Applied Data Mining: Midterm

Anshul Doshi

Anshul Doshi (ad3222)
STAT UN3106 Section 001
Assignment: Midterm

Problem 1: Mixture of two Normals

i.) Set up the likelihood function

Our probability density function is defined by:

$$\begin{aligned} f(x) &= f(x;\mu_1,\sigma_1,\mu_2,\sigma_2) \\ &= \delta(x;\mu_1,\sigma_1) + (1-\delta)(x;\mu_2,\sigma_2) \\ &= \delta\frac{1}{\sqrt{2\pi\sigma_1^2}}e^{-\frac{1}{2\sigma_1^2}(x-\mu_1)^2} + (1-\delta)\frac{1}{\sqrt{2\pi\sigma_2^2}}e^{-\frac{1}{2\sigma_2^2}(x-\mu_2)^2} \end{aligned}$$

We now take the likelihood of this equation:

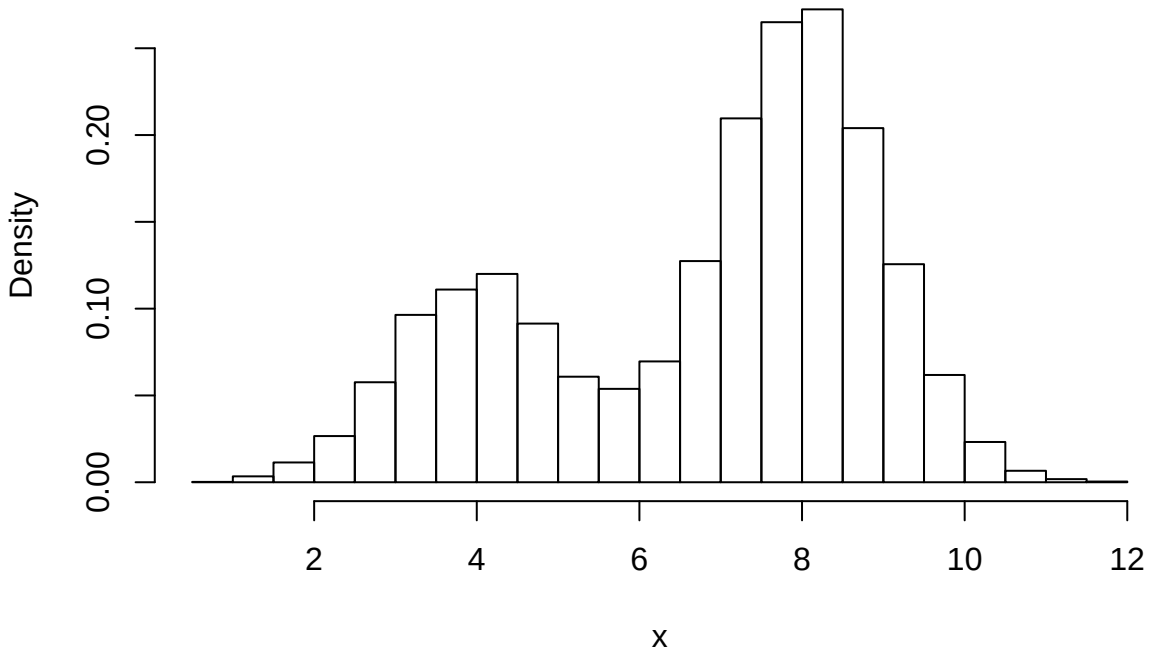
$$L(x;\mu_1,\sigma_1,\mu_2,\sigma_2) = \prod_{i=1}^{n}\left(\delta\frac{1}{\sqrt{2\pi\sigma_1^2}}e^{-\frac{1}{2\sigma_1^2}(x_i-\mu_1)^2} + (1-\delta)\frac{1}{\sqrt{2\pi\sigma_2^2}}e^{-\frac{1}{2\sigma_2^2}(x_i-\mu_2)^2}\right)$$

It is very difficult to simplify here so we will go directly to doing log-likelihood from the above likelihood equation. Note, the log of the products is the sum of the logs:

$$\ell(x;\mu_1,\sigma_1,\mu_2,\sigma_2) = \sum_{i=1}^{n} log\left(\delta\frac{1}{\sqrt{2\pi\sigma_1^2}}e^{-\frac{1}{2\sigma_1^2}(x_i-\mu_1)^2} + (1-\delta)\frac{1}{\sqrt{2\pi\sigma_2^2}}e^{-\frac{1}{2\sigma_2^2}(x_i-\mu_2)^2}\right)$$

ii.) Run the following code:

```
NormalMix <- read.csv("NormalMix.csv")[,-1]
hist(NormalMix,breaks=20,xlab="x",probability = T)
```



iii.) Define the neg-log-likelihood function and evaluate at point

$$(\mu_1, \sigma_1, \mu_2, \sigma_2, \delta) = (4, 2, 8, 2, .5)$$

```
neg.NormalMix.ll <- function(params, data) {
  n = nrow(data)
  mu_1 <- params[1]
  sigma_1 <- params[2]
  mu_2 <- params[3]
  sigma_2 <- params[4]
  delta <- params[5]

  return(-sum(log(delta*dnorm(data, mean=mu_1, sd=sigma_1, log=FALSE) +
                (1-delta)*dnorm(data, mean=mu_2, sd=sigma_2, log=FALSE))))

  # return(-1 * sum( log( delta*(2*pi*sigma_1^2)^(-.5)*exp( (-2*sigma_1^2)^-1*(data-mu_1)^2)
  #       + (1-delta)*(2*pi*sigma_2^2)^(-.5)*exp( (-2*sigma_2^2)^-1*(data-mu_2)^2)) ) )
}

my_params = c(4,2,8,2,.5)
neg.NormalMix.ll(my_params,data = NormalMix)
```

```
## [1] 22479.07
```

iv.) Using the nlm function to compute the maximum likelihood estimate

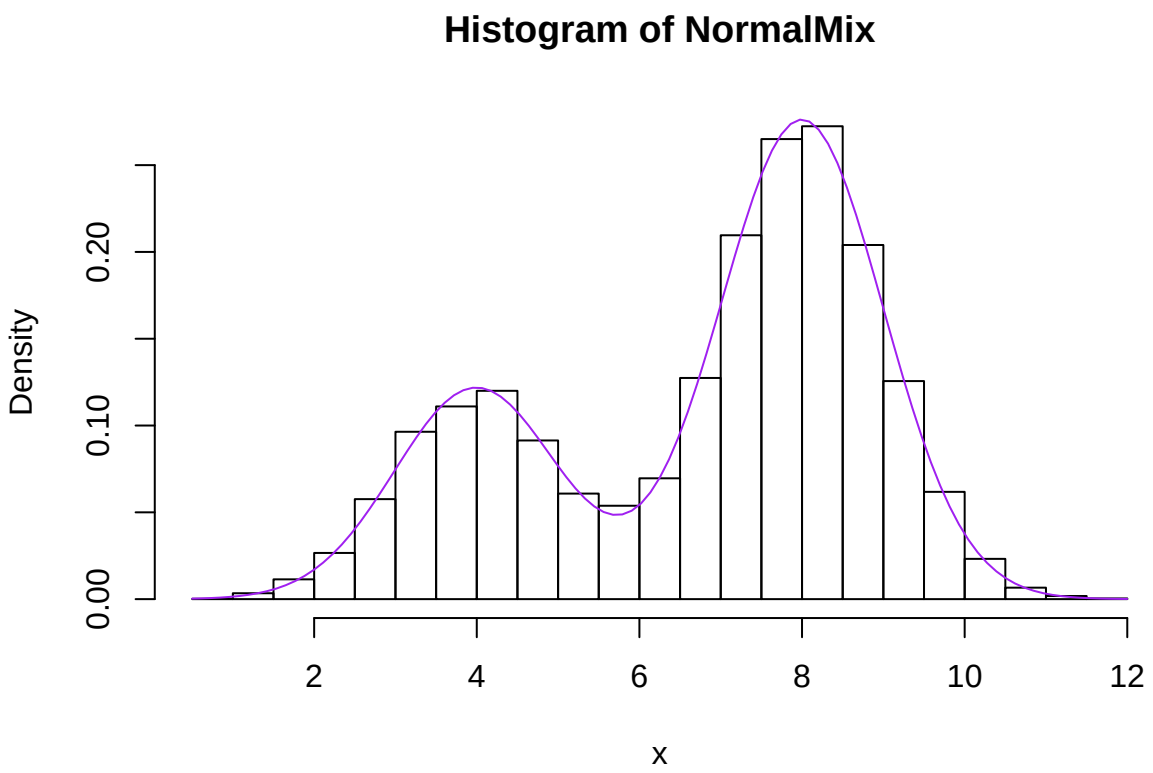
```
# Run optimization procedure
# Extract minimum
suppressWarnings(nlm(neg.NormalMix.ll, my_params, data = NormalMix)$minimum)
```

```
## [1] 19814.09
```

```
# Extract estimates
suppressWarnings(NormalMix.MLE <- nlm(neg.NormalMix.ll, my_params, data = NormalMix)$estimate)
NormalMix.MLE
```

```
## [1] 3.9914714 1.0034196 7.9981117 1.0018488 0.3062931
```

```
# Plot histogram with gamma curve using MLEs
hist(NormalMix,breaks=20,xlab="x",probability = T)
curve(NormalMix.MLE[5]*dnorm(x, mean=NormalMix.MLE[1], sd=NormalMix.MLE[2], log=FALSE)
      +(1-NormalMix.MLE[5])*dnorm(x, mean=NormalMix.MLE[3], sd=NormalMix.MLE[4], log=FALSE)
      ,add = TRUE, col = "purple")
```



v.) Approx. what percentage of males and females contribute to the distribution of X?

Looking at out probability distribution from the beginning. The deciding factor on how much either males or females contribute is given by δ. Therefore, looking at our MLE estimate for gamma, we get:

```
cat("Delta MLE is estimated as:", NormalMix.MLE[5])
```

```
## Delta MLE is estimated as: 0.3062931
```

```
cat("\nMales contribute", NormalMix.MLE[5]*100, "%\n")
```

```
##
## Males contribute 30.62931 %
```

```
cat("Females contribute",  100-(NormalMix.MLE[5]*100), "%")
```

```
## Females contribute 69.37069 %
```

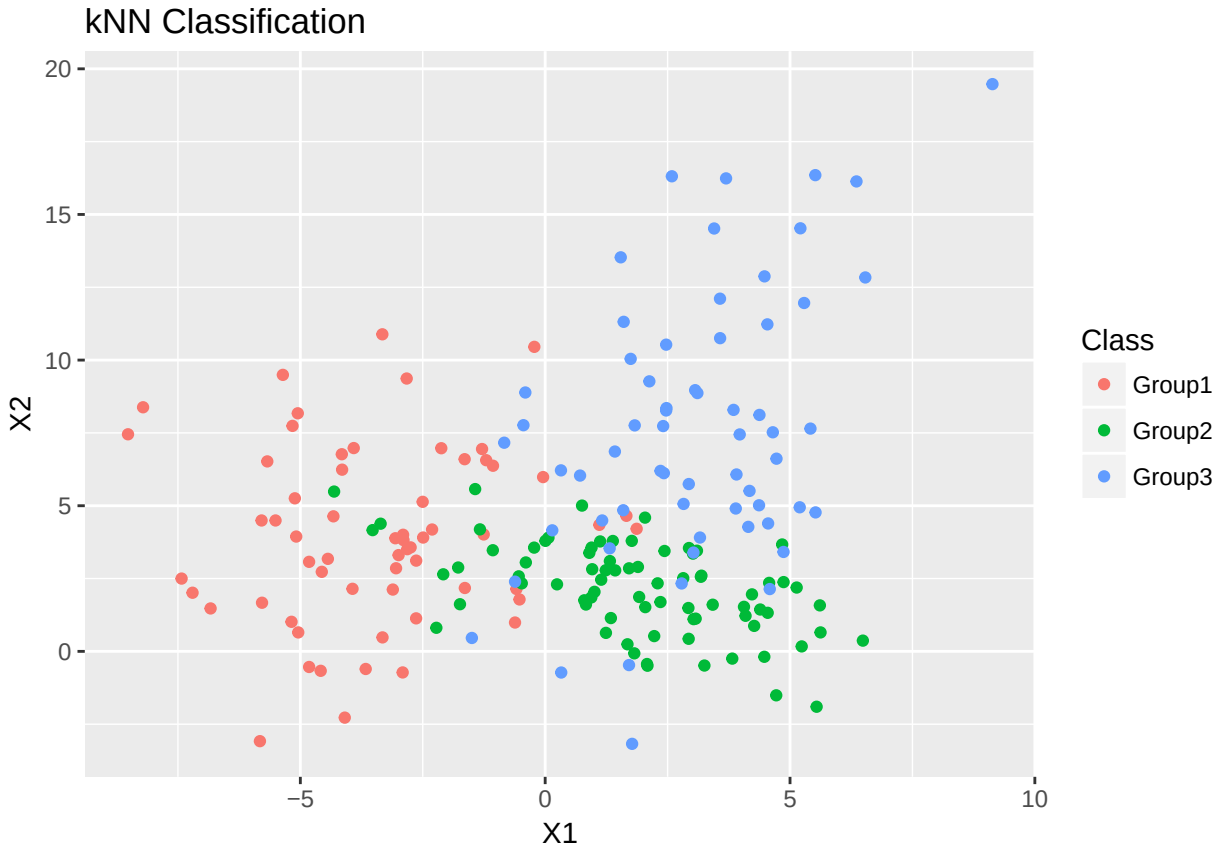
Problem 2: k-Nearest Neighbors

i.) Run the followig to ensure everyone has the same training and test dataset

```
kNNData <- read.csv("kNNData.csv")[,c("X1","X2","Class")]
set.seed(2)
test.index <- sample(1:nrow(kNNData),100,replace=F)
kNNData.test  <- kNNData[test.index, ]
kNNData.train <- kNNData[-test.index, ]
```

ii.) Run the following to visualize how the data responds to changes in features x_1 and x_2

```
library(ggplot2)
ggplot(data=kNNData.train)+
  geom_point(mapping=aes(x=X1,y=X2,col=Class))+
  labs(title="kNN Classification")
```



iii.) Modify the kNN.decision() function from class so that it can be applied here. Use K = 5 to test your function at the query points $(x_{1_{test}}, x_{2_{test}})$ = (0,10) and (0,5)

```
## kNN function
KNN.decision <- function(x1.new, x2.new, K = 5,
                         x1 = kNNData$X1,
                         x2 = kNNData$X2,
                         Class = kNNData$Class) {
  n <- length(x1)
  stopifnot(length(x2) == n, length(x1.new) == 1,
            length(x2.new) == 1, K <= n)

  dists <- sqrt((x1-x1.new)^2 + (x2-x2.new)^2)

  neighbors  <- order(dists)[1:K]
```

```
  neighb.dir <- Class[neighbors]
  choice     <- names(which.max(table(neighb.dir)))
  return(choice)
}
KNN.decision(0, 10)
```

```
## [1] "Group3"
```

```
KNN.decision(0, 5)
```

```
## [1] "Group2"
```

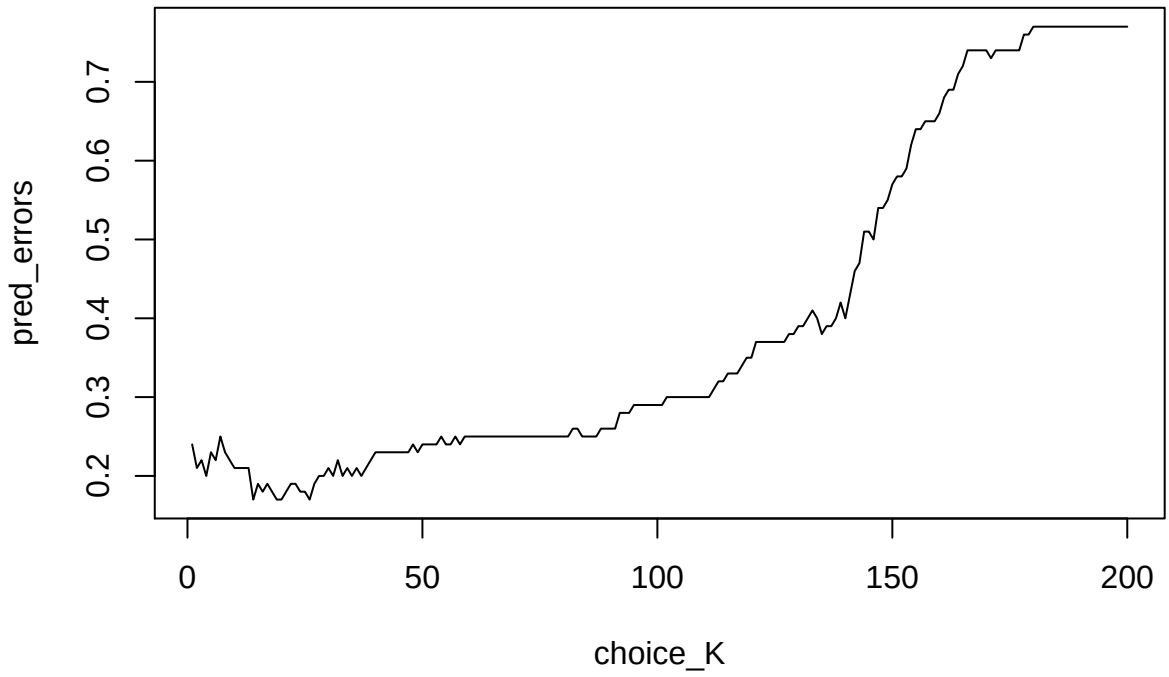
iv.) Compute the prediction error for k=5

```
n.test <- nrow(kNNData.test)
predictions <- rep(NA, n.test)

for (i in 1:n.test){
  predictions[i] <- KNN.decision(kNNData.test$X1[i],kNNData.test$X2[i],
                                 x1 = kNNData.train$X1, x2 = kNNData.train$X2,
                                 Class = kNNData.train$Class)
}
test.error <- sum(predictions != kNNData.test$Class)/n.test
test.error
```

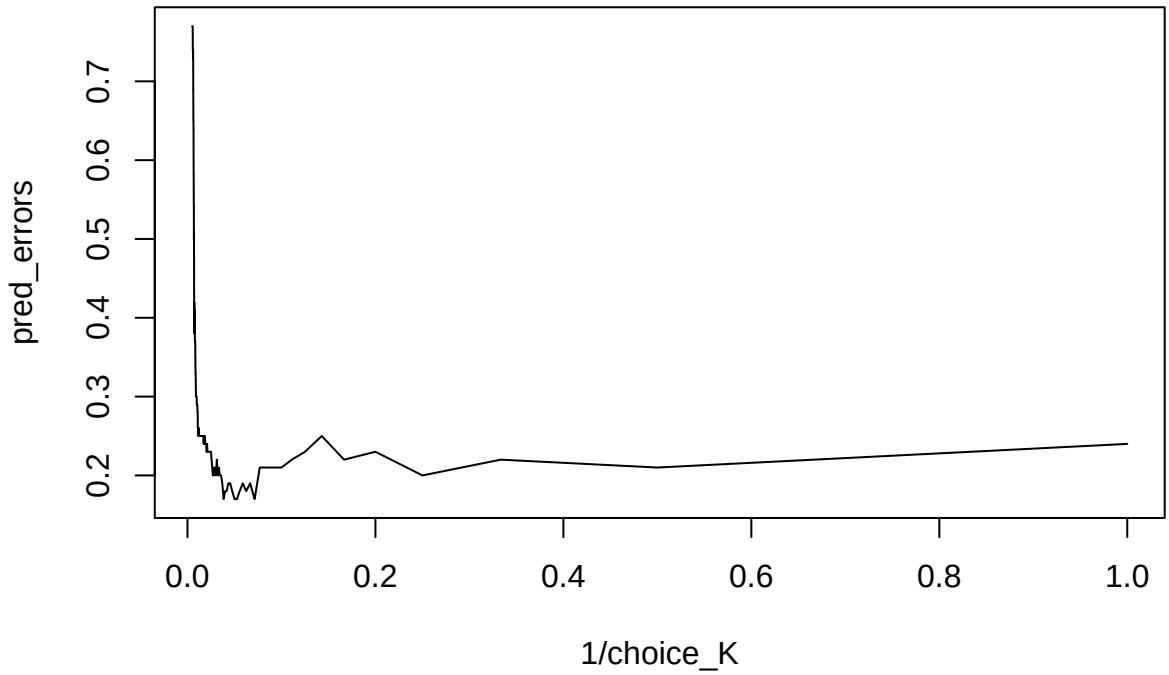
```
## [1] 0.23
```

v.) Compute the prediction error for K=1...200 and plot this error vs. K

```
choice_K <- c(1:200)
pred_errors <- list()

for(i in choice_K) {
  n.test <- nrow(kNNData.test)
  predictions <- rep(NA, n.test)

  for (j in 1:n.test){
    predictions[j] <- KNN.decision(kNNData.test$X1[j],kNNData.test$X2[j], K=i,
                                   x1 = kNNData.train$X1, x2 = kNNData.train$X2,
                                   Class = kNNData.train$Class)
  }
  test.error <- sum(predictions != kNNData.test$Class)/n.test
  pred_errors[i] = test.error
}
plot(choice_K, pred_errors, 'l')
```




```
plot(1/choice_K, pred_errors, 'l')
```



vi.) Using the plot above, what range would you choose for the tuning parameter K?

We would like to minimize the prediction errors when choosing K. Thus, using the plot above, there is a little patch where the errors are minimized before the graph rises again. We can get this range of values by finding the global min, looking at all indices of this global min and selecting the min and max of these indices.

```
pred_errors <- unlist(pred_errors)
pred_min <- min(pred_errors)
min_range <- which(pred_min == pred_errors)
min_range
```

```
## [1] 14 19 20 26
```

```
cat("Choose K between", min(min_range), "and", max(min_range))
```

```
## Choose K between 14 and 26
```

As a check, we can apply the rule of thumb to see what K we should select. The rule of thumb says that we should set K to the square root of the number of training samples:

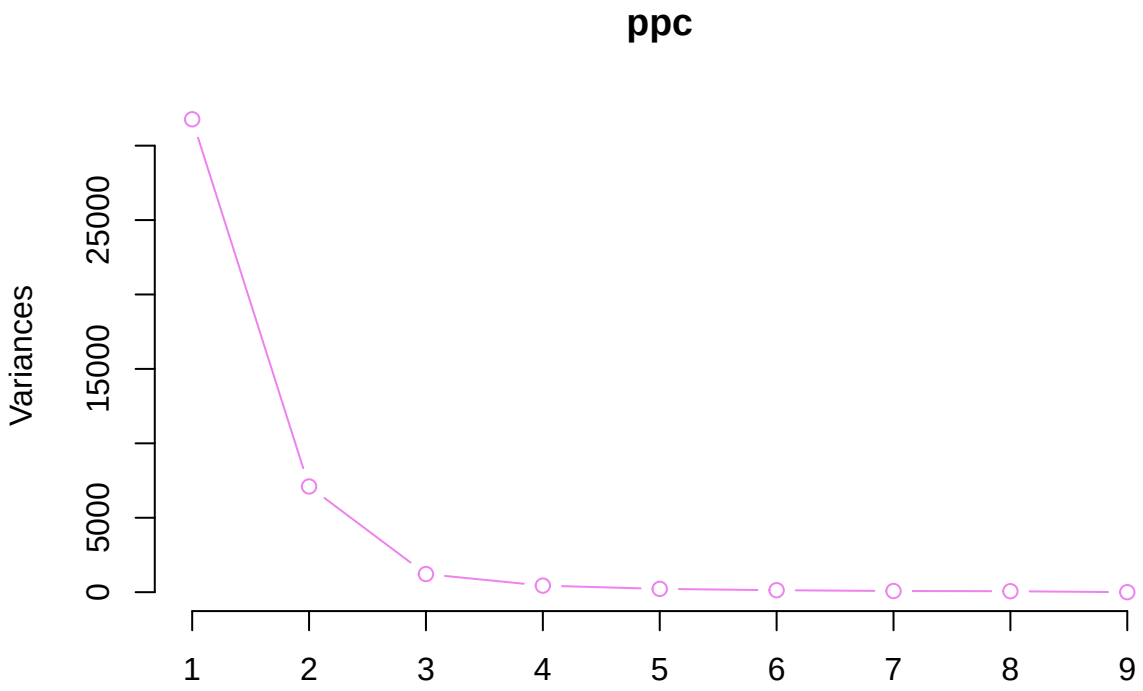
```
sqrt(nrow(kNNData.train))
```

```
## [1] 14.14214
```

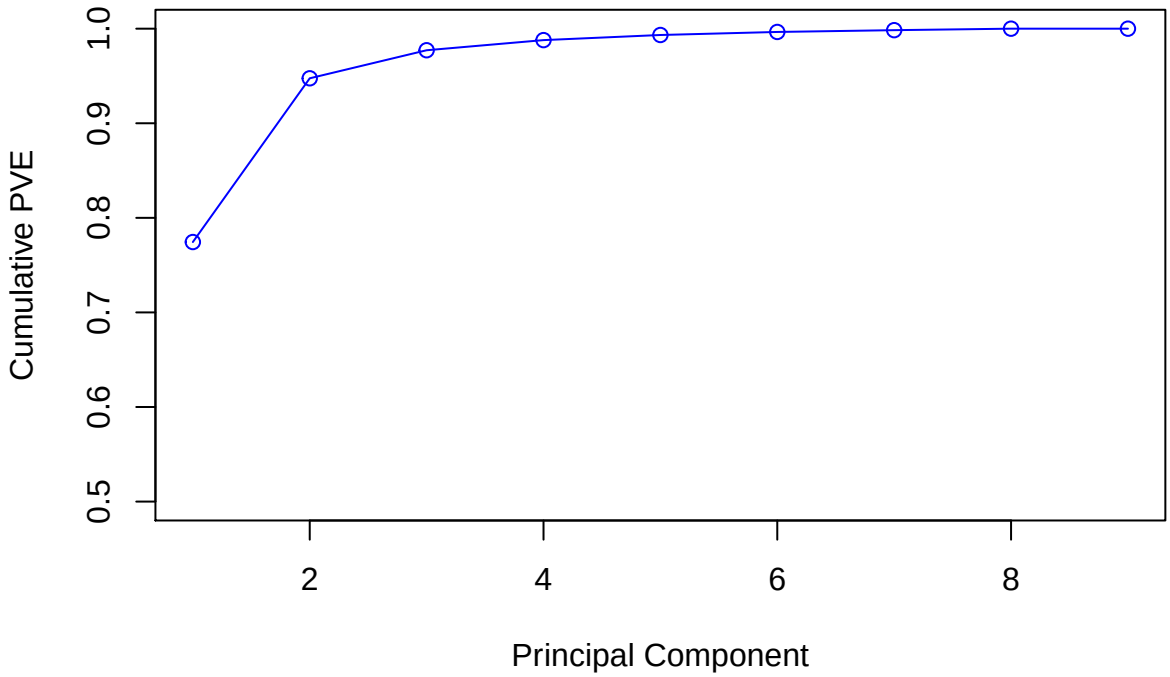
Problem 3: High Dimensional PCA

i.) How many components do we need to explain 95% of the variance?

```
daily.1995 <- read.csv("Daily1995-1.csv")

# Center each variable (center each day)
daily.cent <- t(scale(t(daily.1995),center=T,scale=F))

# Run PCA using prcomp() on the transpose of the data
ppc <- prcomp(t(daily.cent))

# scree plot
screeplot(ppc,type="lines",col="violet")
```




```
# Cumulative explained variance (scree plot)
CPVE <- (cumsum((ppc$sdev)^2)/sum((ppc$sdev)^2))
plot(1:9,CPVE,type="l",col="blue",
     ylim=c(.5,1),xlab="Principal Component",ylab="Cumulative PVE")
points(1:9,CPVE,col="blue")
```




```
# number of components to explain 95% of the variance
num_comp=1
while(CPVE[num_comp]<.95)
{
  num_comp=num_comp+1
}
cat("Number of components needed to explain 95% of the variance is:", num_comp)
```

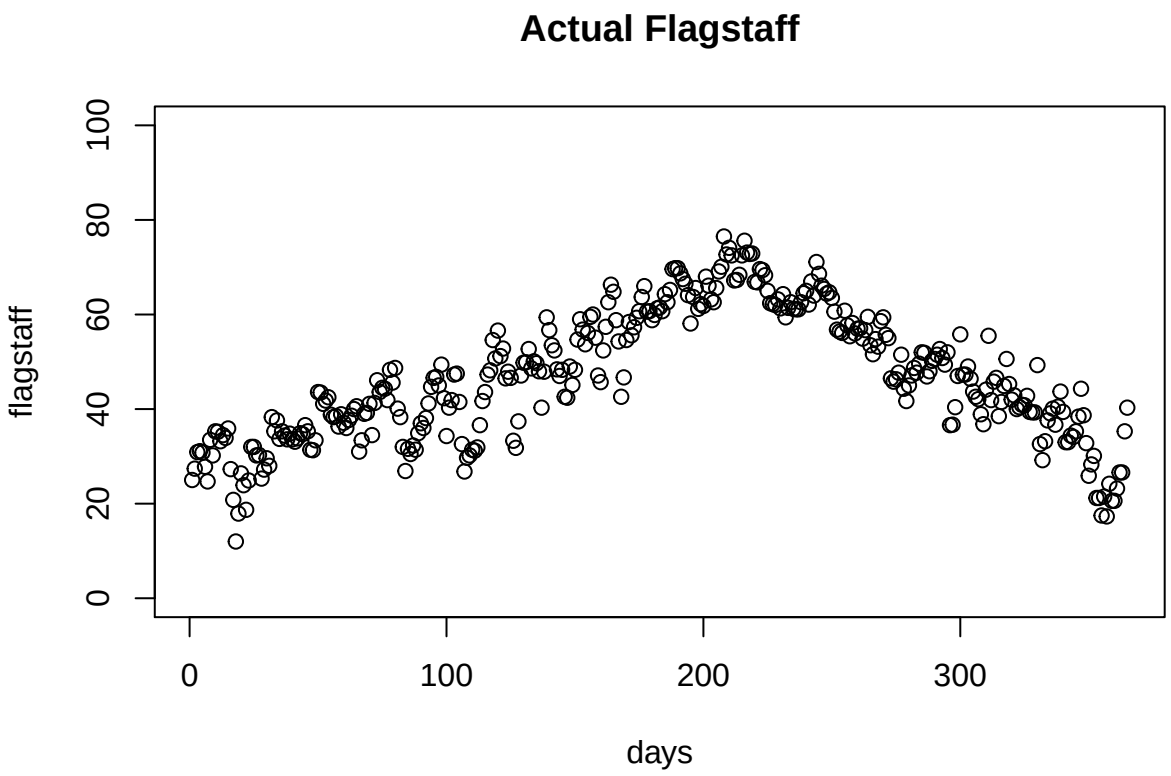
```
## Number of components needed to explain 95% of the variance is: 3
```

ii.) Construct the yearly weather for Flagstaff using the minimum number of PCs that explain 95% of the data's variation. Plot this constructed case with the actual data for Flagstaff. Make sure to label your plots appropriately.

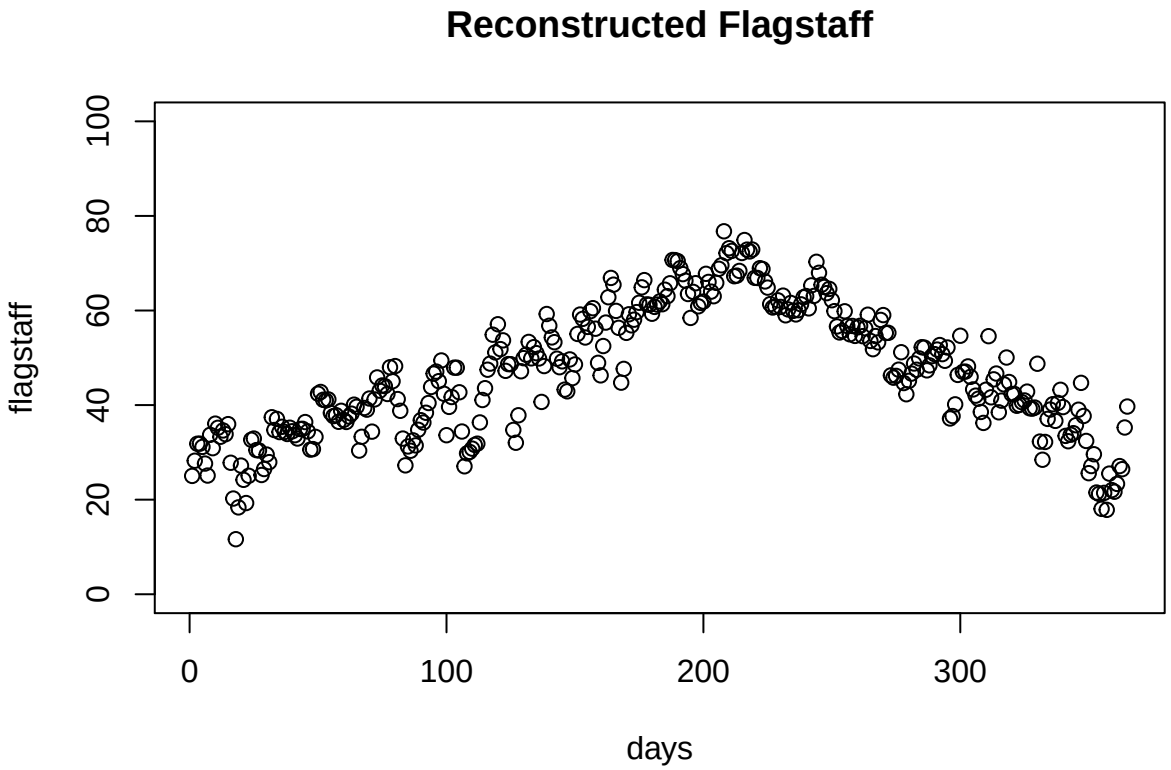
```
# How is this data organized?
dim(daily.1995)
```

```
## [1] 365   9
```

```
# Plot actual Flagstaff
plot(1:365,daily.1995[,9],xlab="days",ylim=c(0,100),ylab=names(daily.1995)[9], main="Actual Flagstaff")
```




```
# Reconstruct flagstaff with 3 pcs
#flag_reconstruct <- mean(daily.1995[,9])
daily.mean <- apply(daily.1995,1,mean)
flag_reconstruct <- daily.mean

for(i in 1:num_comp) {
  flag_reconstruct = flag_reconstruct + ppc$x[9,i]*ppc$rotation[,i]
}

#flag_reconstruct

# Plot reconstructed Flagstaff
plot(1:365,flag_reconstruct,xlab="days",ylim=c(0,100),ylab="flagstaff", main="Reconstructed Flagstaff")
```


Reconstructed Flagstaff
flagstaff
0
20
40
60
80
100
0
100
200
300
days