all the Ui , i= 1,2,...,N

# Function that hides desired color (used to exclude classes on plot)

# Credit: http://www.dataanalytics.org.uk

t\_col <- function(color, percent = 100) {

rgb.val <- col2rgb(color)

t.col <- rgb(rgb.val[1], rgb.val[2], rgb.val[3],

max = 255,

alpha = (100-percent)\*255/100)

invisible(t.col)

}

# Install and load package to create 3D scatterplot

start\_time <- Sys.time()

install.packages("scatterplot3d")

library("scatterplot3d")

colors <- c("red1", "mediumblue", t\_col("forestgreen"))[SDATA.font]

scatterplot3d(scor1, scor2, scor3, pch=16, color=colors, grid=TRUE, box=TRUE,

xlab="scor1", zlab="scor2", ylab="scor3")

colors <- c("red1", t\_col("mediumblue"), "forestgreen")[SDATA.font]

scatterplot3d(scor1, scor2, scor3, pch=16, color=colors, grid=TRUE, box=TRUE,

xlab="scor1", zlab="scor2", ylab="scor3")

colors <- c(t\_col("red1"), "mediumblue", "forestgreen")[SDATA.font]

scatterplot3d(scor1, scor2, scor3, pch=16, color=colors, grid=TRUE, box=TRUE,

xlab="scor1", zlab="scor2", ylab="scor3")

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 9.51 secs

**#** **Code for Part 2**

# 2.1) kNN algorithm on SDATA for k=15

# Randomly reorder rows of SDATA

start\_time <- Sys.time()

set.seed(1)

rows <- sample(nrow(SDATA))

SDATA.subset <- SDATA[rows,]

# Convert font to factor variable (response variable to predict classes)

SDATA.subset$font <- factor(SDATA.subset$font)

# Split training and test set using an 80:20 ratio

split <- floor(nrow(SDATA)\*.20)

test <- 1:split

train.font <- SDATA.subset$font[-test]

test.font <- SDATA.subset$font[test]

# Choose the 400 features to predict the classes

SDATA.subset <- SDATA.subset[,4:403]

train.SDATA <- SDATA.subset[-test,]

test.SDATA <- SDATA.subset[test,]

library(class)

# k=15 and per(15)

knn.15 <- knn(train.SDATA, test.SDATA, train.font, k=15)

table(knn.15, test.font)

per.15 <- round(((594+740+775)/split),4)

per.15

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 40.71 secs

# 2.2) kNN algorithm on SDATA for k =5, 10, 15, 20, 30, 40, 50, 100, 200

start\_time <- Sys.time()

# k=5 and per(5)

knn.5 <- knn(train.SDATA, test.SDATA, train.font, k=5)

table(knn.5, test.font)

per.5 <- round(((638+787+779)/split),4)

per.5

# k=10 and per(10)

knn.10 <- knn(train.SDATA, test.SDATA, train.font, k=10)

table(knn.10, test.font)

per.10 <- round(((614+760+774)/split),4)

per.10

# Refer to 2.1 for k=15 and per(15)

# k=20 and per(20)

knn.20 <- knn(train.SDATA, test.SDATA, train.font, k=20)

table(knn.20, test.font)

per.20 <- round(((572+718+776)/split),4)

per.20

# k=30 and per(30)

knn.30 <- knn(train.SDATA, test.SDATA, train.font, k=30)

table(knn.30, test.font)

per.30 <- round(((558+689+768)/split),4)

per.30

# k=40 and per(40)

knn.40 <- knn(train.SDATA, test.SDATA, train.font, k=40)

table(knn.40, test.font)

per.40 <- round(((535+655+772)/split),4)

per.40

# k=50 and per(50)

knn.50 <- knn(train.SDATA, test.SDATA, train.font, k=50)

table(knn.50, test.font)

per.50 <- round(((511+637+757)/split),4)

per.50

# k=100 and per(100)

knn.100 <- knn(train.SDATA, test.SDATA, train.font, k=100)

table(knn.100, test.font)

per.100 <- round(((505+540+767)/split),4)

per.100

# k=200 and per(100)

knn.200 <- knn(train.SDATA, test.SDATA, train.font, k=200)

table(knn.200, test.font)

per.200 <- round(((504+425+737)/split),4)

per.200

# Curve per(k) versus k

pers <- c(per.5, per.10, per.15, per.20, per.30, per.40, per.50,

per.100, per.200)

k <- c(5,10,15,20,30,40,50,100,200)

plot(k, pers, type="b", col="chocolate1", ann=FALSE)

title(main="Curve per(k) versus k", xlab="k", ylab="Percentage of correct classifications per(k)")

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 6.38 mins

# 2.3) Selecting a "best" value k\* for the integer k

start\_time <- Sys.time()

# k=1 and per(1)

knn.1 <- knn(train.SDATA, test.SDATA, train.font, k=1)

table(knn.1, test.font)

per.1 <- round(((696+841+844)/split),4)

per.1

# k=2 and per(2)

knn.2 <- knn(train.SDATA, test.SDATA, train.font, k=2)

table(knn.2, test.font)

per.2 <- round(((655+809+810)/split),4)

per.2

# k=3 and per(3)

knn.3 <- knn(train.SDATA, test.SDATA, train.font, k=3)

table(knn.3, test.font)

per.3 <- round(((655+806+802)/split),4)

per.3

# k=4 and per(4)

knn.4 <- knn(train.SDATA, test.SDATA, train.font, k=4)

table(knn.4, test.font)

per.4 <- round(((633+802+797)/split),4)

per.4

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 3.30 mins

# 2.4) 3x3 confusion matrix for kNN classification using the "best" k= k\*

start\_time <- Sys.time()

table(knn.1, test.font)

end\_time <- Sys.time()

end\_time - start\_time # Computation time: 0.018 secs