Homework 2: Unsupervised Learning and Clustering

Harvard CS 109B, Spring 2018

Jan 23, 2018

Contents

Libraries

```
library(gam)
library(ggplot2)
library(splines)
library(MASS)
library(readr) #for reading csv files
library(scales) #for controlling chart alpha values
library(dplyr)
library(lubridate)
library(gridExtra)
library(mclust)
library(factoextra)
library(NbClust)
library(dbscan)
library(DMwR)
```

```r
library(cluster)
library(corrplot)
```

Seed for Pseudo-RNG

```r
set.seed(11235813)
```

Homework 2 is due February 19, 2018

"Handy" Algorithms

In this assignment, you will be working with data collected from a motion capture camera system. The system was used to record 12 different users performing 5 distinct hand postures with markers attached to a left-handed glove. A set of markers on the back of the glove was used to establish a local coordinate system for the hand, and 11 additional markers were attached the the thumb and fingers of the glove. Three markers were attached to the thumb with one above the thumbnail and the other two on the knuckles. Finally, 2 markers were attached to each finger with one above the fingernail and the other in the middle of the finger. A total of 36 features were collected resulting from the camera system. Two other variables in the dataset were the ID of the user and the posture that the user made.

The data were partially preprocessed. First, all markers were transformed to the local coordinate system of the record containing them. Second, each transformed marker with a norm greater than 200 millimeters was eliminated. Finally, any record that contained fewer than three markers was removed.

A few issues with the data are worth noting. Based on the manner in which data were captured, it is likely that, for a given record and user, there exists a near duplicate record originating from the same user. Additionally, There are many instances of missing data in the feature set. These instances are denoted with a ? in the dataset. Finally, there is the potential for imbalanced classes, as there is no guarantee that each user and/or posture is represented with equal frequency in the dataset.

The dataset, provided in CSV format, contains 78,095 rows and 38 columns. Each row corresponds to a single instant or frame as recorded by the camera system. The data are represented in the following manner:

`Class - Integer. The hand posture of the given obervation, with 1=Fist (with thumb out), 2=Stop (hand flat), 3=Point1 (point with index finger), 4=Point2 (point with index and middle fingers), 5=Grab (fingers curled as if to grab).`

`User - Integer. The ID of the user that contributed the record.`

`X0, Y0, Z0, X1, Y1, Z1, ..., X11, Y11, Z11 - Real. The x-coordinate, y-coordinate and z-coordinate of the twelve unlabeled marker positions.`

1 Missing Data Imputation

The data contain many missing values. Before attempting to perform any statistical procedures, we will need to address the missing data. One way to address missing data is to impute it.

Given the knowledge of how the data was collected, we can hypothesize that there are two ways in which the data might cluster together: by user and by posture. Perhaps the users have significantly different heights and/or hand sizes, resulting in the data generated by each user to be distinct from each other. Or, perhaps the hand postures are sufficiently unique such that the markers on the glove tend to be grouped together by the posture, regardless of who the user is. We will examine these hypotheses to see if either one provides a reasonable way to impute the data.

For this part of the assignment, you will want to use the following libraries:

```
# import libraries

library(dplyr)

library(ggplot2)

library(cluster)

library(mclust)

library(factoextra)

library(NbClust)

library(dbscan)
```

Write code to impute the missing data values with the mean of their respective feature (column), grouped by both users and postures. That is, you should create two new dataframes: one where the missing values are replaced by the mean of the user's feature, and one where the missing values are replaced by the mean of the posture's feature.

Hint: when loaded into R, the raw CSV might list the observations as factors. You will want to change that. One way to convert factors to numeric is to cast the columns of the dataframe like so:

```
for (i in 1:ncol(data)) {

  data[,i] <- as.numeric(as.character(data[,i]))

}
```

The "dplyr" package might also be useful for data cleaning.

Helper Function: Col means data imputation

Hint: function to impute means for one column, you will want to adapt this for all the necessary columns.

```
impute_column <- function(column) {

missing_indices <- which(is.na(column))

column_mean <- mean(as.numeric(column[-missing_indices]))

column[missing_indices] <- column_mean

return(column)

}
```

For data imputation, you can use this code after you make necessary adjustment above

First, create each imputed posture data frame

```
`for (posture in unique(mydata$Class)) {

  df <- mydata %>% filter(Class == posture)

  df_imputed_means <- impute_means(df)

  assign(paste0("imputed_posture_means", posture), df_imputed_means)

}`
```

Second, create each imputed user data frame

```
`for (user in unique(mydata$User)) {

  df <- mydata %>% filter(User == user)

  df_imputed_means <- impute_means(df)

  assign(paste0("imputed_user_means", user), df_imputed_means)
}`
```

Then, stack data frames back together (inserting appropriate variable names instead of ellipsis).

```
imputed_posture_all <- rbind(imputed_posture_means1, ..., imputed_posture_means5)
```

```
imputed_user_all <- rbind(imputed_user_means0, ... , imputed_user_means14)
```

Finally manage your stack and remove clutter (inserting appropriate variable names instead of ellipsis).

```
`rm(imputed_posture_means1, ... , imputed_posture_means5)

rm(imputed_user_means0, ... , imputed_user_means14)

rm(posture)
rm(user)
rm(df)
rm(df_imputed_means)`
```

Solution 1

Load Data

```
# import hand posture data
mydata <- read.csv('postures.csv',
                   na.strings = c("NA", "?"))
```

Data inspection

```
#inspect
summary(mydata)
```

```
##      Class            User              X0                Y0
##  Min.   :1.000   Min.   : 0.000   Min.   :-108.55   Min.   :-98.23
##  1st Qu.:2.000   1st Qu.: 5.000   1st Qu.:  29.30   1st Qu.: 63.50
##  Median :3.000   Median : 9.000   Median :  54.62   Median : 86.53
##  Mean   :2.984   Mean   : 7.959   Mean   :  50.35   Mean   : 85.81
##  3rd Qu.:4.000   3rd Qu.:12.000   3rd Qu.:  72.49   3rd Qu.:113.11
##  Max.   :5.000   Max.   :14.000   Max.   : 190.02   Max.   :169.18
##
##        Z0                  X1                Y1                Z1
##  Min.   :-126.771   Min.   :-111.69   Min.   :-96.14   Min.   :-166.0068
##  1st Qu.: -56.357   1st Qu.:  28.76   1st Qu.: 64.16   1st Qu.: -57.3604
##  Median : -30.864   Median :  54.22   Median : 87.54   Median : -30.1853
##  Mean   : -29.985   Mean   :  49.60   Mean   : 86.19   Mean   : -29.5096
##  3rd Qu.:  -1.419   3rd Qu.:  71.76   3rd Qu.:116.23   3rd Qu.:  -0.3681
##  Max.   : 113.345   Max.   : 188.69   Max.   :170.21   Max.   : 104.6979
##
##        X2                Y2                Z2                  X3
##  Min.   :-106.89   Min.   :-100.79   Min.   :-129.5953   Min.   :-111.76
##  1st Qu.:  25.17   1st Qu.:  58.05   1st Qu.: -58.6543   1st Qu.:  23.86
##  Median :  53.81   Median :  86.46   Median : -32.3565   Median :  54.14
##  Mean   :  48.61   Mean   :  83.77   Mean   : -30.5609   Mean   :  48.49
##  3rd Qu.:  71.56   3rd Qu.: 106.66   3rd Qu.:  -0.9461   3rd Qu.:  71.44
##  Max.   : 188.76   Max.   : 168.19   Max.   : 104.5909   Max.   : 151.03
##                                                         NA's   :690
##        Y3               Z3                 X4                Y4
##  Min.   :-97.60   Min.   :-143.54   Min.   :-99.11   Min.   :-97.95
##  1st Qu.: 54.07   1st Qu.: -59.37   1st Qu.: 22.83   1st Qu.: 48.96
##  Median : 85.75   Median : -34.02   Median : 53.86   Median : 85.78
##  Mean   : 82.06   Mean   : -31.15   Mean   : 48.41   Mean   : 80.42
##  3rd Qu.:105.53   3rd Qu.:  -1.44   3rd Qu.: 71.97   3rd Qu.:105.48
##  Max.   :168.29   Max.   : 129.32   Max.   :172.28   Max.   :168.26
##  NA's   :690      NA's   :690       NA's   :3120     NA's   :3120
##        Z4                X5                Y5                Z5
##  Min.   :-157.199   Min.   :-120.66   Min.   :-97.47   Min.   :-135.699
##  1st Qu.: -60.548   1st Qu.:  19.54   1st Qu.: 49.25   1st Qu.: -58.994
##  Median : -35.166   Median :  51.95   Median : 88.14   Median : -31.714
##  Mean   : -31.994   Mean   :  47.04   Mean   : 81.39   Mean   : -30.270
##  3rd Qu.:  -2.142   3rd Qu.:  71.89   3rd Qu.:107.84   3rd Qu.:   0.117
##  Max.   : 119.237   Max.   : 180.56   Max.   :167.93   Max.   : 110.899
##  NA's   :3120       NA's   :13023     NA's   :13023    NA's   :13023
##        X6                Y6                Z6                 X7
##  Min.   :-100.08   Min.   :-67.28   Min.   :-153.450   Min.   :-108.61
##  1st Qu.:  15.49   1st Qu.: 51.66   1st Qu.: -55.375   1st Qu.:  13.08
##  Median :  52.09   Median : 92.15   Median : -26.457   Median :  49.60
##  Mean   :  45.68   Mean   : 83.73   Mean   : -26.639   Mean   :  44.46
##  3rd Qu.:  72.61   3rd Qu.:111.47   3rd Qu.:   2.434   3rd Qu.:  75.72
##  Max.   : 176.41   Max.   :168.60   Max.   : 117.915   Max.   : 189.22
##  NA's   :25848     NA's   :25848    NA's   :25848      NA's   :39152
##        Y7               Z7                 X8               Y8
##  Min.   :-64.97   Min.   :-113.73   Min.   :-121.18   Min.   :-65.08
##  1st Qu.: 63.54   1st Qu.: -45.68   1st Qu.:  17.04   1st Qu.: 53.67
##  Median : 93.80   Median : -19.43   Median :  55.95   Median : 92.18
##  Mean   : 88.46   Mean   : -20.37   Mean   :  48.21   Mean   : 86.03
##  3rd Qu.:119.19   3rd Qu.:   6.59   3rd Qu.:  79.63   3rd Qu.:121.91
##  Max.   :169.13   Max.   : 117.82   Max.   : 173.91   Max.   :169.32
##  NA's   :39152    NA's   :39152     NA's   :47532     NA's   :47532
##        Z8                X9               Y9               Z9
##  Min.   :-142.65   Min.   :-99.23   Min.   :-64.73   Min.   :-113.40
##  1st Qu.: -52.52   1st Qu.: 26.11   1st Qu.: 44.04   1st Qu.: -55.62
##  Median : -22.46   Median : 62.00   Median : 84.74   Median : -26.80
##  Mean   : -24.36   Mean   : 54.75   Mean   : 80.55   Mean   : -27.78
##  3rd Qu.:   6.42   3rd Qu.: 83.77   3rd Qu.:115.56   3rd Qu.:   3.07
##  Max.   : 119.21   Max.   :174.05   Max.   :167.94   Max.   : 123.38
##  NA's   :47532     NA's   :54128    NA's   :54128    NA's   :54128
##       X10              Y10              Z10               X11
##  Min.   :-80.20   Min.   :-65.02   Min.   :-112.67   Min.   :-96.95
##  1st Qu.: 20.42   1st Qu.: 37.70   1st Qu.: -62.96   1st Qu.:-62.75
##  Median : 61.84   Median : 79.00   Median : -32.91   Median :-48.59
##  Mean   : 53.76   Mean   : 74.00   Mean   : -29.74   Mean   :-29.70
##  3rd Qu.: 83.06   3rd Qu.:100.95   3rd Qu.:   6.07   3rd Qu.: -2.20
##  Max.   :149.49   Max.   :168.35   Max.   : 108.46   Max.   : 84.68
##  NA's   :63343    NA's   :63343    NA's   :63343     NA's   :78064
##       Y11              Z11
##  Min.   :-65.43   Min.   :-48.27
##  1st Qu.: 24.50   1st Qu.: -4.52
##  Median : 38.94   Median : 11.31
##  Mean   : 25.96   Mean   :  1.70
##  3rd Qu.: 40.81   3rd Qu.: 12.82
##  Max.   :127.95   Max.   : 18.06
##  NA's   :78064    NA's   :78064
```

```
str(mydata)
```

```
## 'data.frame':    78095 obs. of  38 variables:
##  $ Class: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ User : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ X0   : num  54.3 56.5 55.8 55.3 55.1 ...
##  $ Y0   : num  71.5 72.3 72.5 71.7 71.4 ...
##  $ Z0   : num  -64.8 -61.9 -62.6 -63.7 -64.2 ...
##  $ X1   : num  76.9 39.1 38 36.6 36.2 ...
##  $ Y1   : num  42.5 82.5 82.6 81.9 81.6 ...
##  $ Z1   : num  -72.8 -49.6 -50.6 -52.8 -53.5 ...
##  $ X2   : num  36.6 79.2 78.5 86.3 77 ...
##  $ Y2   : num  81.7 43.3 43.6 68.2 42.4 ...
##  $ Z2   : num  -52.9 -70 -70.7 -72.2 -72.6 ...
##  $ X3   : num  85.2 87.5 86.8 61.6 86.4 ...
##  $ Y3   : num  67.7 68.4 68.9 11.3 67.9 ...
##  $ Z3   : num  -73.7 -70.7 -71.1 -69 -72.4 ...
##  $ X4   : num  59.2 61.6 61.7 77.4 61.3 ...
##  $ Y4   : num  10.7 11.8 11.8 42.7 10.8 ...
##  $ Z4   : num  -71.3 -68.8 -68.9 -72 -69.3 ...
##  $ X5   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Y5   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Z5   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ X6   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Y6   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Z6   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ X7   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Y7   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Z7   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ X8   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Y8   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Z8   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ X9   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Y9   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Z9   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ X10  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Y10  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Z10  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ X11  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Y11  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Z11  : num  NA NA NA NA NA NA NA NA NA NA ...
```

```
cat("Data size: ", dim(mydata), "\n")
```

```
## Data size:  78095 38
```

```
head(mydata)
```

```
##   Class User       X0       Y0        Z0       X1       Y1        Z1
## 1     1    0 54.26388 71.46678 -64.80771 76.89563 42.46250 -72.78055
## 2     1    0 56.52756 72.26661 -61.93525 39.13598 82.53853 -49.59651
## 3     1    0 55.84993 72.46906 -62.56279 37.98880 82.63135 -50.60626
## 4     1    0 55.32965 71.70727 -63.68896 36.56186 81.86875 -52.75278
## 5     1    0 55.14240 71.43561 -64.17730 36.17582 81.55687 -53.47575
## 6     1    0 55.58118 71.64120 -63.70314 34.85056 81.35204 -54.74744
##         X2       Y2        Z2       X3       Y3        Z3       X4
## 1 36.62123 81.68056 -52.91927 85.23226 67.74922 -73.68413 59.18858
## 2 79.22374 43.25409 -69.98249 87.45087 68.40081 -70.70399 61.58745
## 3 78.45153 43.56740 -70.65849 86.83539 68.90792 -71.13834 61.68643
## 4 86.32063 68.21465 -72.22846 61.59616 11.25065 -68.95643 77.38723
## 5 76.98614 42.42685 -72.57474 86.36875 67.90126 -72.44465 61.27540
## 6 77.07851 42.54825 -72.48549 86.85133 68.01184 -71.90994 61.85685
##         Y4        Z4 X5 Y5 Z5 X6 Y6 Z6 X7 Y7 Z7 X8 Y8 Z8 X9 Y9 Z9 X10 Y10
## 1 10.67894 -71.29778 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  NA  NA
## 2 11.77992 -68.82742 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  NA  NA
## 3 11.79344 -68.88932 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  NA  NA
## 4 42.71783 -72.01515 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  NA  NA
## 5 10.84111 -69.27991 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  NA  NA
## 6 10.85197 -68.85375 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  NA  NA
##   Z10 X11 Y11 Z11
## 1  NA  NA  NA  NA
## 2  NA  NA  NA  NA
## 3  NA  NA  NA  NA
## 4  NA  NA  NA  NA
## 5  NA  NA  NA  NA
## 6  NA  NA  NA  NA
```

Data Cleaning and missing values imputation

Due to data inspection we see that the first row is a headings row, hence to be removed.

```
mydata <- mydata[-1,]
#data check
head(mydata) ## sanity check --->OK
```

```
##   Class User       X0       Y0        Z0       X1       Y1        Z1
## 2     1    0 56.52756 72.26661 -61.93525 39.13598 82.53853 -49.59651
## 3     1    0 55.84993 72.46906 -62.56279 37.98880 82.63135 -50.60626
## 4     1    0 55.32965 71.70727 -63.68896 36.56186 81.86875 -52.75278
## 5     1    0 55.14240 71.43561 -64.17730 36.17582 81.55687 -53.47575
## 6     1    0 55.58118 71.64120 -63.70314 34.85056 81.35204 -54.74744
## 7     1    0 34.52282 81.45732 -54.90099 55.82769 71.87879 -63.19437
##         X2       Y2        Z2       X3       Y3        Z3       X4
## 2 79.22374 43.25409 -69.98249 87.45087 68.40081 -70.70399 61.58745
## 3 78.45153 43.56740 -70.65849 86.83539 68.90792 -71.13834 61.68643
## 4 86.32063 68.21465 -72.22846 61.59616 11.25065 -68.95643 77.38723
## 5 76.98614 42.42685 -72.57474 86.36875 67.90126 -72.44465 61.27540
## 6 77.07851 42.54825 -72.48549 86.85133 68.01184 -71.90994 61.85685
## 7 86.90265 68.31268 -71.64207 61.82953 11.01498 -68.95880 76.95451
##         Y4        Z4 X5 Y5 Z5 X6 Y6 Z6 X7 Y7 Z7 X8 Y8 Z8 X9 Y9 Z9 X10 Y10
## 2 11.77992 -68.82742 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  NA  NA
## 3 11.79344 -68.88932 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  NA  NA
## 4 42.71783 -72.01515 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  NA  NA
## 5 10.84111 -69.27991 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  NA  NA
## 6 10.85197 -68.85375 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  NA  NA
## 7 42.73464 -72.50062 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  NA  NA
##   Z10 X11 Y11 Z11
## 2  NA  NA  NA  NA
## 3  NA  NA  NA  NA
## 4  NA  NA  NA  NA
## 5  NA  NA  NA  NA
## 6  NA  NA  NA  NA
## 7  NA  NA  NA  NA
```

Helper Functions: Col means data imputation

```r
#################################################################
# function to impute means for one column
impute_column <- function(column) {
  missing_indices <- which(is.na(column))
  column_mean <- mean(as.numeric(column[-missing_indices]))
  column[missing_indices] <- column_mean
  return(column)
}
################################################################

################################################################
# function to impute missing data with column means
# can be done by posture, user, or both
impute_means <- function(df) {
  # remove warnings from changing '?' to 'NA'
  options(warn=-1)

  # convert everything to numeric
  for (i in 1:38) {
    df[,i] <- as.numeric(as.character(df[,i]))
  }

  # apply function across all feature columns
  for (i in 3:38) {
    df[,i] <- impute_column(df[,i])
  }

  # if every observation in column is NA, impute mean of everything in dataframe
  overall_mean <- mean(colMeans(df[,3:38], na.rm=T), na.rm=T)
  for (i in 3:38) {
    if (length(unique(df[,i])) == 1) {
      df[,i] <- overall_mean
    }
  }

  # turn warnings back on
  options(warn=0)

  return(df)
}
##############################################################
```

Data Imputation

```
# create each imputed posture dataframe
for (posture in unique(mydata$Class)) {
  df <- mydata %>% filter(Class == posture)
  df_imputed_means <- impute_means(df)
  assign(paste0("imputed_posture_means", posture), df_imputed_means)
}

# create each imputed user dataframe
for (user in unique(mydata$User)) {
  df <- mydata %>% filter(User == user)
  df_imputed_means <- impute_means(df)
  assign(paste0("imputed_user_means", user), df_imputed_means)
}

# stack dataframes back together
imputed_posture_all <- rbind(imputed_posture_means1, imputed_posture_means2,
                             imputed_posture_means3, imputed_posture_means4,
                             imputed_posture_means5)

imputed_user_all <- rbind(imputed_user_means0, imputed_user_means1,
                          imputed_user_means2, imputed_user_means4,
                          imputed_user_means5, imputed_user_means6,
                          imputed_user_means7, imputed_user_means8,
                          imputed_user_means9, imputed_user_means10,
                          imputed_user_means11, imputed_user_means12,
                          imputed_user_means13, imputed_user_means14)

# data checks
head(imputed_user_all)
```

```
##   Class User       X0       Y0        Z0       X1       Y1        Z1
## 1     1    0 56.52756 72.26661 -61.93525 39.13598 82.53853 -49.59651
## 2     1    0 55.84993 72.46906 -62.56279 37.98880 82.63135 -50.60626
## 3     1    0 55.32965 71.70727 -63.68896 36.56186 81.86875 -52.75278
## 4     1    0 55.14240 71.43561 -64.17730 36.17582 81.55687 -53.47575
## 5     1    0 55.58118 71.64120 -63.70314 34.85056 81.35204 -54.74744
## 6     1    0 34.52282 81.45732 -54.90099 55.82769 71.87879 -63.19437
##         X2       Y2        Z2       X3       Y3        Z3       X4
## 1 79.22374 43.25409 -69.98249 87.45087 68.40081 -70.70399 61.58745
## 2 78.45153 43.56740 -70.65849 86.83539 68.90792 -71.13834 61.68643
## 3 86.32063 68.21465 -72.22846 61.59616 11.25065 -68.95643 77.38723
## 4 76.98614 42.42685 -72.57474 86.36875 67.90126 -72.44465 61.27540
## 5 77.07851 42.54825 -72.48549 86.85133 68.01184 -71.90994 61.85685
## 6 86.90265 68.31268 -71.64207 61.82953 11.01498 -68.95880 76.95451
##         Y4        Z4       X5       Y5        Z5       X6       Y6      Z6
## 1 11.77992 -68.82742 36.82819 97.15847 -31.14014 38.14252 99.61602 -26.643
## 2 11.79344 -68.88932 36.82819 97.15847 -31.14014 38.14252 99.61602 -26.643
## 3 42.71783 -72.01515 36.82819 97.15847 -31.14014 38.14252 99.61602 -26.643
## 4 10.84111 -69.27991 36.82819 97.15847 -31.14014 38.14252 99.61602 -26.643
## 5 10.85197 -68.85375 36.82819 97.15847 -31.14014 38.14252 99.61602 -26.643
## 6 42.73464 -72.50062 36.82819 97.15847 -31.14014 38.14252 99.61602 -26.643
##         X7       Y7        Z7       X8       Y8        Z8       X9
## 1 47.92583 100.3876 -17.28761 51.48269 101.5622 -17.21505 51.82053
## 2 47.92583 100.3876 -17.28761 51.48269 101.5622 -17.21505 51.82053
## 3 47.92583 100.3876 -17.28761 51.48269 101.5622 -17.21505 51.82053
## 4 47.92583 100.3876 -17.28761 51.48269 101.5622 -17.21505 51.82053
## 5 47.92583 100.3876 -17.28761 51.48269 101.5622 -17.21505 51.82053
## 6 47.92583 100.3876 -17.28761 51.48269 101.5622 -17.21505 51.82053
##         Y9      Z9      X10      Y10       Z10     X11     Y11     Z11
## 1 100.5583 -15.739 48.58361 102.6991 -18.53954 34.7282 34.7282 34.7282
## 2 100.5583 -15.739 48.58361 102.6991 -18.53954 34.7282 34.7282 34.7282
## 3 100.5583 -15.739 48.58361 102.6991 -18.53954 34.7282 34.7282 34.7282
## 4 100.5583 -15.739 48.58361 102.6991 -18.53954 34.7282 34.7282 34.7282
## 5 100.5583 -15.739 48.58361 102.6991 -18.53954 34.7282 34.7282 34.7282
## 6 100.5583 -15.739 48.58361 102.6991 -18.53954 34.7282 34.7282 34.7282
```

```r
head(imputed_posture_all)
```

```
##   Class User       X0       Y0        Z0       X1       Y1        Z1
## 1     1    0 56.52756 72.26661 -61.93525 39.13598 82.53853 -49.59651
## 2     1    0 55.84993 72.46906 -62.56279 37.98880 82.63135 -50.60626
## 3     1    0 55.32965 71.70727 -63.68896 36.56186 81.86875 -52.75278
## 4     1    0 55.14240 71.43561 -64.17730 36.17582 81.55687 -53.47575
## 5     1    0 55.58118 71.64120 -63.70314 34.85056 81.35204 -54.74744
## 6     1    0 34.52282 81.45732 -54.90099 55.82769 71.87879 -63.19437
##         X2       Y2        Z2       X3       Y3        Z3       X4
## 1 79.22374 43.25409 -69.98249 87.45087 68.40081 -70.70399 61.58745
## 2 78.45153 43.56740 -70.65849 86.83539 68.90792 -71.13834 61.68643
## 3 86.32063 68.21465 -72.22846 61.59616 11.25065 -68.95643 77.38723
## 4 76.98614 42.42685 -72.57474 86.36875 67.90126 -72.44465 61.27540
## 5 77.07851 42.54825 -72.48549 86.85133 68.01184 -71.90994 61.85685
## 6 86.90265 68.31268 -71.64207 61.82953 11.01498 -68.95880 76.95451
##         Y4        Z4       X5       Y5        Z5       X6       Y6
## 1 11.77992 -68.82742 44.28417 60.15773 -52.68218 36.12515 52.53732
## 2 11.79344 -68.88932 44.28417 60.15773 -52.68218 36.12515 52.53732
## 3 42.71783 -72.01515 44.28417 60.15773 -52.68218 36.12515 52.53732
## 4 10.84111 -69.27991 44.28417 60.15773 -52.68218 36.12515 52.53732
## 5 10.85197 -68.85375 44.28417 60.15773 -52.68218 36.12515 52.53732
## 6 42.73464 -72.50062 44.28417 60.15773 -52.68218 36.12515 52.53732
##          Z6       X7       Y7       Z7       X8       Y8       Z8       X9
## 1 -56.44404 25.44856 97.62212 1.356154 33.04124 100.1406 -1.07738 37.17284
## 2 -56.44404 25.44856 97.62212 1.356154 33.04124 100.1406 -1.07738 37.17284
## 3 -56.44404 25.44856 97.62212 1.356154 33.04124 100.1406 -1.07738 37.17284
## 4 -56.44404 25.44856 97.62212 1.356154 33.04124 100.1406 -1.07738 37.17284
## 5 -56.44404 25.44856 97.62212 1.356154 33.04124 100.1406 -1.07738 37.17284
## 6 -56.44404 25.44856 97.62212 1.356154 33.04124 100.1406 -1.07738 37.17284
##         Y9        Z9      X10      Y10       Z10      X11      Y11
## 1 87.56249 -10.66372 54.05703 76.31192 -27.77595 25.15897 25.15897
## 2 87.56249 -10.66372 54.05703 76.31192 -27.77595 25.15897 25.15897
## 3 87.56249 -10.66372 54.05703 76.31192 -27.77595 25.15897 25.15897
## 4 87.56249 -10.66372 54.05703 76.31192 -27.77595 25.15897 25.15897
## 5 87.56249 -10.66372 54.05703 76.31192 -27.77595 25.15897 25.15897
## 6 87.56249 -10.66372 54.05703 76.31192 -27.77595 25.15897 25.15897
##        Z11
## 1 25.15897
## 2 25.15897
## 3 25.15897
## 4 25.15897
## 5 25.15897
## 6 25.15897
```

Stack memory management

```
# remove clutter
rm(imputed_posture_means1, imputed_posture_means2, imputed_posture_means3,
   imputed_posture_means4, imputed_posture_means5)

rm(imputed_user_means0, imputed_user_means1, imputed_user_means2,
   imputed_user_means4, imputed_user_means5, imputed_user_means6,
   imputed_user_means7, imputed_user_means8, imputed_user_means9,
   imputed_user_means10, imputed_user_means11, imputed_user_means12,
   imputed_user_means13, imputed_user_means14)

rm(posture)
rm(user)
rm(df)
rm(df_imputed_means)
```

Another way to impute the missing data would be to use a k-nearest neighbors algorithm. One potential advantage of imputing the data this way is that could yield more accurate imputations, especially since we know that there are likely near duplicate records in the dataset. One potential drawback of imputing data this way is that, because there are observations with so many missing features, it might be difficult to assess the observation's nearest neighbor. It is also computationally expensive to run a kNN algorithm on such a large matrix—in time and resource complexity, since so many values must be cached.

Cons of Missing Data via Mean imputation: dilutes correlations making the mean more representative in the dataset.

---

2 Clustering with k-means

Now that we have imputed the missing values, we can investigate our hypotheses by examining how well the data clusters by user and by posture. In the following problem, we will explore a wider choice of options for the number of centroids.

We will first use the k-means algorithm to carry out the clustering.

(a) Using the "kmeans" function in R, run kmeans on the features using 14 centroids (repre- senting the 14 users). Do not run the algorithm on the entire dataset, as the eventual visualization can become unwieldy. Instead, obtain a random sample of 2,000 observations without replacement, and run the algorithm on the sampled values. Set a seed at '42' and set the 'nstart' parameter in the kmeans function to '46' to ensure that we can check your results. Hint: You can take a random sample of the dataframe's indices using R's "sample" function:

```
# take random sample

set.seed(42)

samp <- sample(x=1:length(imputed_user_target), size=2000, replace=F)

cluster_target <- imputed_user_target[samp]

cluster_features <- imputed_user_features[samp,]
```

Solution 2(a)

```r
# take random sample
# cluster imputed data by user
# split target from features
imputed_user_target <- imputed_user_all$User

# scale features
imputed_user_features <- scale(imputed_user_all[,3:38])

# take random sample
set.seed(42)
samp <- sample(x=1:length(imputed_user_target), size=2000, replace=F)
cluster_target <- imputed_user_target[samp]
cluster_features <- imputed_user_features[samp,]

data.scaled <- scale(cluster_features)
```

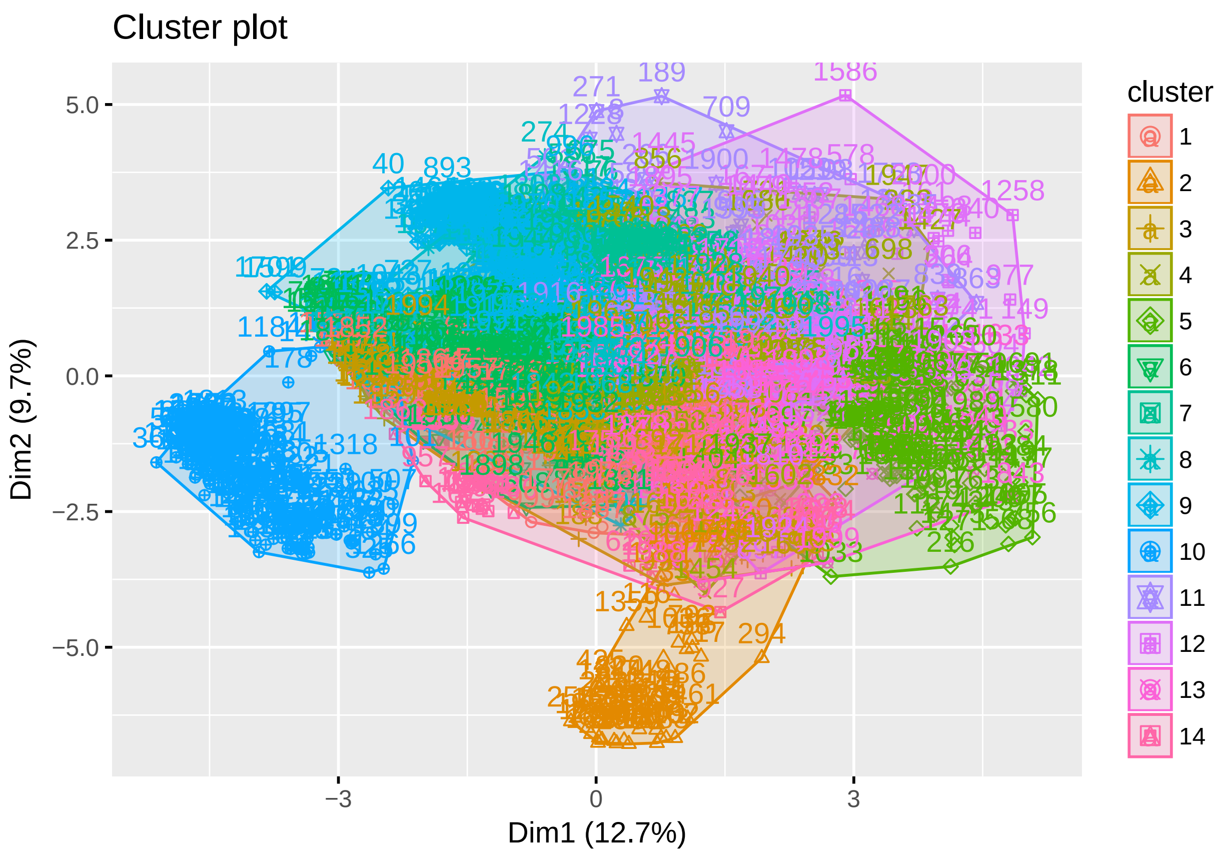
```r
# run kmeans
users_k_means = kmeans(cluster_features, 14, nstart = 46)
```

(b) Use the "fviz-cluster" function to visualize the results of your clustering algorithm (you will probably want to press the "Zoom" button in the plots section of R Studio so that you can see the results on a larger plot). How much of the variance in the data is explained by the first two principal components? Does it look like the data separate into 14 distinct clusters?

Solution 2(b)

ANSWER: There is no visible cluster separation - data does not separate into 14 distinct clusters. With 5 different classes, running kmeans with 5 centroids instead 14 might make more sense in this context.

```r
# visualize the results of the clustering algorithm
# probably want to press 'Zoom' in the plot window
fviz_cluster(users_k_means, cluster_features)
```



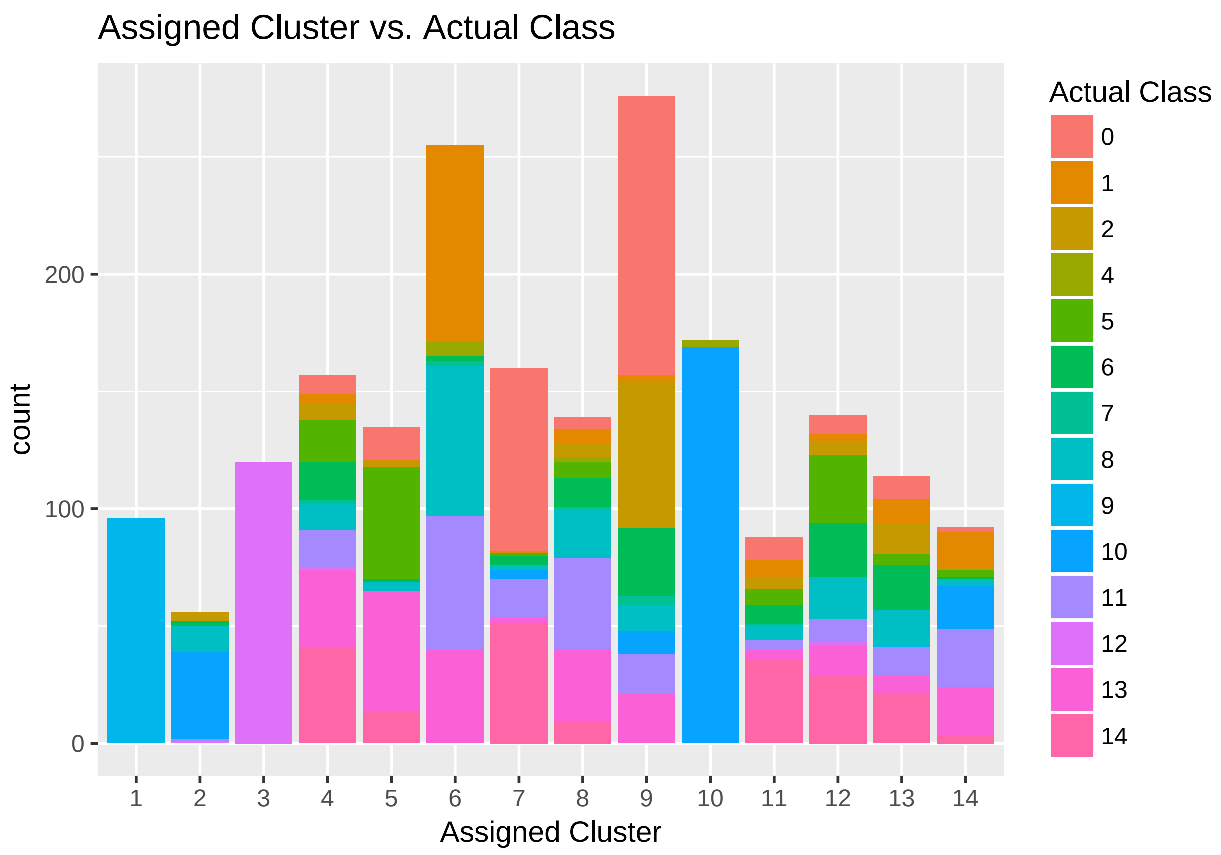
(c) Compare the results from your clustering algorithm to the actual users. Specifically, make a bar plot showing the assigned cluster from kmeans against the actual user of the observation. Have the area of each bar correspond to the the percentage of observations that belong to a given user. Based on this graph, does it look like the data clusters well by user?

Hint: You will probably want to make use the "geom-bar" function in ggplot2 to do this.

Solution 2(c)

ANSWER: No. The Barchart shows very similar as Cluster plot. Even more precisley, we can see that majority of the datapoints are spread between almost all clusters. Only cluster 10 (blue) has been separated well and its' obvious with both, Cluster plot and Barchart.

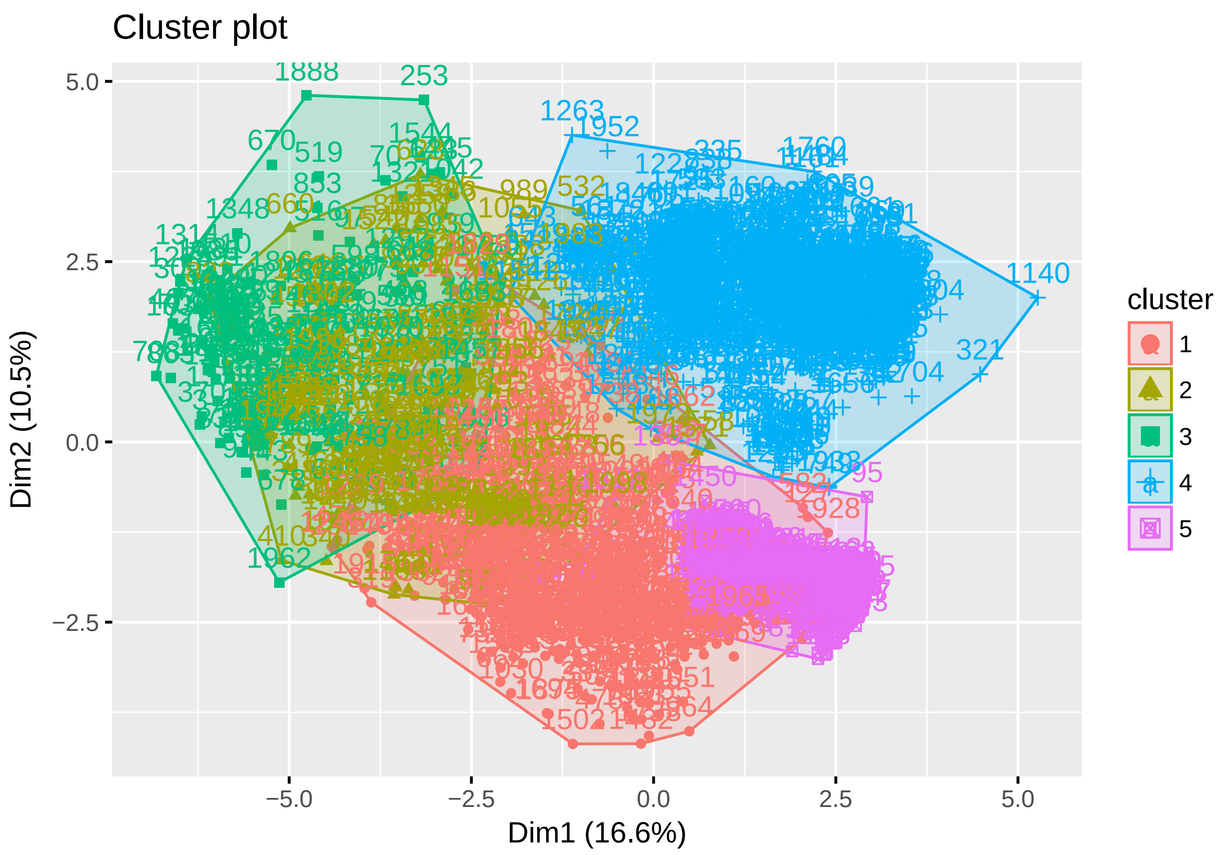
```
# compare to actual
clusters <- users_k_means$cluster
results <- data.frame(cbind(clusters, cluster_target))
g <- ggplot(results, aes(factor(clusters), fill = factor(cluster_target)))
g + geom_bar() + labs(title='Assigned Cluster vs. Actual Class', x = "Assigned Cluster", fill = "Actual
```



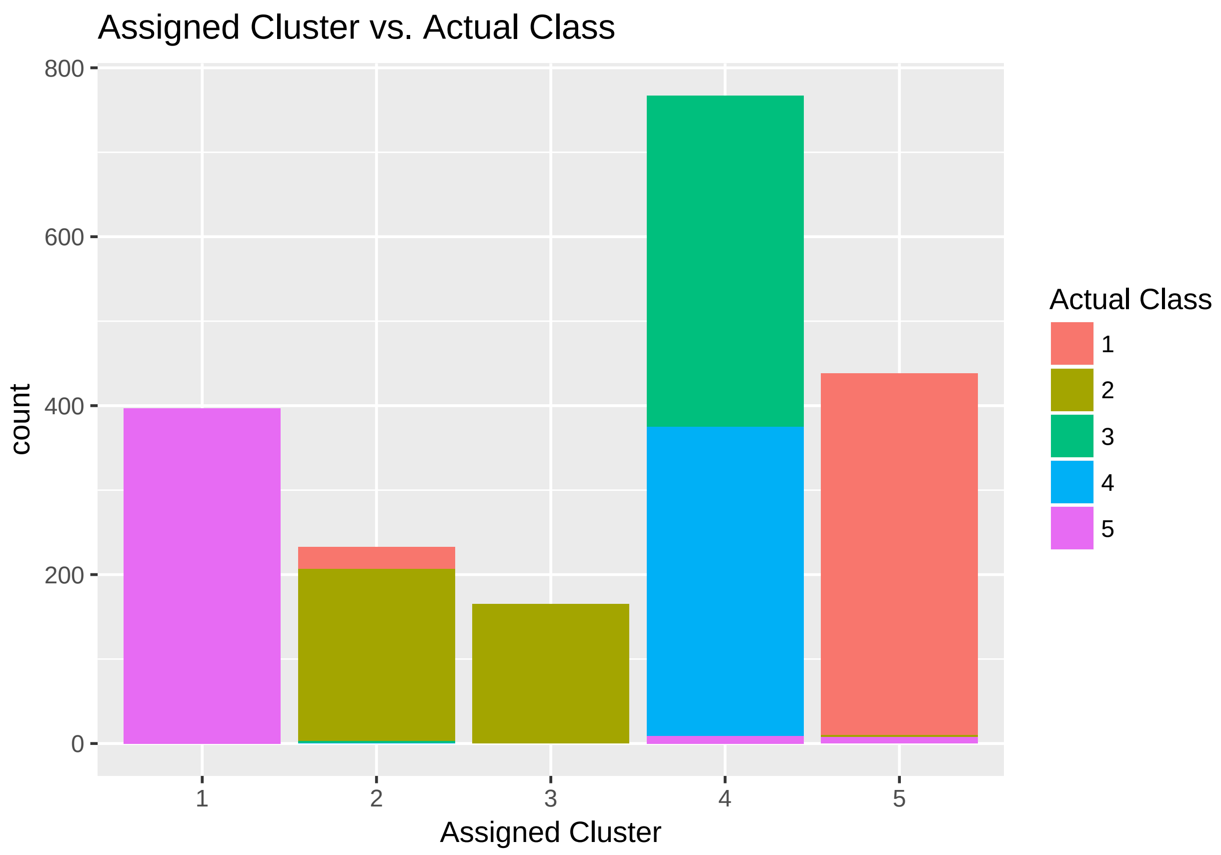
(d) Repeat all of the above steps, but group by posture rather than by user. That is, run the kmeans algorithm with 5 centroids instead of 14. Construct the same plots and answer the same questions.

Solution 2(d)

```
# cluster imputed data by posture
# split target from features
imputed_posture_target <- imputed_posture_all$Class

# scale features
imputed_posture_features <- scale(imputed_posture_all[,3:38])

# take random sample
cluster_target <- imputed_posture_target[samp]
cluster_features <- imputed_posture_features[samp,]

# run kmeans
postures_k_means = kmeans(cluster_features, 5, nstart = 46)

# plot it # probably want to press 'Zoom' in the plot window
fviz_cluster(postures_k_means, cluster_features)
```



```r
# compare to actual
clusters <- postures_k_means$cluster
results <- data.frame(cbind(clusters, cluster_target))
g <- ggplot(results, aes(factor(clusters), fill = factor(cluster_target)))
g + geom_bar() + labs(title='Assigned Cluster vs. Actual Class', x = "Assigned Cluster", fill = "Actual
```



*****

(e) What do the results of the bar plot clustered by posture suggest about the data? Why does this make sense in the context of what we know about the problem?

Solution 2(e)

ANSWER: Clusters are separated much better. However, there is a significant overlap between clusters 2 and 3. Using 5 centroids instead of 14 makes more sense in this scenario, given the nature of the problem - recognize 5 different postures, not 14 different users. *****

(f) Using all of the information gleaned from this problem, how do you recommend the missing data be imputed? Why?

Solution 2(f)

ANSWER: Taking into consideration the structure of the data, any proper justification for choosing between imputation by users or by postures is accepted. Additionally, one of the solutions might be imputing the missing data with KNN. However, two problems emerge with this technique: 1. it is computationally inefficient. 2. dealing with to many empty rows might be challenging. *****

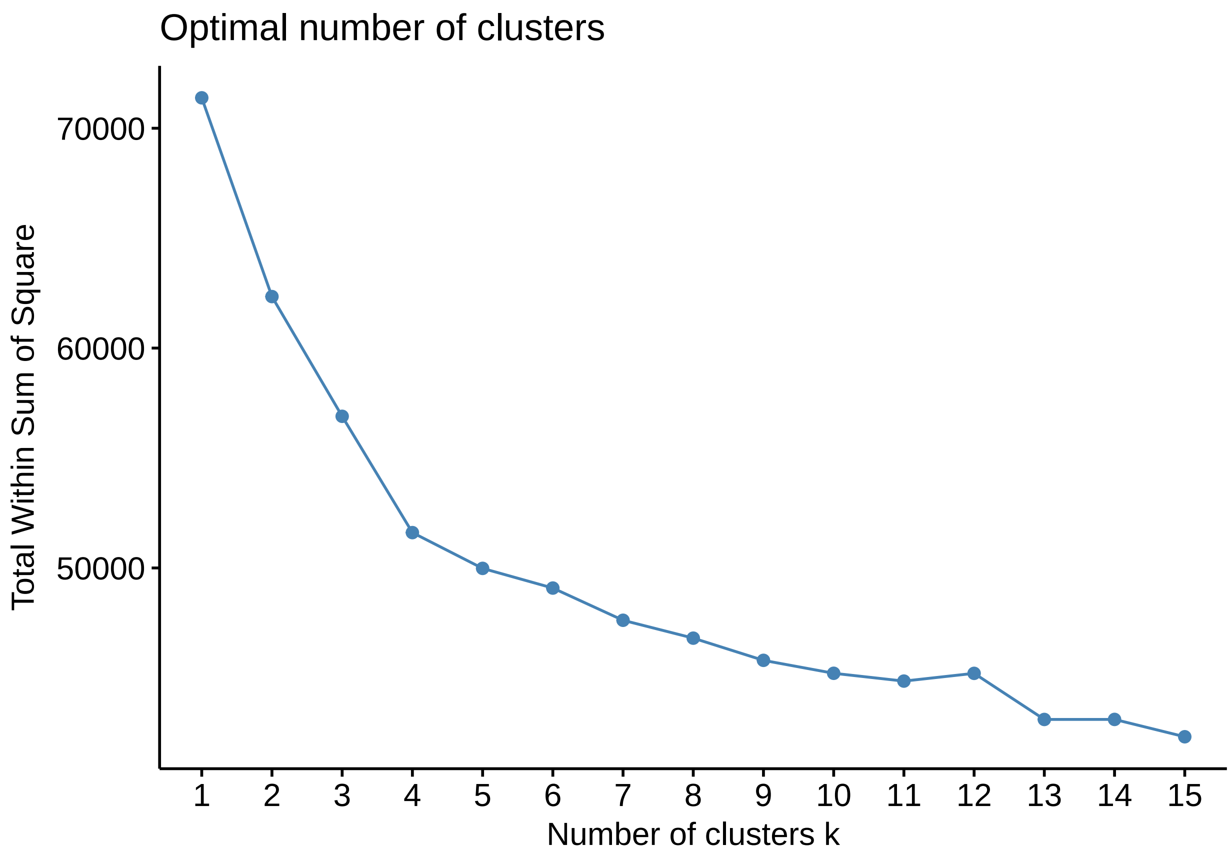
3 Clustering Evaluation

In the previous problem, we used k-means with 5 and 14 centroids to decide how we should impute missing data. In this problem, we will investigate various ways of evaluating the quality of a clustering assignment.

(a) Use the elbow method to evaluate the best choice of the number of clusters, plotting the total within-cluster variation against the number of clusters for k-means clustering with k ∈ (1, 2, ... 15).

Solution 3(a)

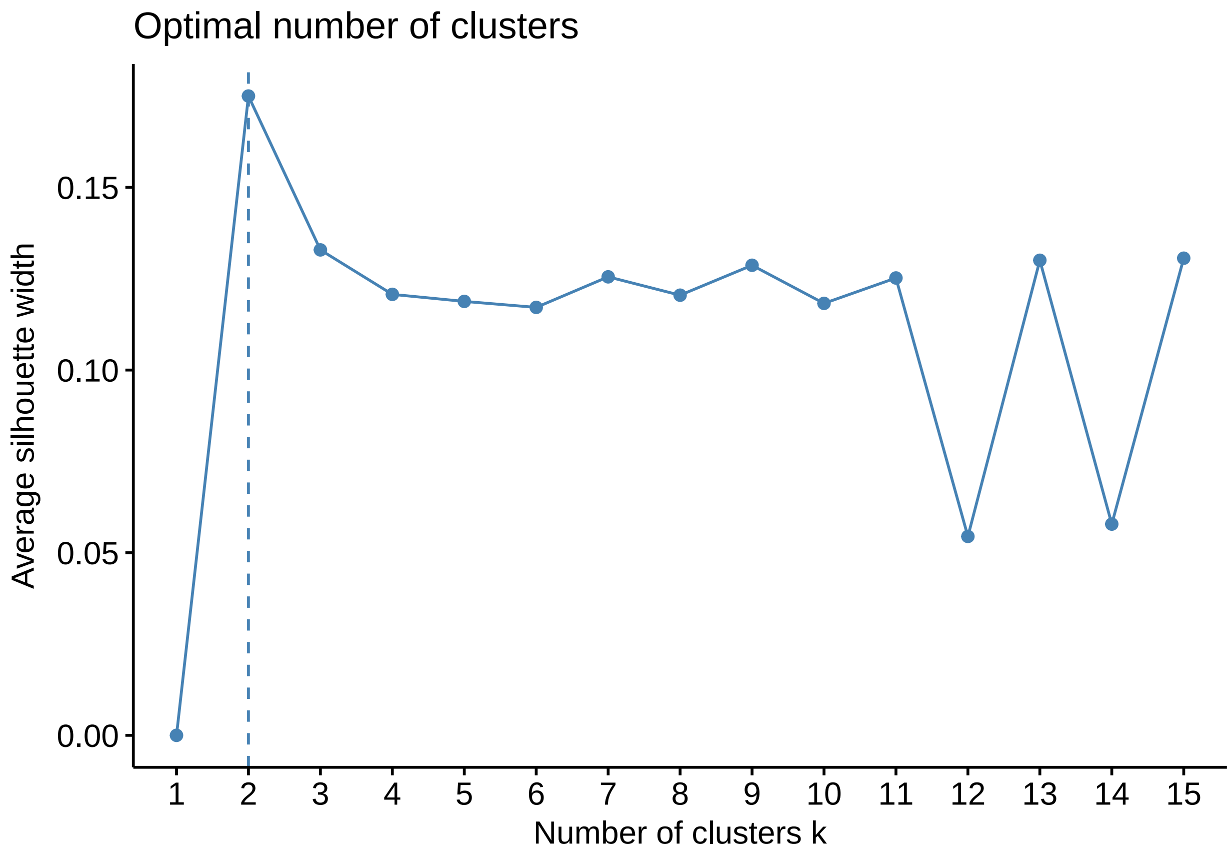
```r
# kmeans elbow plot
k_means_elbow_plot <- fviz_nbclust(cluster_features,
                                   FUNcluster=kmeans,
                                   method='wss',
                                   k.max=15)
plot(k_means_elbow_plot)
```



(b) Use the average silhouette to evaluate the choice of the number of clusters for k-means clustering with k ∈ (1, 2, ... 15). Plot the results.

Solution 3(b)

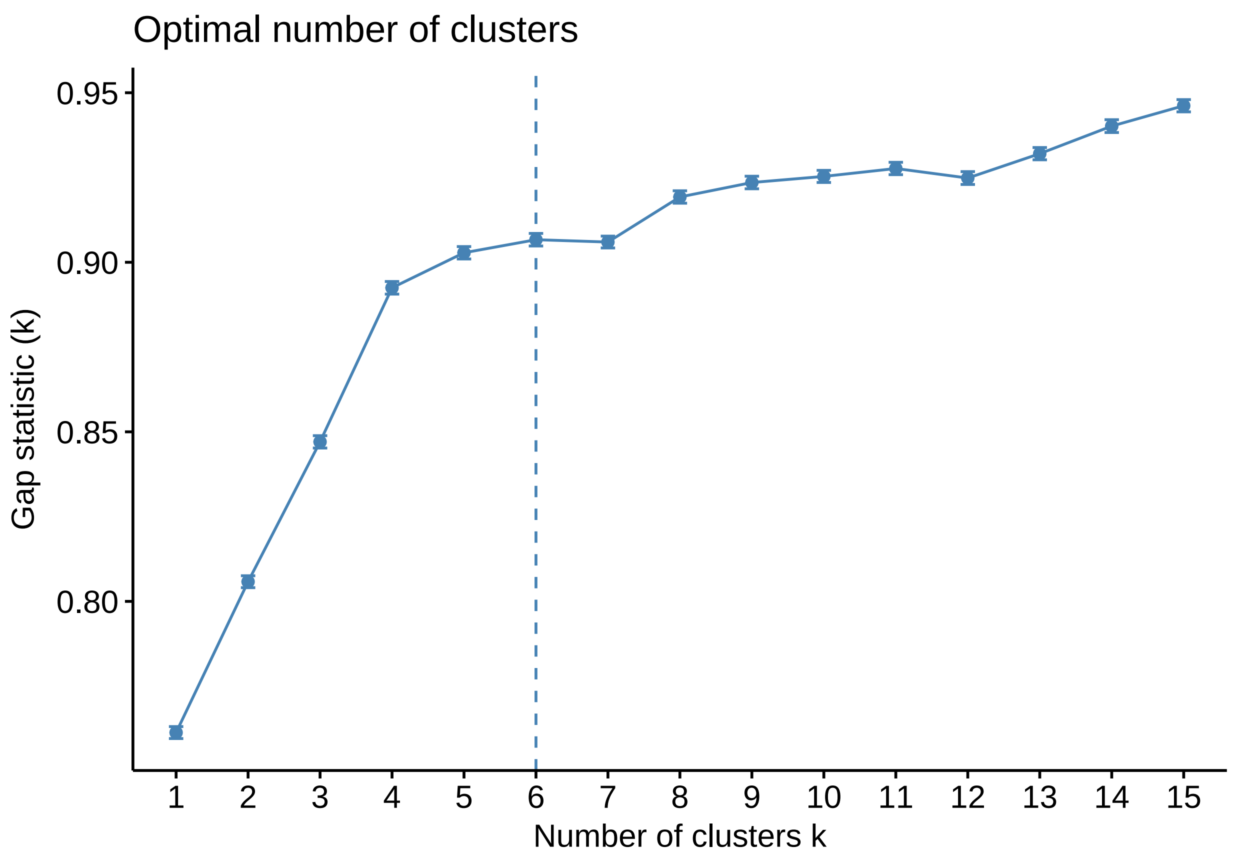
```r
# kmeans silhouette plot
k_means_sil_plot <- fviz_nbclust(cluster_features,
                                 FUNcluster=kmeans,
                                 method='silhouette',
                                 k.max=15)
plot(k_means_sil_plot)
```



(c) Use the gap statistic to evaluate the choice of the number of clusters for k-means clustering with k ∈ (1, 2, . . . 15). Plot the results. Be patient – this might take a few minutes.

Solution 3(c)

```r
# kmeans gap stat plot
k_means_gap_plot <- fviz_nbclust(cluster_features,
                                 FUNcluster=kmeans,
                                 method='gap_stat',
                                 k.max=15)
plot(k_means_gap_plot)
```



(d) After analyzing the plots produced by all three of these measures, discuss the number of clusters that you feel is the best fit for this dataset. Defend your answer with evidence from the previous parts of this assignment, the three graphs produced here, and what you surmise about this dataset.

Solution 3(d)

Based on previous analysis (from questions 2 and 3) the best number of clusters seems to be k=4. This choice of clusters is close enough to the original structure of the data (k=5). It seems that postures 3 and 5 are very similar and produce overlapping clusters. Thus, k=4 seems to be the best fit in a given scenario.

---

4. Other Clustering Algorithms

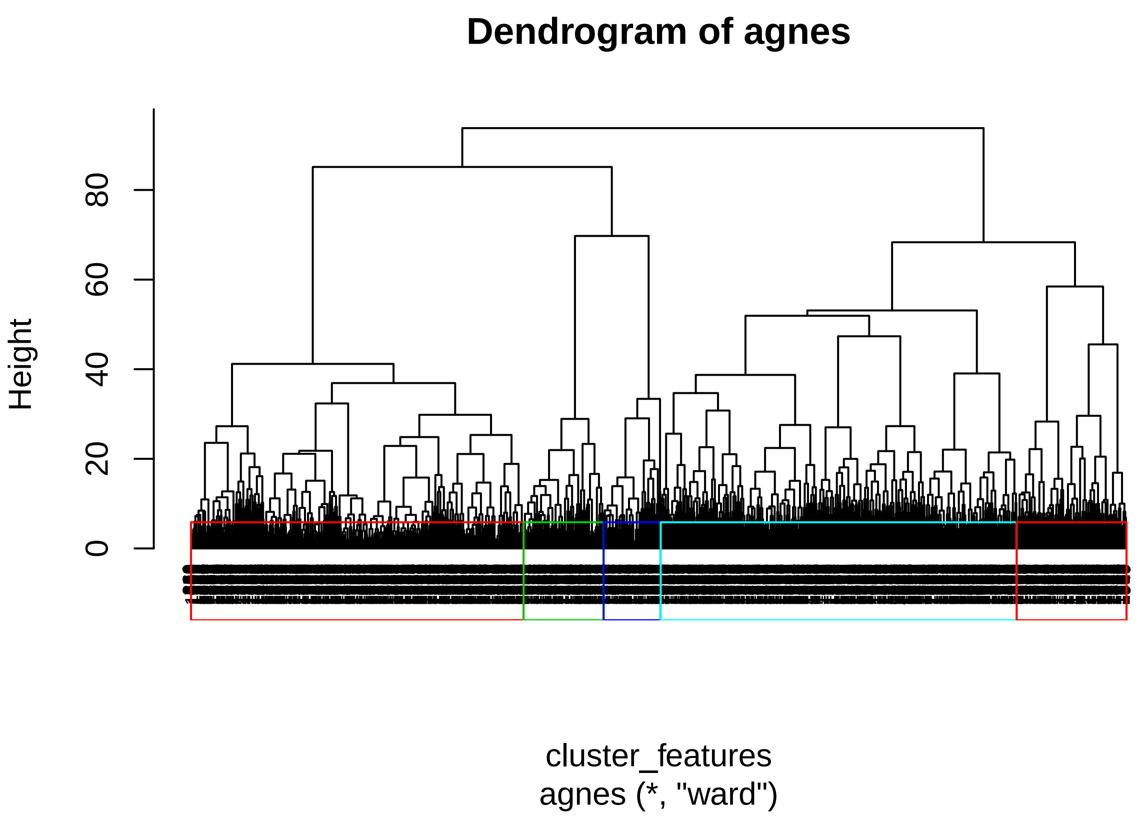
Up until now, we have used the k-means algorithm to cluster the data. In this problem, we will explore other methods used to create clusters.

(a) Hierarchical clustering: Implement agglomerative clustering (using Ward's method) and divisive clustering. Plot the results of these algorithms using a dendrogram and interpret the results. Hint: Use the "agnes" and "diana" functions, respectively.

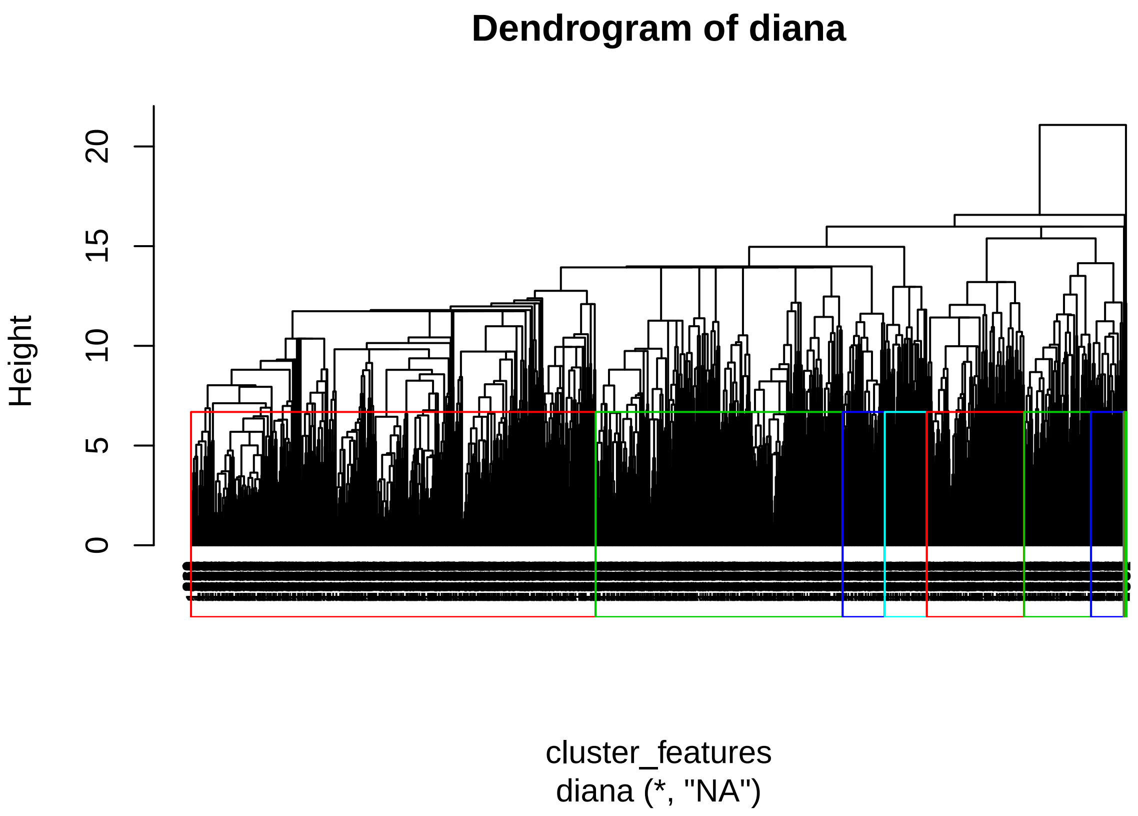
Solution 4a

```
# cluster imputed data by user
# split target from features
imputed_user_target <- imputed_user_all$User

# scale features
imputed_user_features <- scale(imputed_user_all[,3:38])

# take random sample
set.seed(42)
samp <- sample(x=1:length(imputed_user_target), size=2000, replace=F)
cluster_target <- imputed_user_target[samp]
cluster_features <- imputed_user_features[samp,]

data.scaled <- scale(cluster_features)
```

```
# agglomerative clustering (using Ward's method) call
posture.agnes = agnes(cluster_features,method="ward")
# results viz
pltree(posture.agnes, cex = 0.6, hang = -1, main = "Dendrogram of agnes")
rect.hclust(posture.agnes,k=5,border=2:5)#explicit clustering
```



```
# divisive clustering
posture.diana = diana(cluster_features)
# results viz
pltree(posture.diana, cex = 0.6, hang = -1, main = "Dendrogram of diana")
rect.hclust(posture.diana,k=10,border=2:5)#explicit clustering
```



(b) Soft clustering: Run fuzzy clustering and a Gaussian mixture model on the scaled features. For the fuzzy clustering, run the algorithm with 5 and 14 clusters and plot the results using the “fviz-silhouette” function. For the Gaussian mixture model, the “Mclust” algorithm chooses the optimal number of clusters internally; report the number of clusters it selects. Also display the membership probabilities for the first 10 observations in your sample.

Hint: Use the “fanny” and “Mclust” functions, respectively. You might need to adjust the “memb.exp” parameter to something between 1 and 2 to get the function to run correctly. Make sure to include analysis for trial-and-error of fanny parameters. Justify your results.

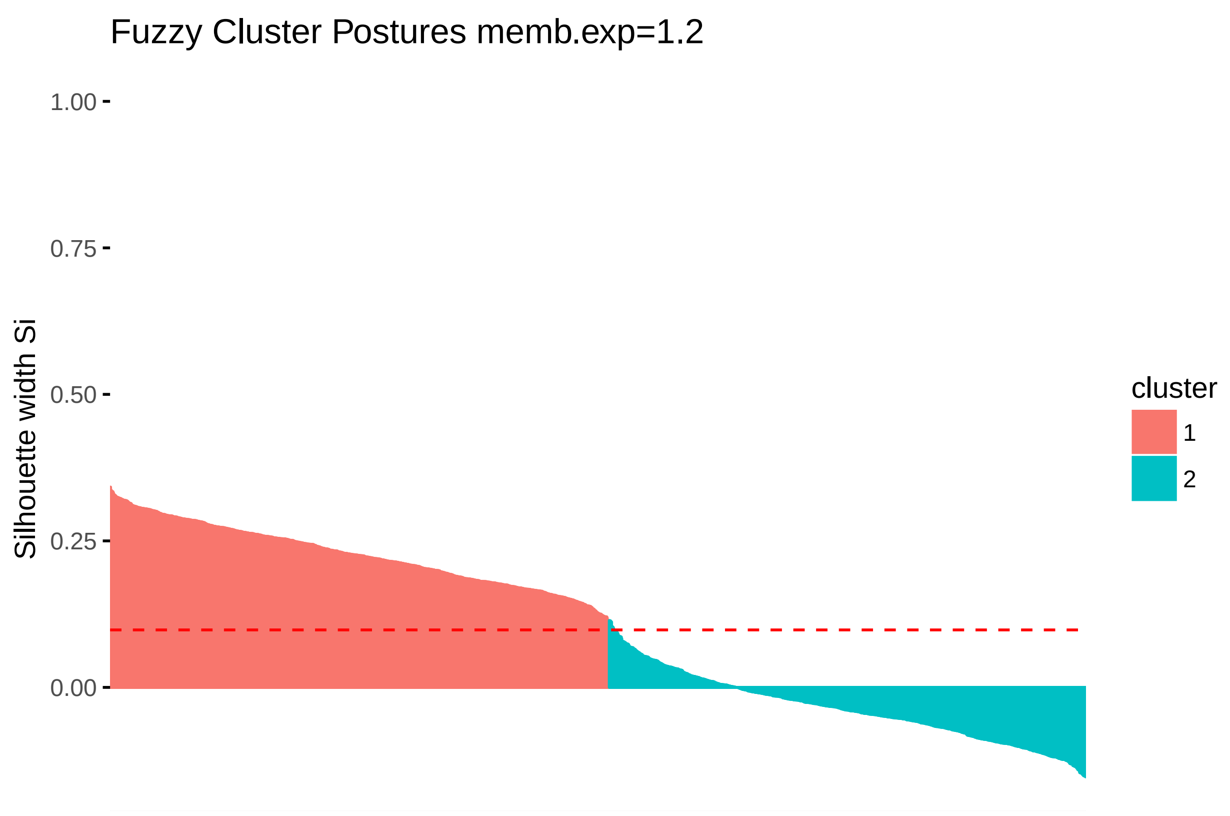
Solution 4b

```
# soft clustering
# fuzzy clustering
fuzzy_cluster_postures <- fanny(cluster_features, k=5, memb.exp=1.2)
```

```
## Warning in fanny(cluster_features, k = 5, memb.exp = 1.2): the memberships
## are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

```
fviz_silhouette(fuzzy_cluster_postures,
                main="Fuzzy Cluster Postures memb.exp=1.2")
```

```
##   cluster size ave.sil.width
## 1       1 1023          0.23
## 2       2  977         -0.04
```

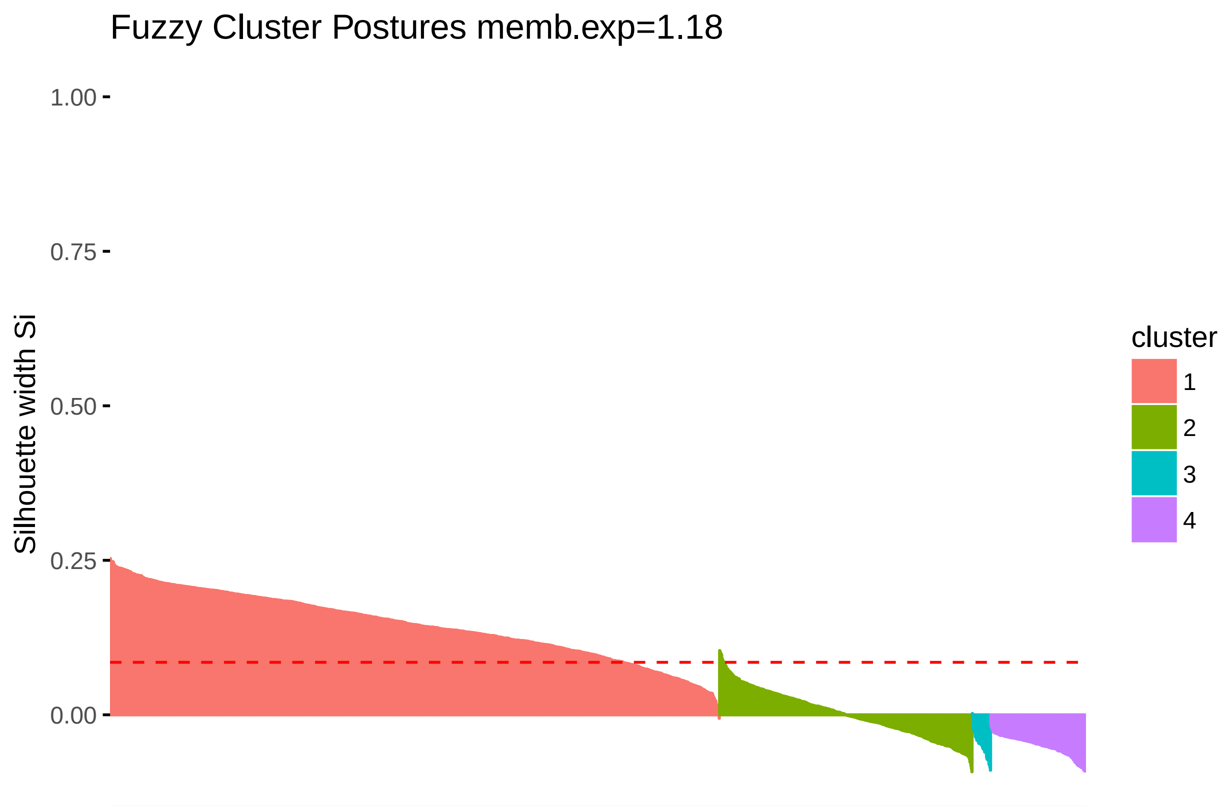


```
## decreasing memb.exp = 1.18
fuzzy_cluster_postures <- fanny(cluster_features, k=5, memb.exp=1.18)
```

```
## Warning in fanny(cluster_features, k = 5, memb.exp = 1.18): the memberships
## are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

```
fviz_silhouette(fuzzy_cluster_postures,
                main="Fuzzy Cluster Postures memb.exp=1.18")
```

```
##   cluster size ave.sil.width
## 1       1 1249          0.14
## 2       2  518          0.00
## 3       3   38         -0.05
## 4       4  195         -0.05
```

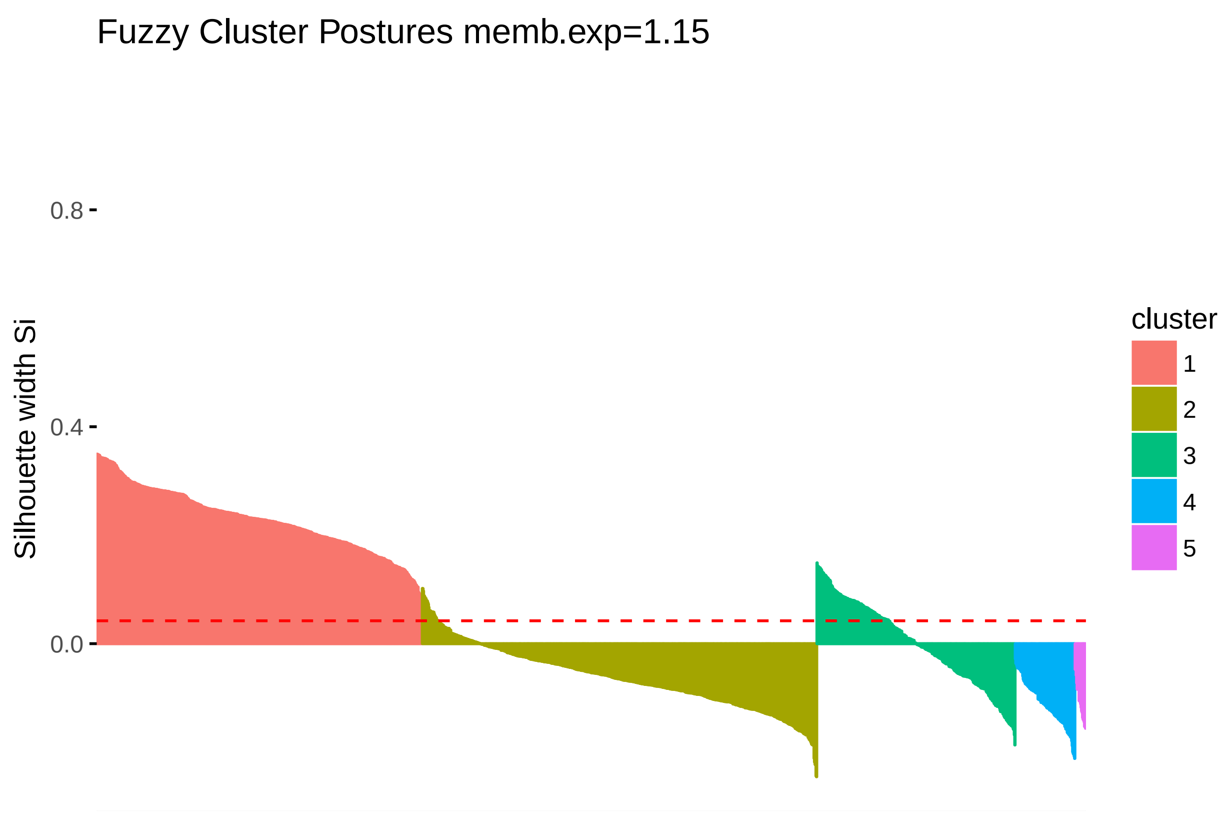


```
## decreasing memb.exp = 1.15
fuzzy_cluster_postures <- fanny(cluster_features, k=5, memb.exp=1.15)
```

```
## Warning in fanny(cluster_features, k = 5, memb.exp = 1.15): FANNY algorithm
## has not converged in 'maxit' = 500 iterations
```

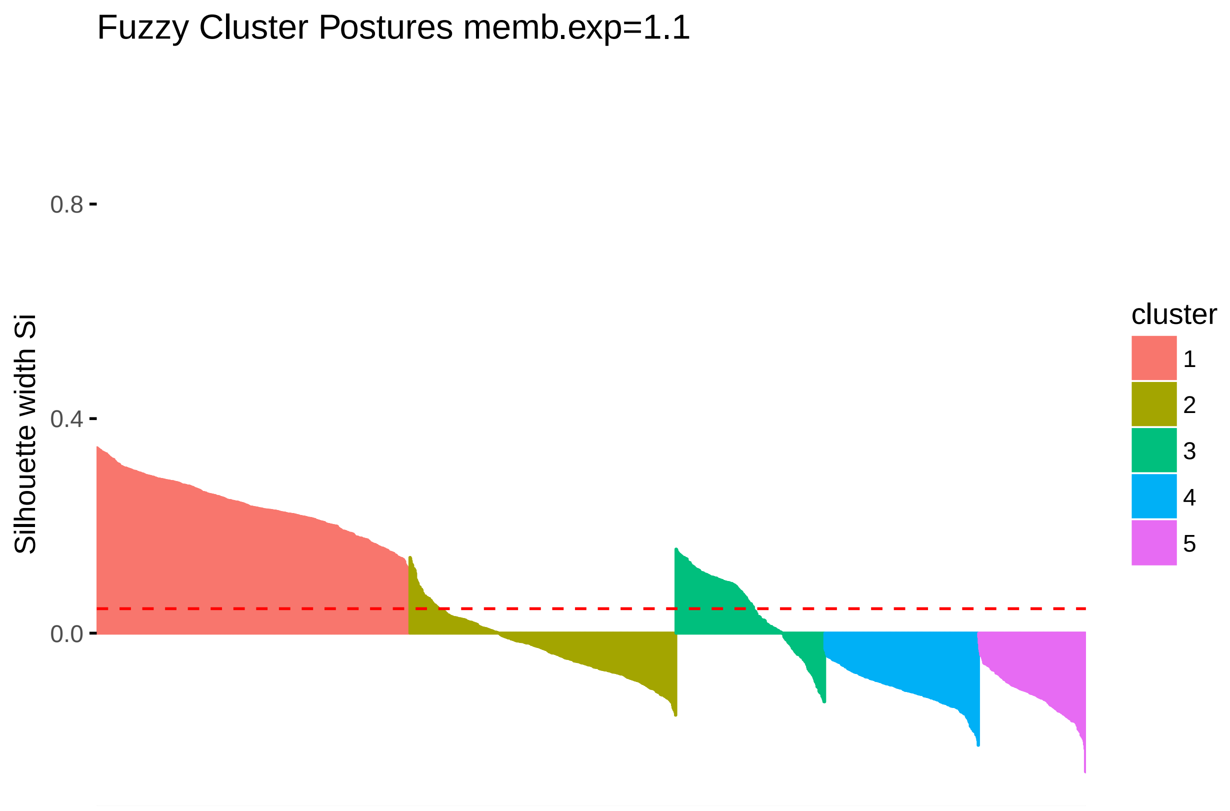
```
fviz_silhouette(fuzzy_cluster_postures,
                main="Fuzzy Cluster Postures memb.exp=1.15")
```

```
##   cluster size ave.sil.width
## 1       1  658          0.23
## 2       2  798         -0.06
## 3       3  401          0.00
## 4       4  121         -0.11
## 5       5   22         -0.11
```



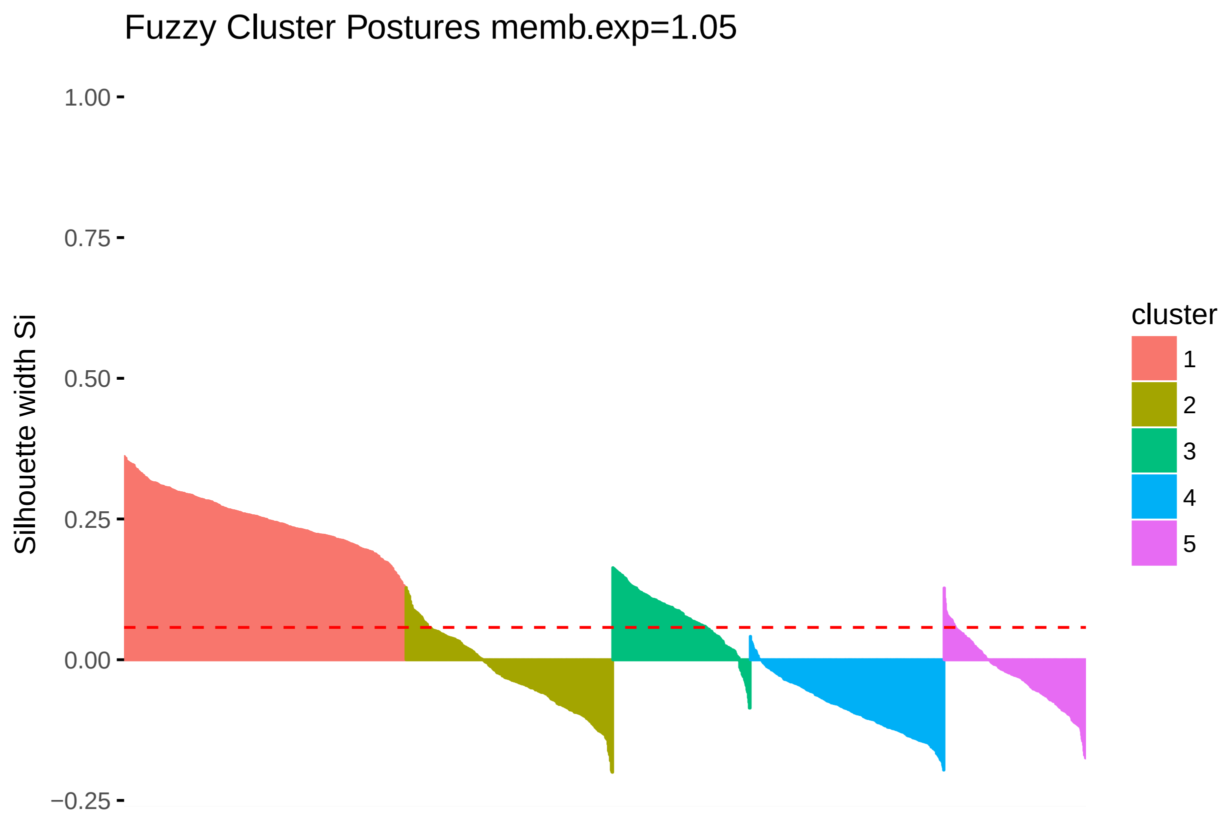
```
## decreasing memb.exp = 1.1
fuzzy_cluster_postures <- fanny(cluster_features, k=5, memb.exp=1.1)
fviz_silhouette(fuzzy_cluster_postures,
                main="Fuzzy Cluster Postures memb.exp=1.1")
```

```
##   cluster size ave.sil.width
## 1       1  633          0.24
## 2       2  538         -0.03
## 3       3  301          0.04
## 4       4  311         -0.10
## 5       5  217         -0.12
```



```
## decreasing memb.exp = 1.05
fuzzy_cluster_postures <- fanny(cluster_features, k=5, memb.exp=1.05)
fviz_silhouette(fuzzy_cluster_postures,
                main="Fuzzy Cluster Postures memb.exp=1.05")
```

```
##   cluster size ave.sil.width
## 1       1  586          0.25
## 2       2  430         -0.03
## 3       3  286          0.08
## 4       4  403         -0.08
## 5       5  295         -0.03
```

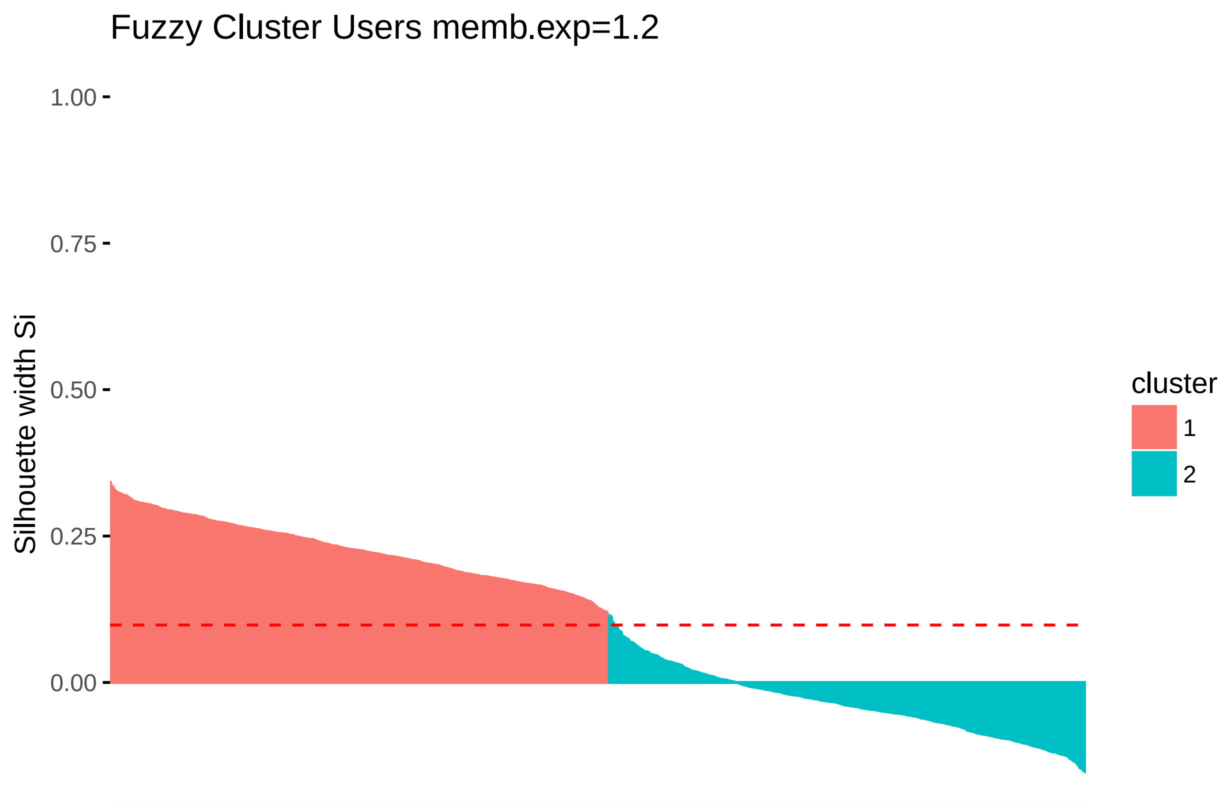


```
fuzzy_cluster_users <- fanny(cluster_features, k=14, memb.exp=1.2)
```

```
## Warning in fanny(cluster_features, k = 14, memb.exp = 1.2): the memberships
## are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

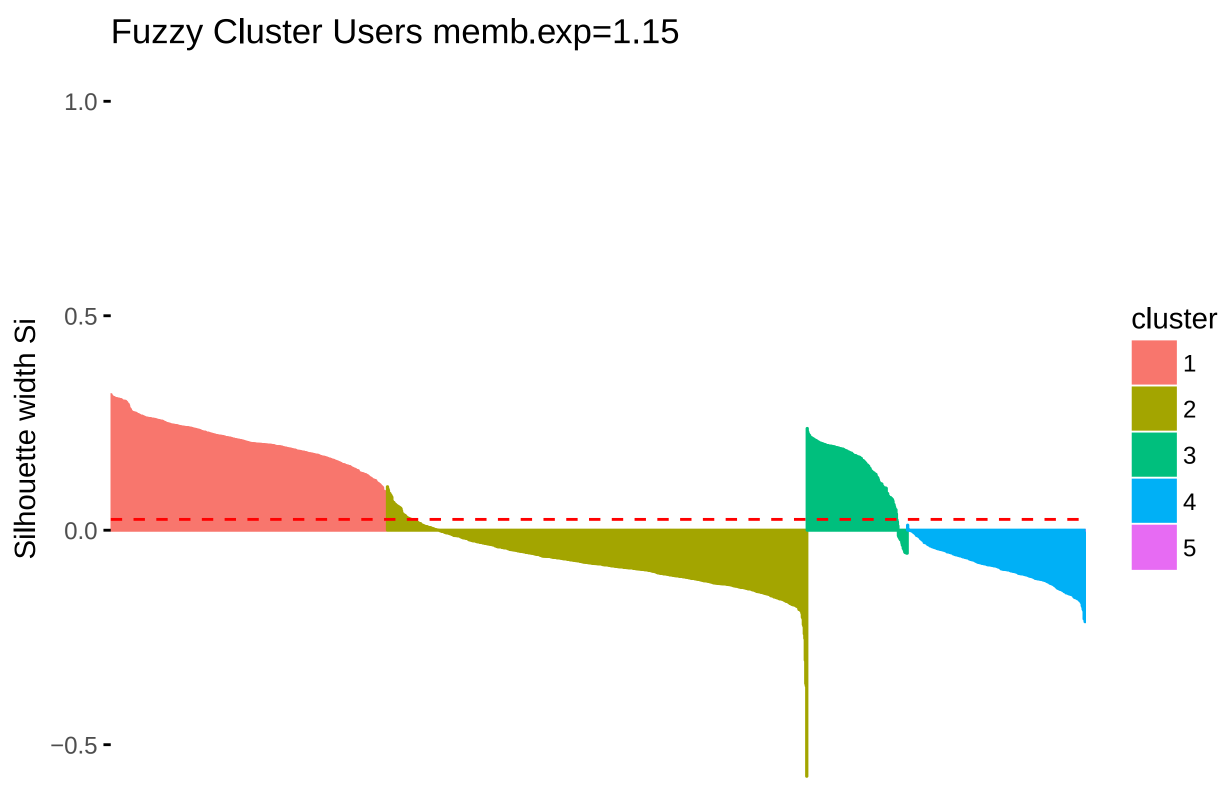
```
fviz_silhouette(fuzzy_cluster_users,
                main="Fuzzy Cluster Users memb.exp=1.2")
```

```
##   cluster size ave.sil.width
## 1       1 1023          0.23
## 2       2  977         -0.04
```



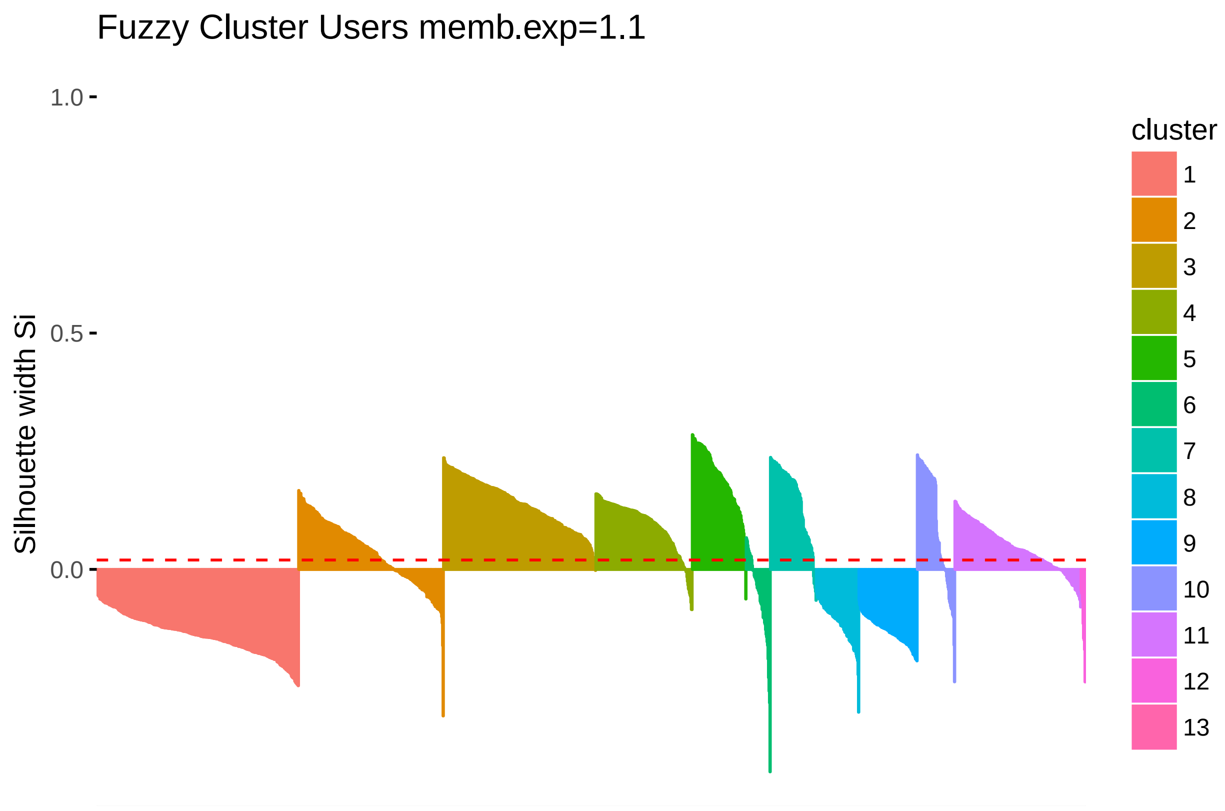
```
## decreasing memb.exp = 1.15
fuzzy_cluster_users <- fanny(cluster_features, k=14, memb.exp=1.15)
fviz_silhouette(fuzzy_cluster_users,
                main="Fuzzy Cluster Users memb.exp=1.15")
```

```
##   cluster size ave.sil.width
## 1       1  567          0.21
## 2       2  861         -0.07
## 3       3  206          0.14
## 4       4  365         -0.09
## 5       5    1          0.00
```



```
## decreasing memb.exp = 1.1
fuzzy_cluster_users <- fanny(cluster_features, k=14, memb.exp=1.1)
fviz_silhouette(fuzzy_cluster_users,
                main="Fuzzy Cluster Users memb.exp=1.1")
```

```
##    cluster size ave.sil.width
## 1        1  408         -0.14
## 2        2  293          0.04
## 3        3  308          0.14
## 4        4  195          0.09
## 5        5  109          0.20
## 6        6   49         -0.06
## 7        7   93          0.15
## 8        8   86         -0.12
## 9        9  118         -0.13
## 10      10   76          0.10
## 11      11  255          0.05
## 12      12    9         -0.11
## 13      13    1          0.00
```



```r
# Gaussian clustering
gaussian_cluster <- Mclust(cluster_features)
summary(gaussian_cluster)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust EEV (ellipsoidal, equal volume and shape) model with 6 components:
##
##  log.likelihood    n   df       BIC       ICL
##       -59100.19 2000 4037 -148885.2 -148886.4
##
## Clustering table:
##   1   2   3   4   5   6
## 627 108 791 241 120 113
```

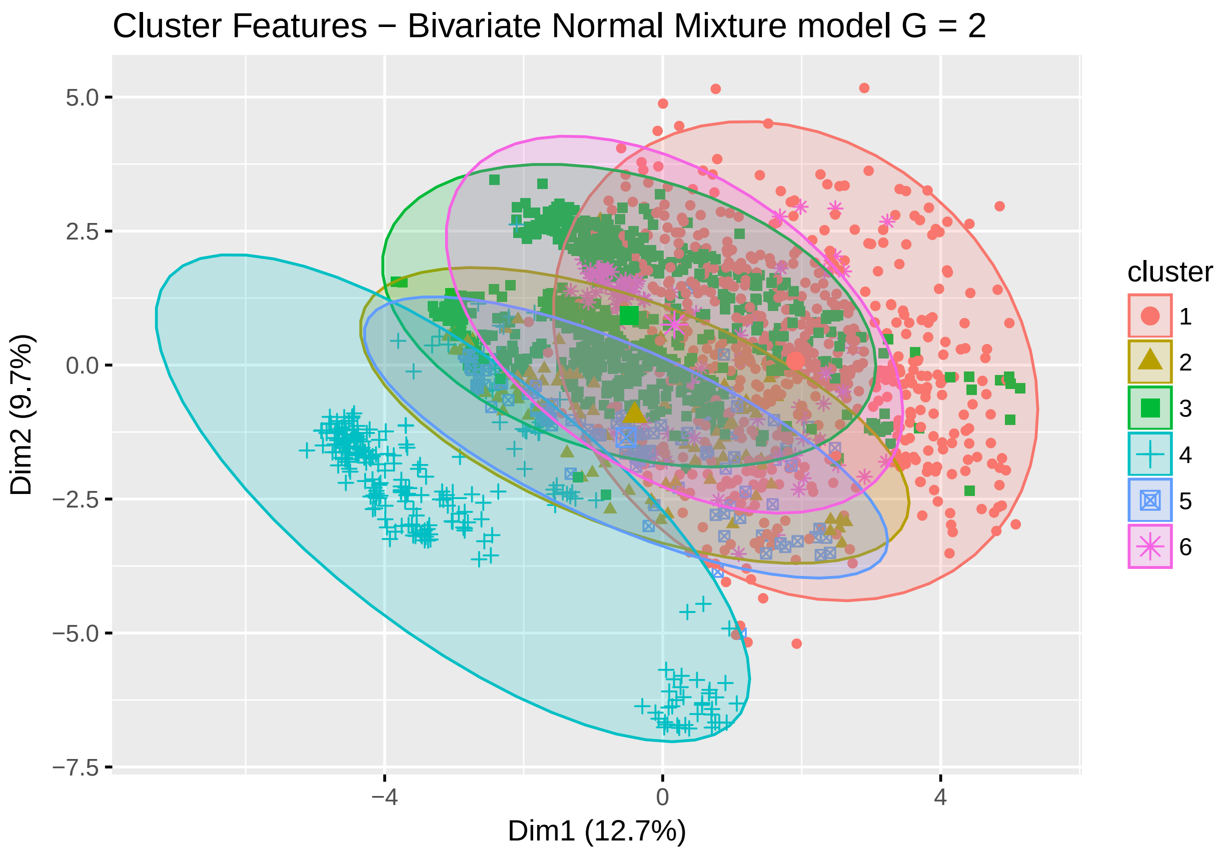
```r
#optimal number of clusters
print(gaussian_cluster$G)
```

```
## [1] 6
```

```r
#estimated probability for an obs to be in each cluster
head(gaussian_cluster$z)
```

```
##               [,1] [,2]         [,3]          [,4] [,5]         [,6]
## [1,]  1.318933e-09    0 1.000000e+00 8.998501e-154    0 0.000000e+00
## [2,]  1.000000e+00    0 9.558886e-43 5.075561e-181    0 0.000000e+00
## [3,]  1.000000e+00    0 3.034623e-08 6.412672e-199    0 0.000000e+00
## [4,]  1.000000e+00    0 1.028028e-26 1.094223e-143    0 0.000000e+00
## [5,]  9.092383e-11    0 1.000000e+00  2.840645e-78    0 2.272792e-136
## [6,] 1.807512e-116    0 3.390254e-36  1.000000e+00    0  0.000000e+00
```

```
fviz_cluster(gaussian_cluster, ellipse.type="norm", geom = "point") +
  ggtitle("Cluster Features - Bivariate Normal Mixture model G = 2")
```



This clustering algorithm already scales the data.

(c) (AC 209b students only) Density-based clustering: Apply DBSCAN to the data. Determine a reasonable combination of ϵ, the radius of the neighborhood around an observation, and the number of nearest neighbors within the ϵ-neighborhood to be considered a core point. You should construct a knee plot to determine the choice of ϵ. Summarize the results using a principal components plot, and comment on the clusters and outliers identified. How does the clustering produced by DBSCAN compare to the previous methods? Read Section 2.2 of the R vignette on DBSCAN

https://cran.r-project.org/web/packages/dbscan/vignettes/dbscan.pdf to learn about the OPTICS algorithm.

1. Describe the difference in goal between the DBSCAN and OPTICS algorithm. You may need to refer to the references cited within.

Solution 4c.1

In the case of DBSCAN, and unlike previously studied clustering algorithms, DBSCAN can find any shape of clusters; identify observations that do not belong to clusters as outliers, and can be used for predicting cluster membership for new data. Even though it doesnot require specifying the number of clusters—like in the case of hierarchical clustering—the procedure still requires to input parameters that are hard to determine and have influencial effects in the clustering results.

Furthermore, for many real-datasets there does not even exist a global parameter setting for which the result of the clustering algorithm describes the intrinsic clustering structure accurately.

It is in this clustering analysis that the OPTICS algorithm—Ordering Points To Identify the Clustering Structure, with attribution to Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander— has benefits. It doesnot produce a clustering of the data explicitly, instead creates an augmented ordering of the database representing its density-based clustering structure. This cluster-ordering contains information which is equivalent to the density-based clusterings corresponding to a broad range of parameter settings.

The key idea of **density-based clustering** is that for each object of a cluster the neighborhood of a given radius— ϵ — has to contain at least a minimum number of object— *MinPts* —ie. the cardinality of the neighborhood has to exceed the threshold. To introduce the notion of **density-based clustering-order**, OPTICS pivots on the observation that for a constant *MinPts-value*, density-based clusters with respect to a higher density (a lower value of ϵ) are completely contained in density-connected sets with respect to a lower density (a higher value of ϵ). Thus, OPTICS extends DBSCAN s.t. several distance parameters are processed at the same time—density-based clusters with respect to different densities are constructed simultaneously. The algorithm produces a consistent result by obeying a specific object processing-order when expanding a cluster, hence, always an object with rechable density with respect to the lowest ϵ must be selected to guarantee that clusters with repects to higher densities are finsihed first.

Moreover, the advantage of cluster-ordering a dataset compared to other clustering methods is that the reachability-plot (produced via OPTICS) is rather insensitive to input parameters used in the methodology. Hence, for the OPTICS algorithm, the values of ϵ (eps) and *MinPts* need be just *large enough* to yield a good result, as the concrete values are not crucial because there's a broad range of possible values for which we always can see the clustering structure of the dataset when looking at a reachability-plot. This is seen in the answers below.

OPTICS is similar to DBSCAN, however, for OPTICS eps is only an upper limit for the neighborhood size used to reduce computational complexity. OPTICS linearly orders the data points such that points which are spatially closest become neighbors in the ordering. The closest analog to this ordering is dendrogram in single-link hierarchical clustering. The algorithm also calculates the reachability distance for each point. The reachability-plot shows each points reachability distance where the points are sorted by OPTICS. Valleys represent clusters (the deeper the valley, the more dense the cluster) and high points indicate points between clusters. If xi is specified, then the algorithm to extract clusters hiearchically specified in Ankers et al (1999), with an added set of fixes to the algorithm originally contributed by the ELKI framework, is used. Internally, optics calls opticsXi to extract the resulting hierarchical representation of what several DBSCANs would produce at varying density thresholds. It has been noted that one interpretation of the xi parameter is that it classifies clusters by change in relative cluster density.

---

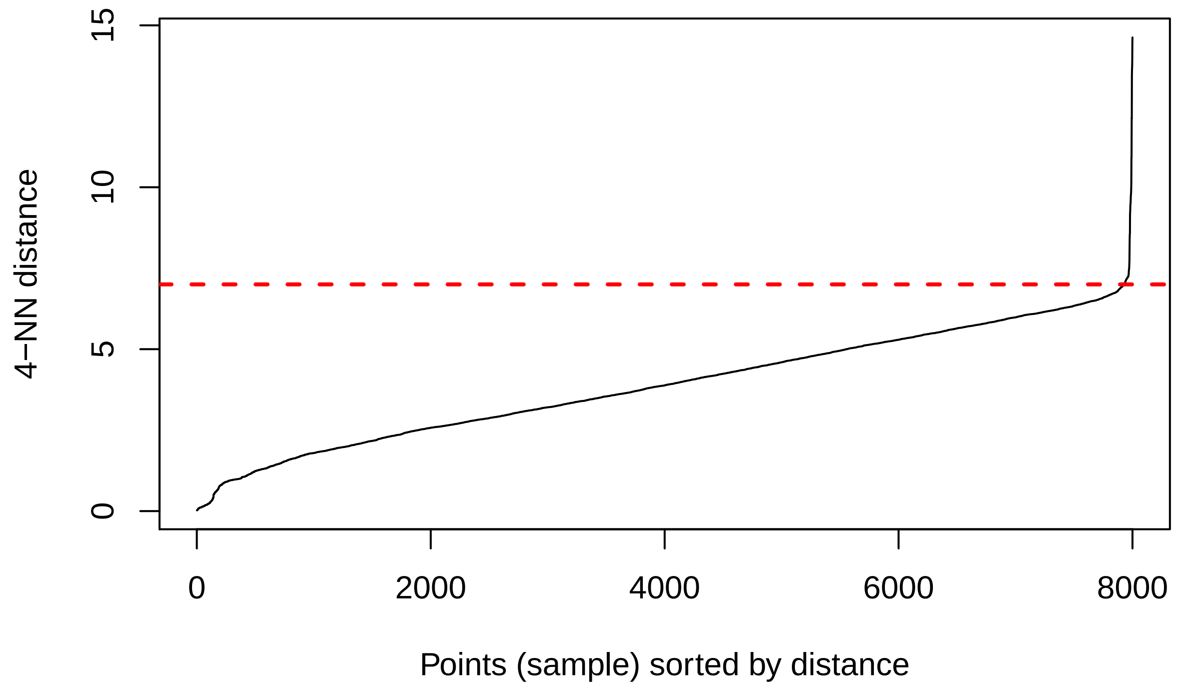
2. Run the OPTICS algorithm on the data within the dbscan package. Choose (and justify) an appropriate value of ϵ and the minimum number of points in the ϵ-neighborhood. Interpret the results of the clustering.

Hint: Make sure to also use and plot `extractXi()` with parameter `xi=0.5` to properly visualize your results. See documentation suggested above.

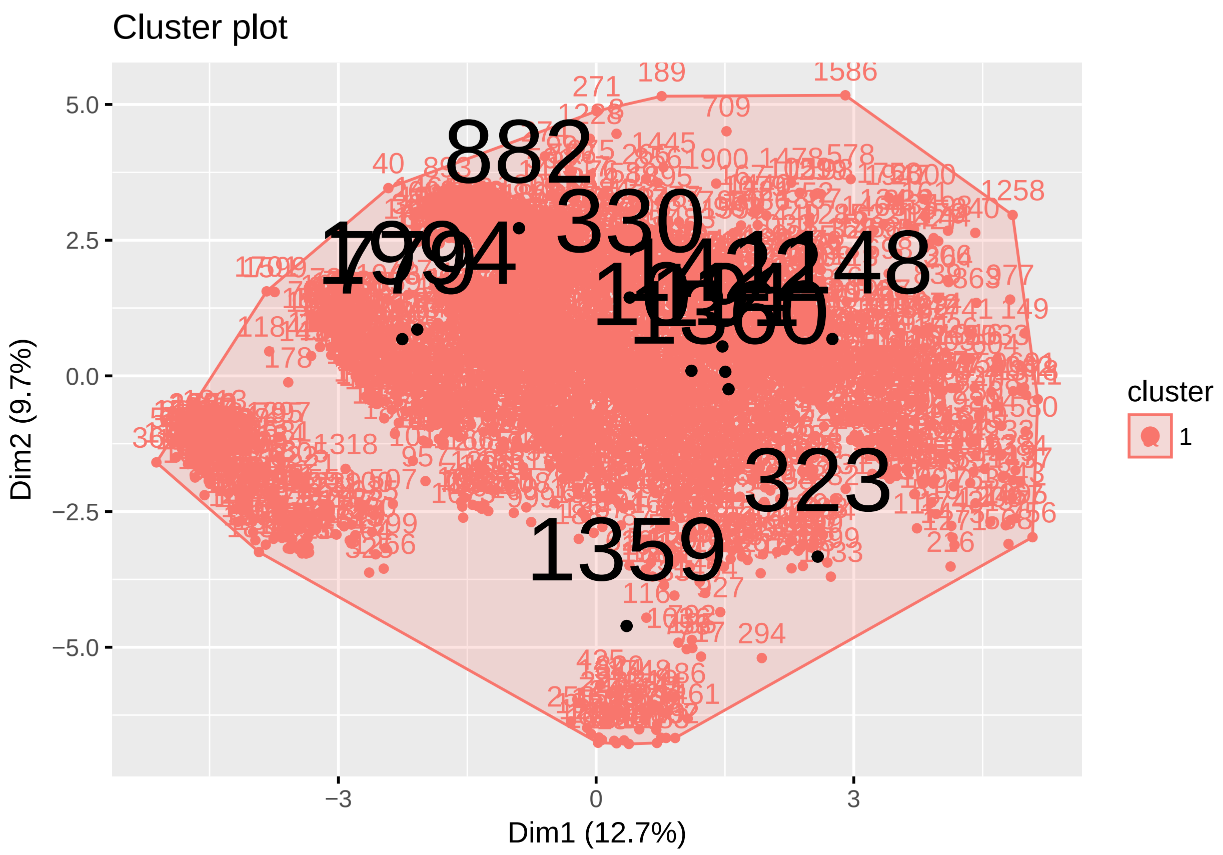
Solution 4c.2

Neighborhoodd tolerance discovery

```
# density-based clustering
kNNdistplot(cluster_features)
abline(7,0,lty=2,lwd=2,col="red") # added after seeing kNN plot
```

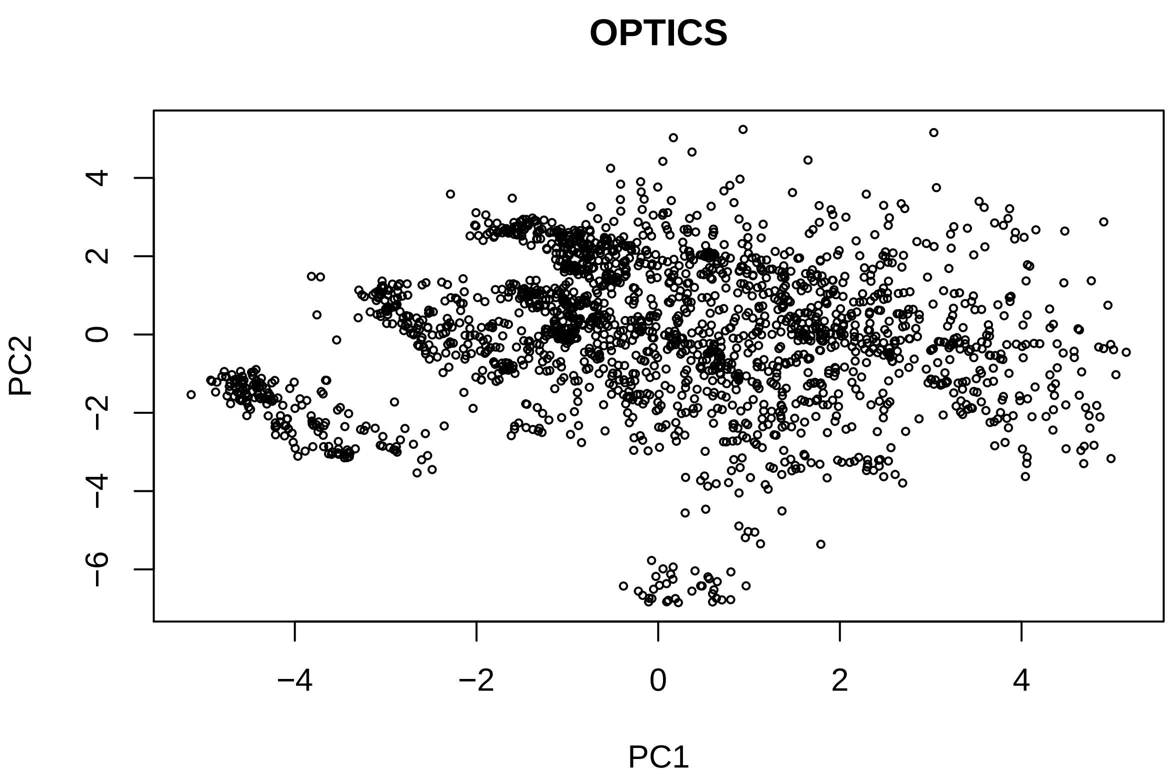


```
knee <- 7 ## eps at 7 or 100 give the same clustering only regulation of MinPts
## higlights some different structure.
db_cluster <- dbscan(cluster_features, minPts=10, eps=knee)
fviz_cluster(db_cluster, cluster_features)
```

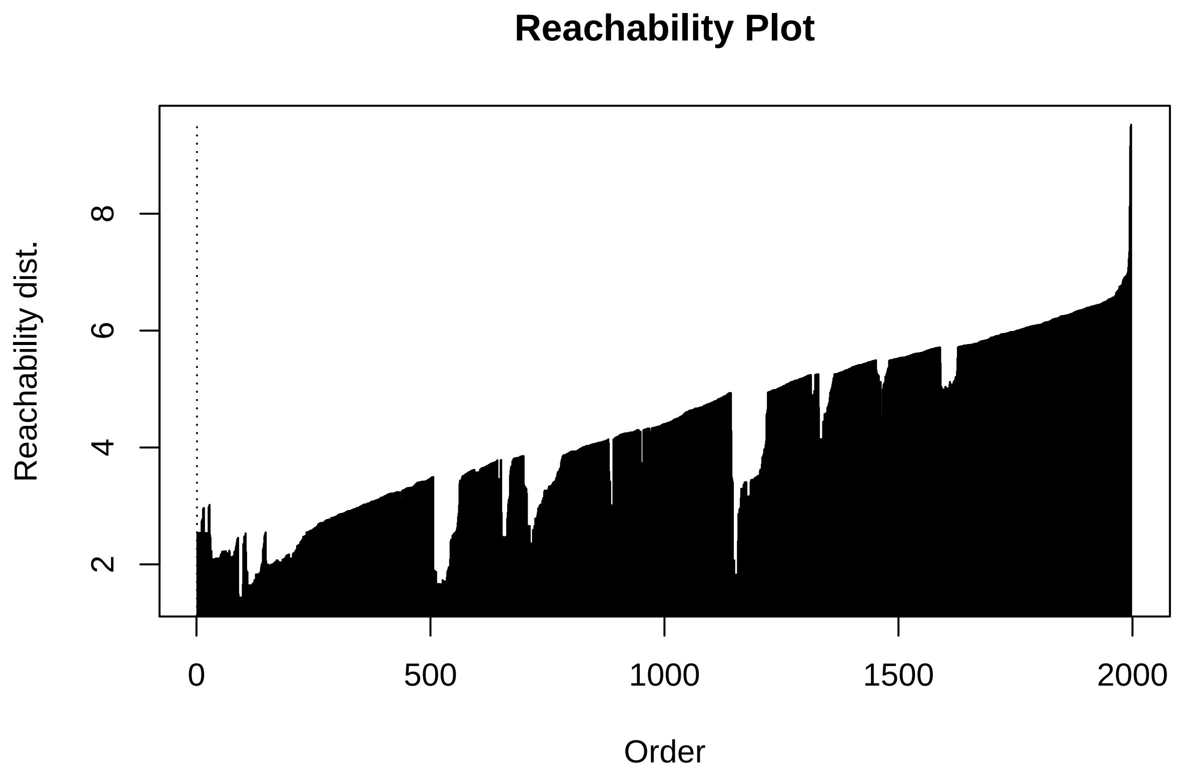


Larger than knee eps, `minPts=10`

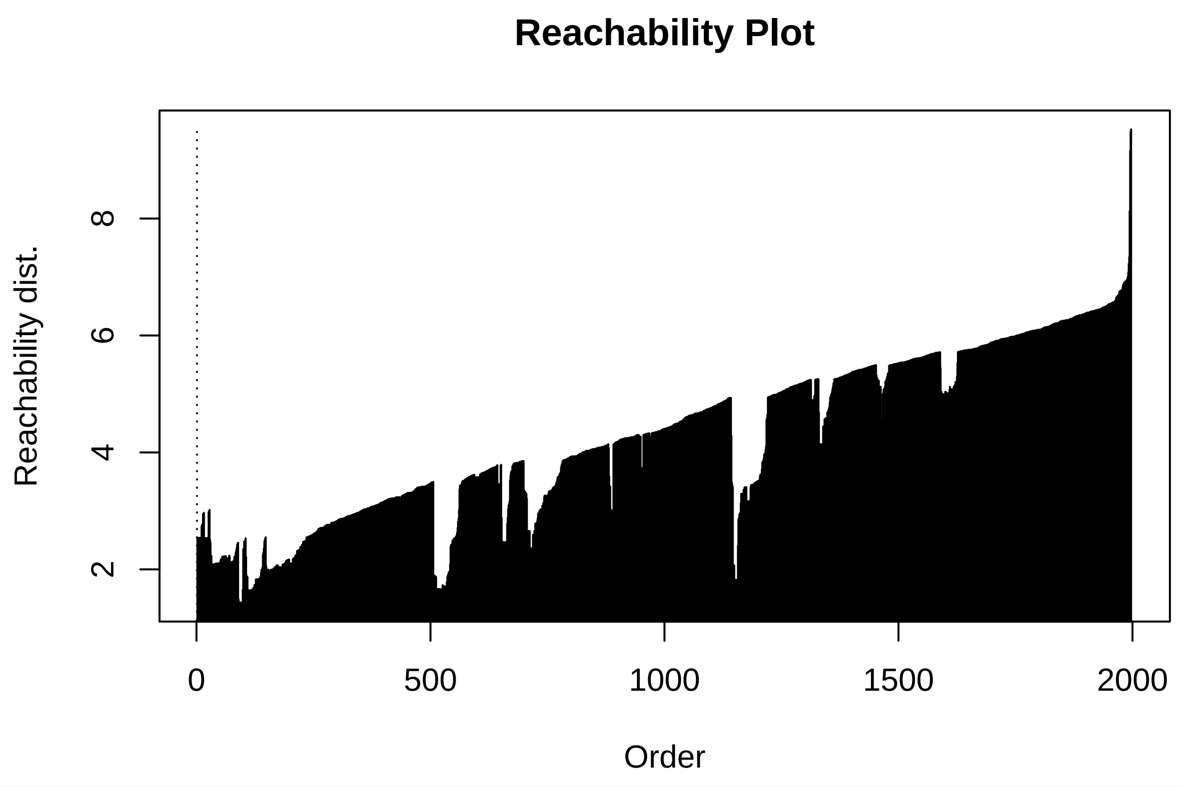
```
op <- optics(cluster_features, eps = 10, minPts = 10)
opDBSCAN <- extractDBSCAN(op, eps_cl = .07)
hullplot(cluster_features, opDBSCAN, main = "OPTICS")
```



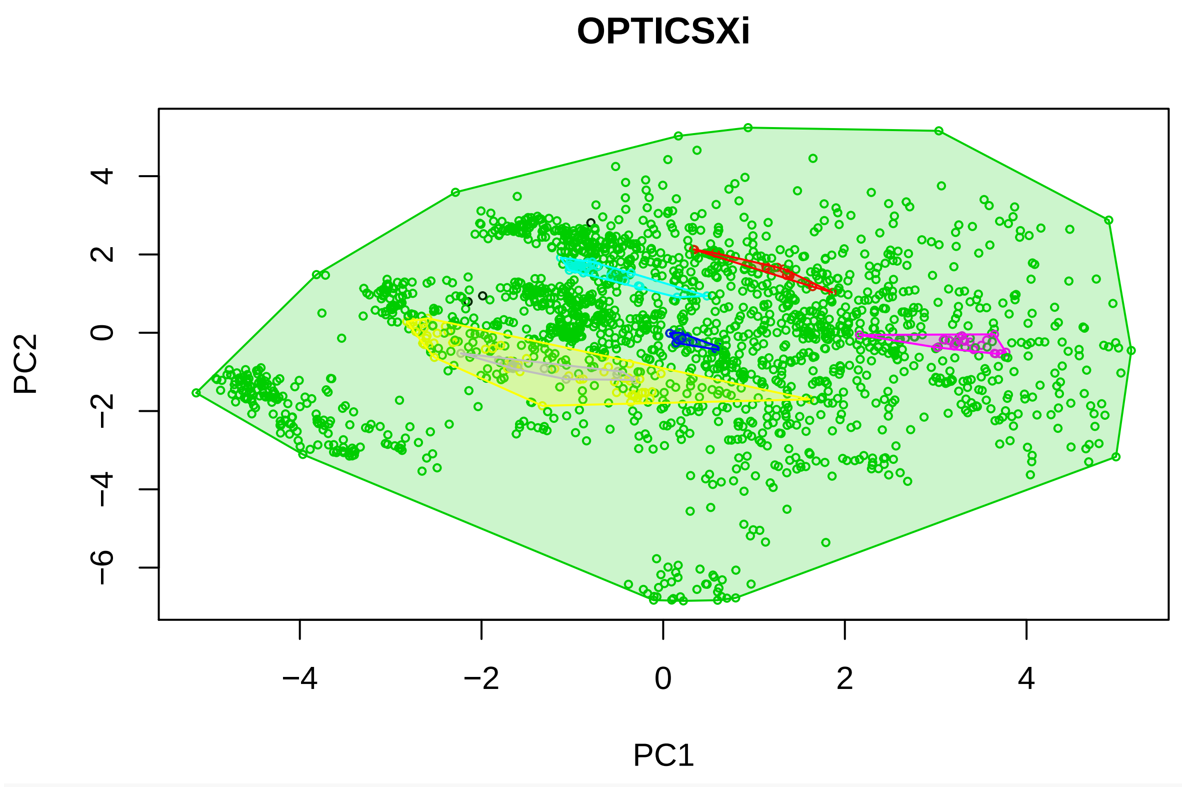
```
plot(op)
```



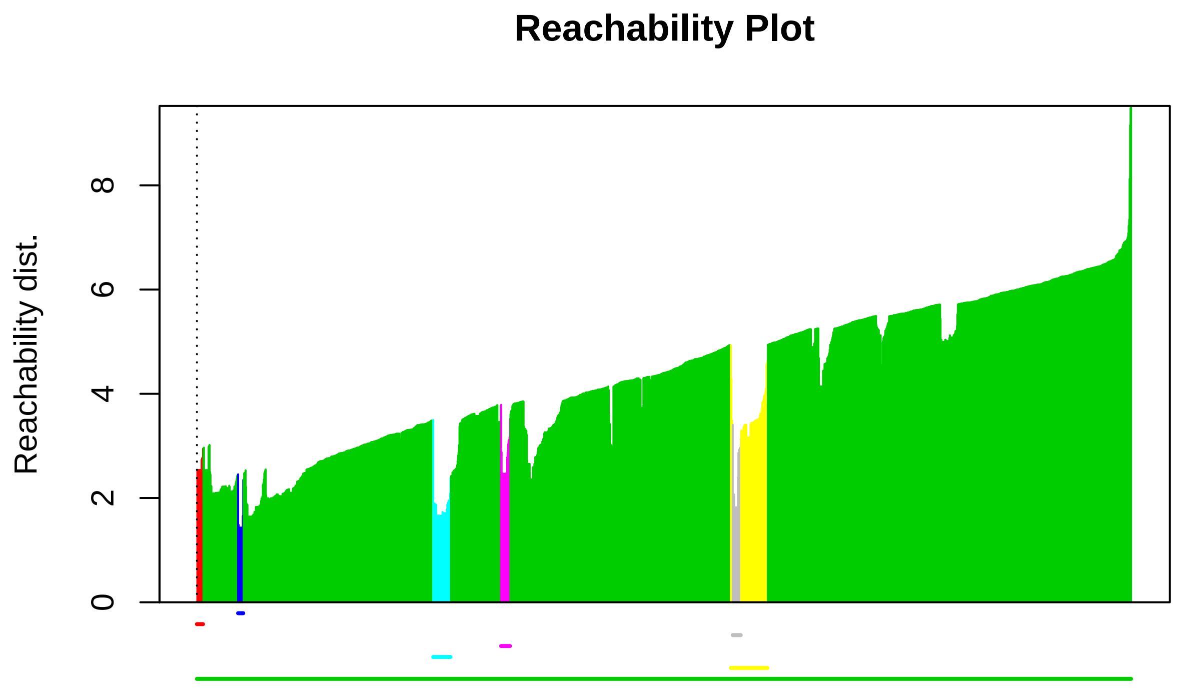
```
plot(opDBSCAN)
```



```
opXi <- extractXi(op, xi = 0.05)
hullplot(cluster_features, opXi, main = "OPTICSXi")
```

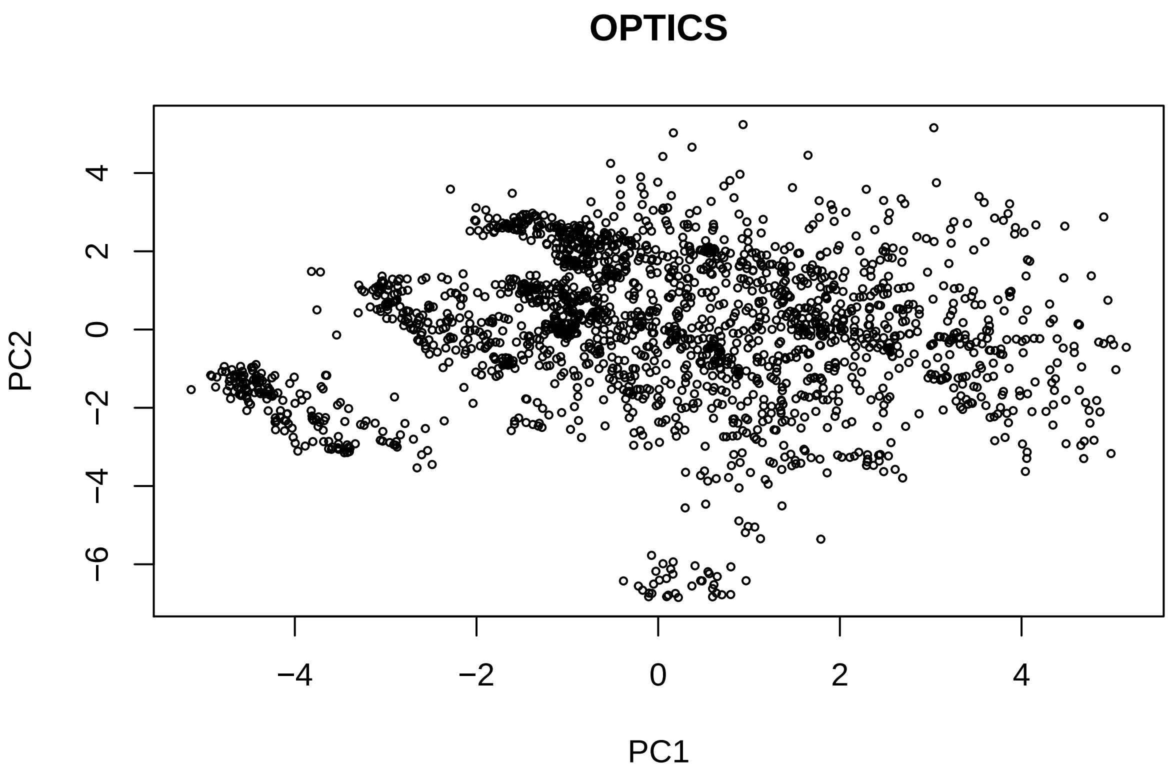


```
plot(opXi)
```

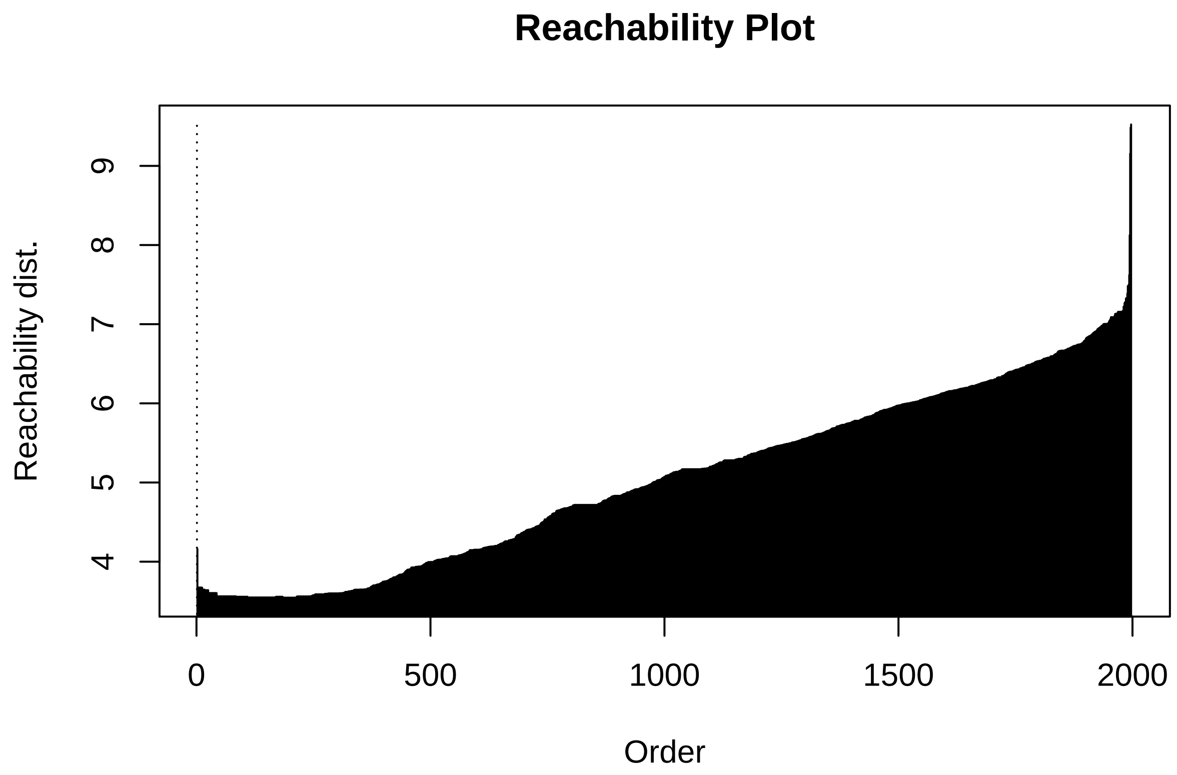


Larger than knee eps, `minPts=100`

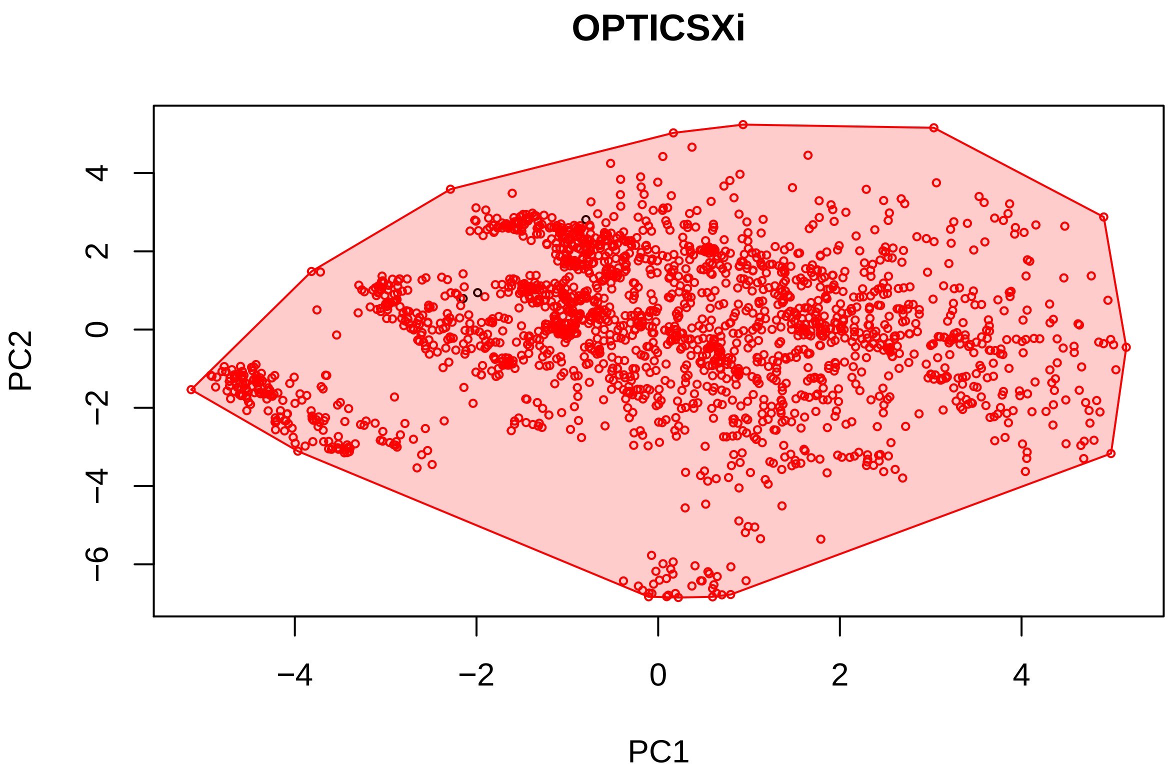
```r
op <- optics(cluster_features, eps = 10, minPts = 100)
opDBSCAN <- extractDBSCAN(op, eps_cl = .07)
hullplot(cluster_features, opDBSCAN, main = "OPTICS")
```



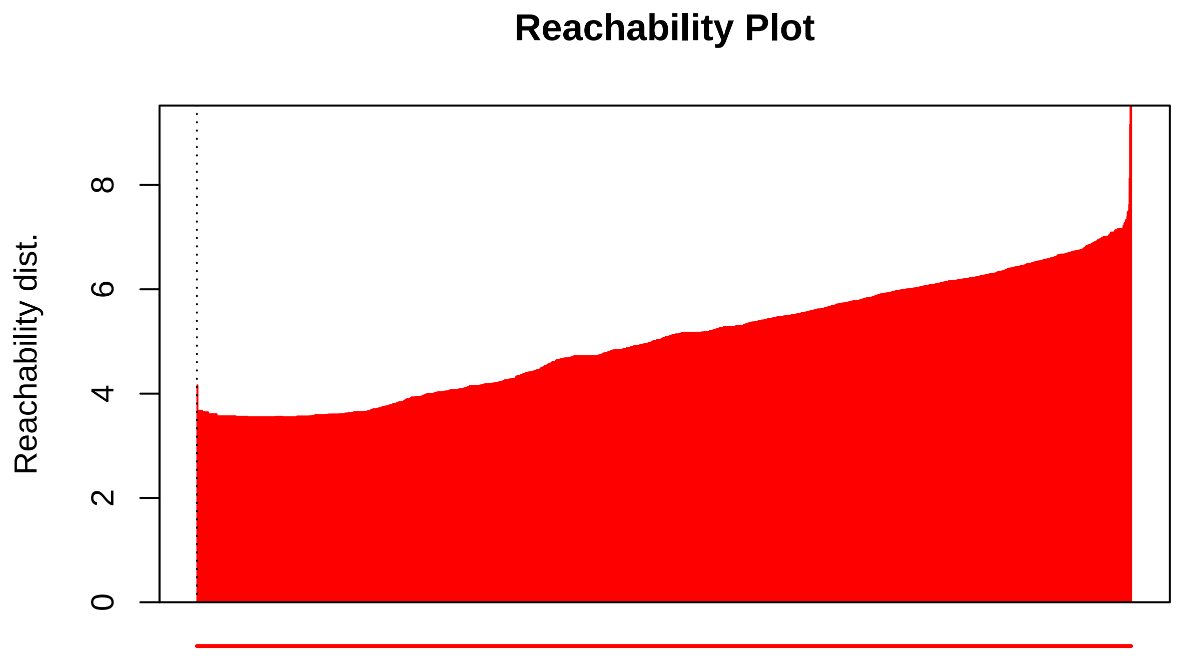
```r
plot(op)
```



```r
opXi <- extractXi(op, xi = 0.05)
hullplot(cluster_features, opXi, main = "OPTICSXi")
```



```
plot(opXi)
```

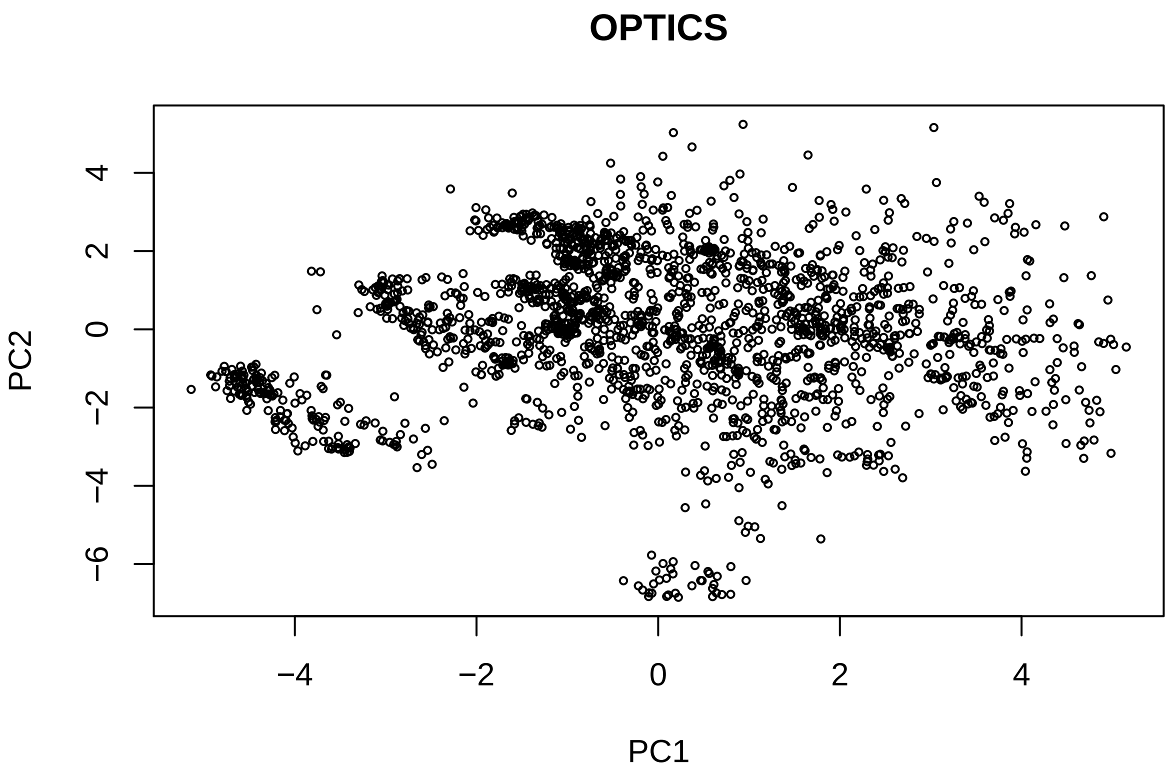


>

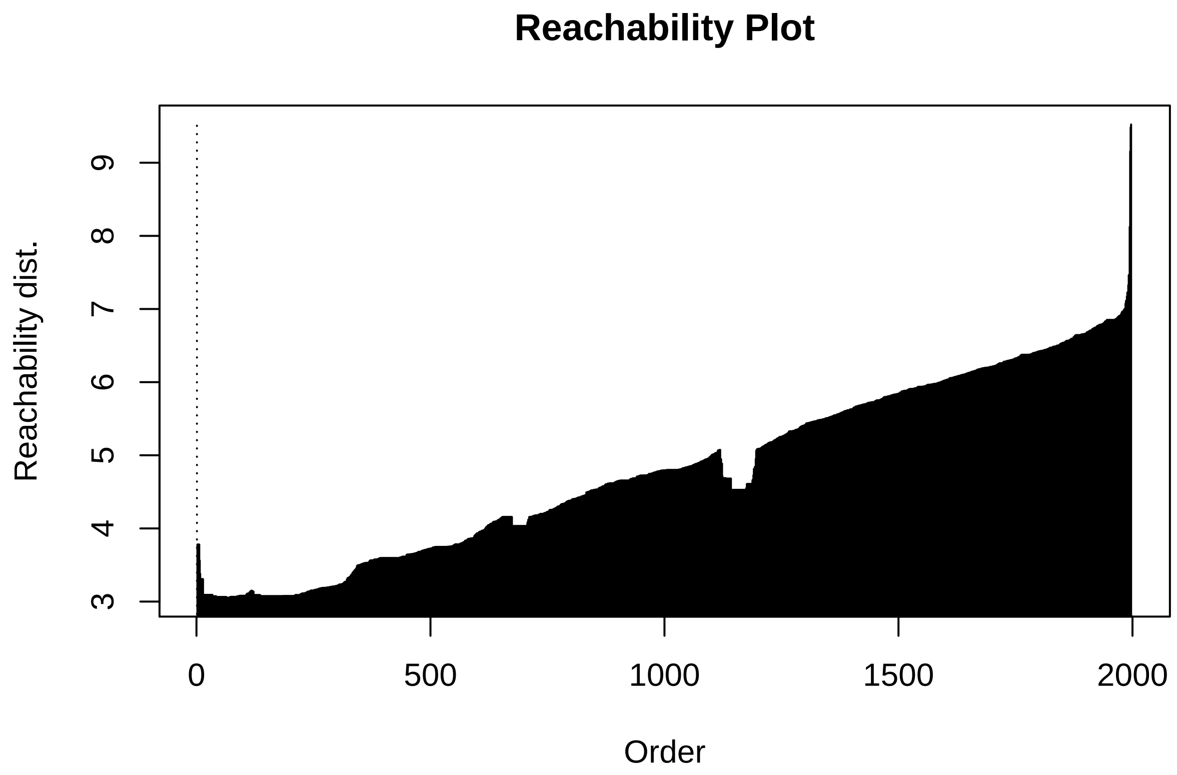
Clearly, the combination effect of a slightly larger ϵ and a very large constraint of minPts for cluster, is an inadequate approach as we barely see any signs of cluster-structure in the reachability plot (above)

Larger than knee eps, `minPts=50`

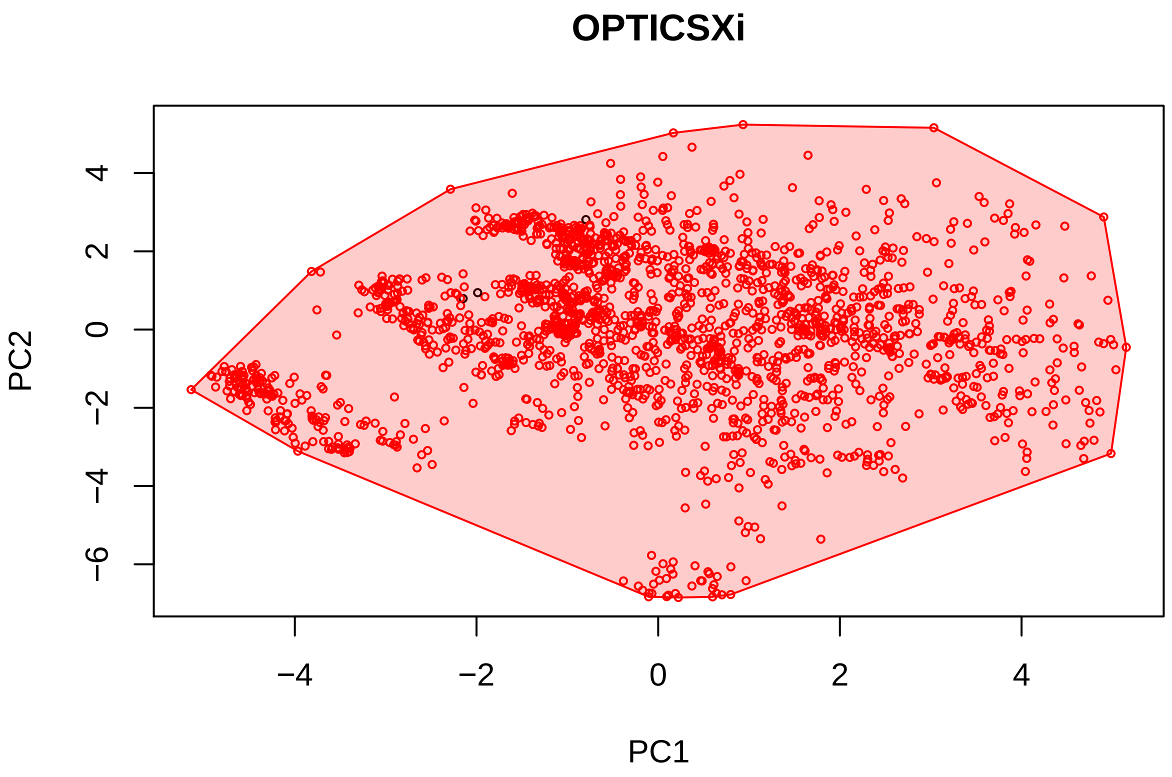
```
op <- optics(cluster_features, eps = 10, minPts = 50)
opDBSCAN <- extractDBSCAN(op, eps_cl = .07)
hullplot(cluster_features, opDBSCAN, main = "OPTICS")
```



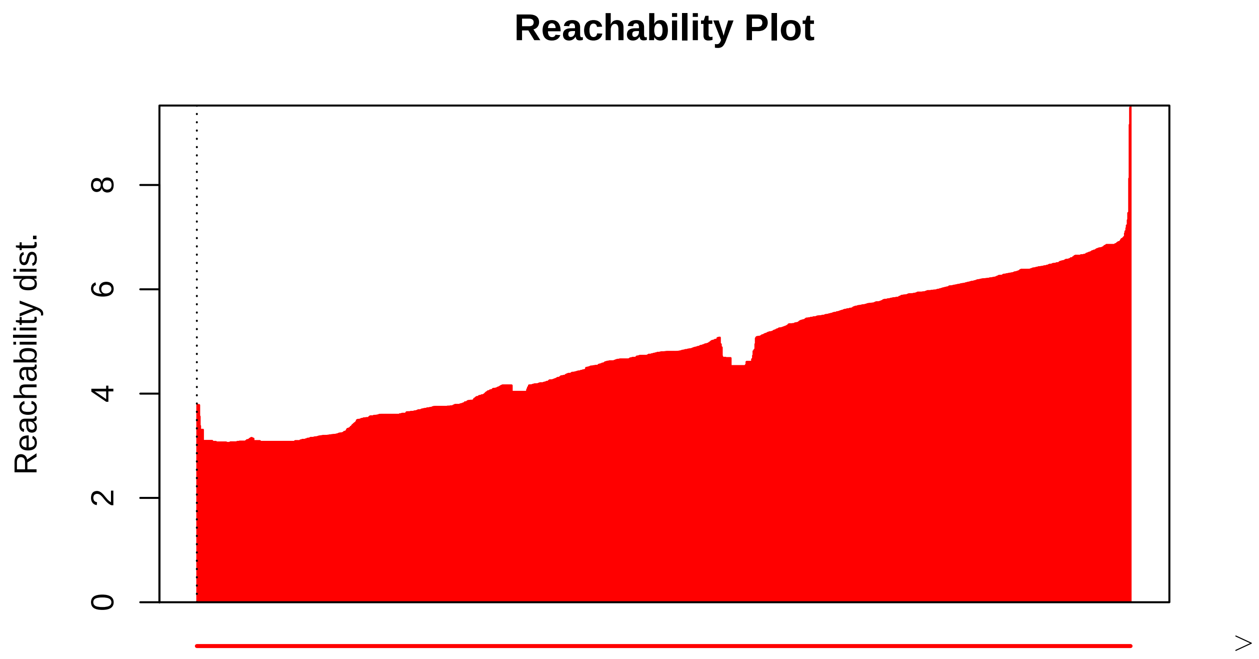
```r
plot(op)
```



```r
opXi <- extractXi(op, xi = 0.05)
hullplot(cluster_features, opXi, main = "OPTICSXi")
```



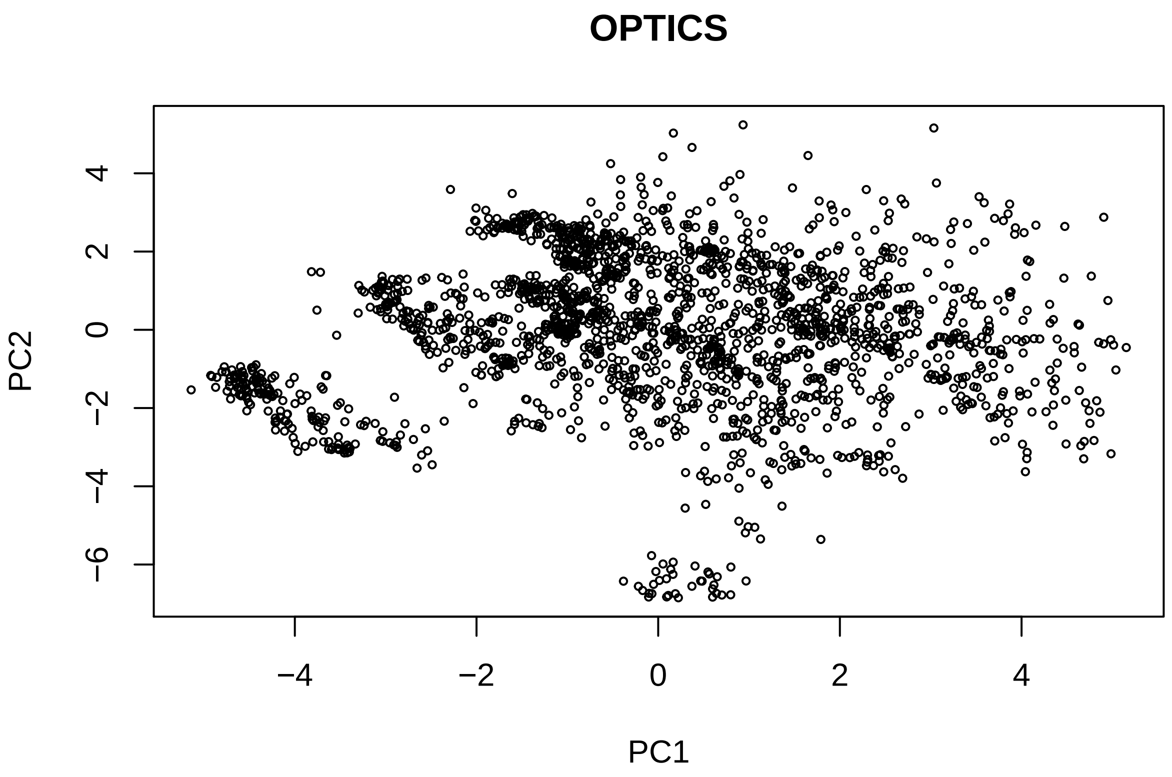
```
plot(opXi)
```



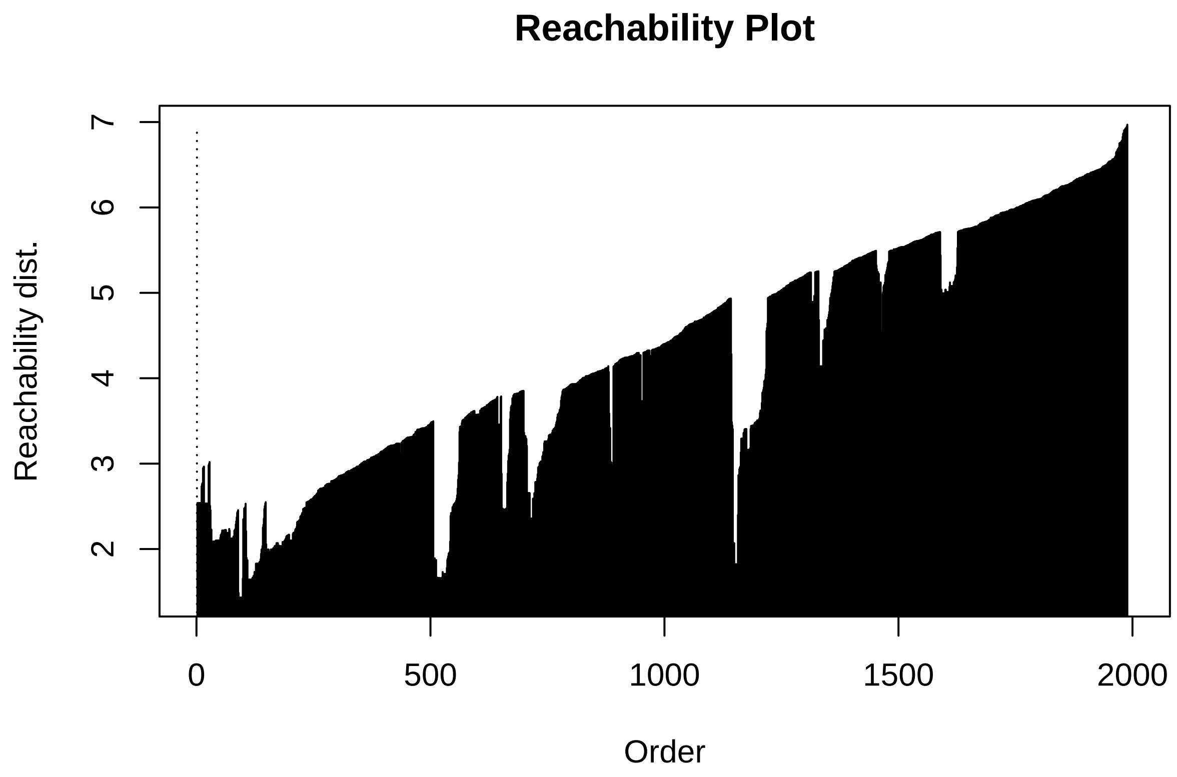
Once more, we can constrast the efficacy of OPTICS in yielding clustering structure results, as minPts of 50 is also to loose of a tolerance—same reason as previously stated, and we can further assess that `minPts=10` is sufficiently large to have a cluster-ordeitng solution that uncovers cluster structre.

eps at knee = 7

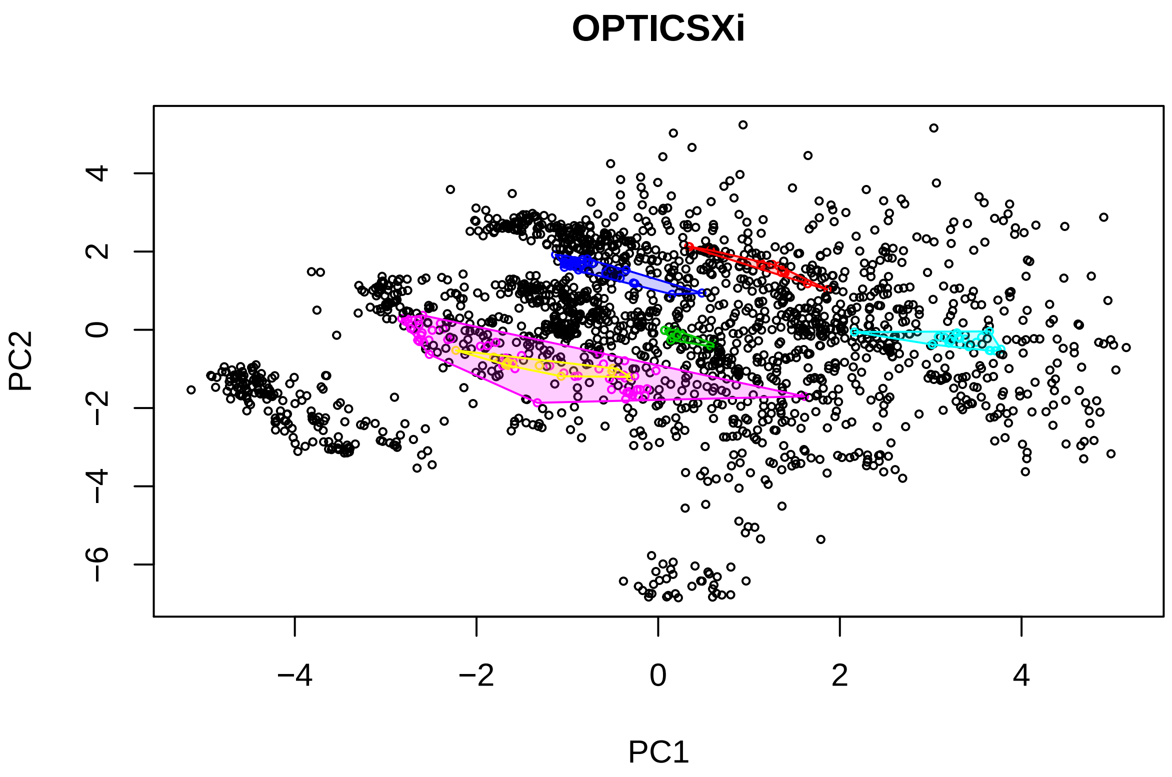
```
op <- optics(cluster_features, eps = 7, minPts = 10)
opDBSCAN <- extractDBSCAN(op, eps_cl = .07)
hullplot(cluster_features, opDBSCAN, main = "OPTICS")
```



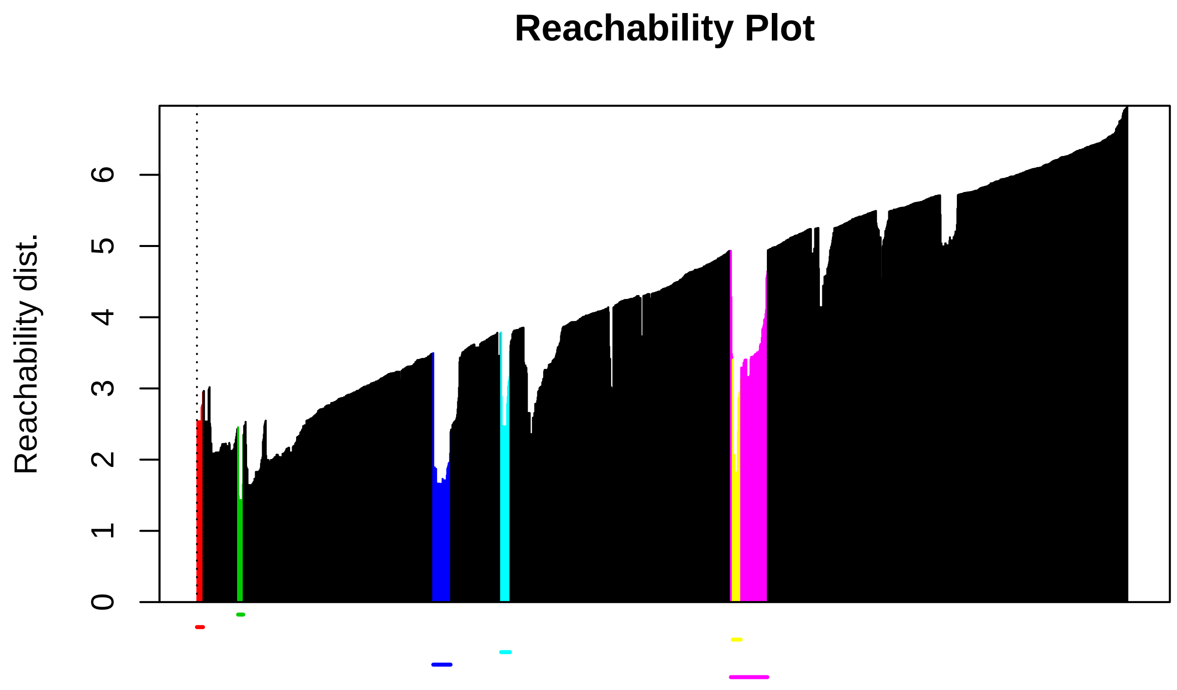
```r
plot(op)
```



```r
opXi <- extractXi(op, xi = 0.05)
hullplot(cluster_features, opXi, main = "OPTICSXi")
```

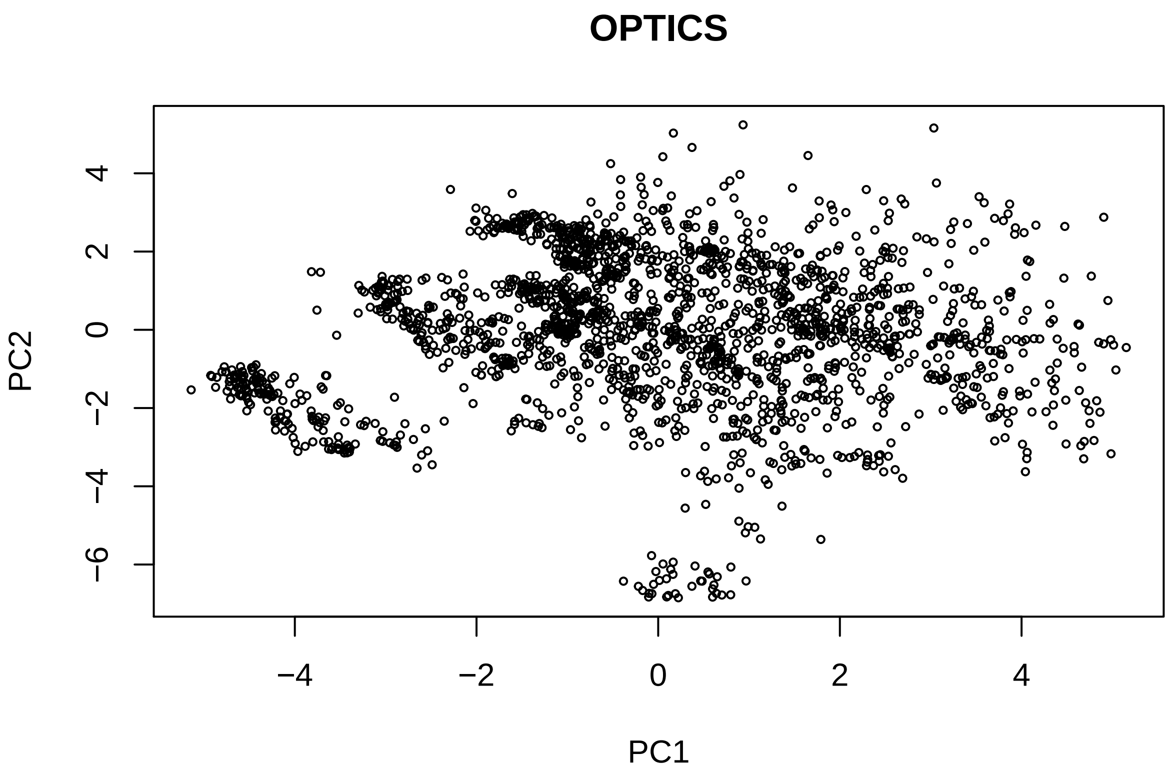


```r
plot(opXi)
```

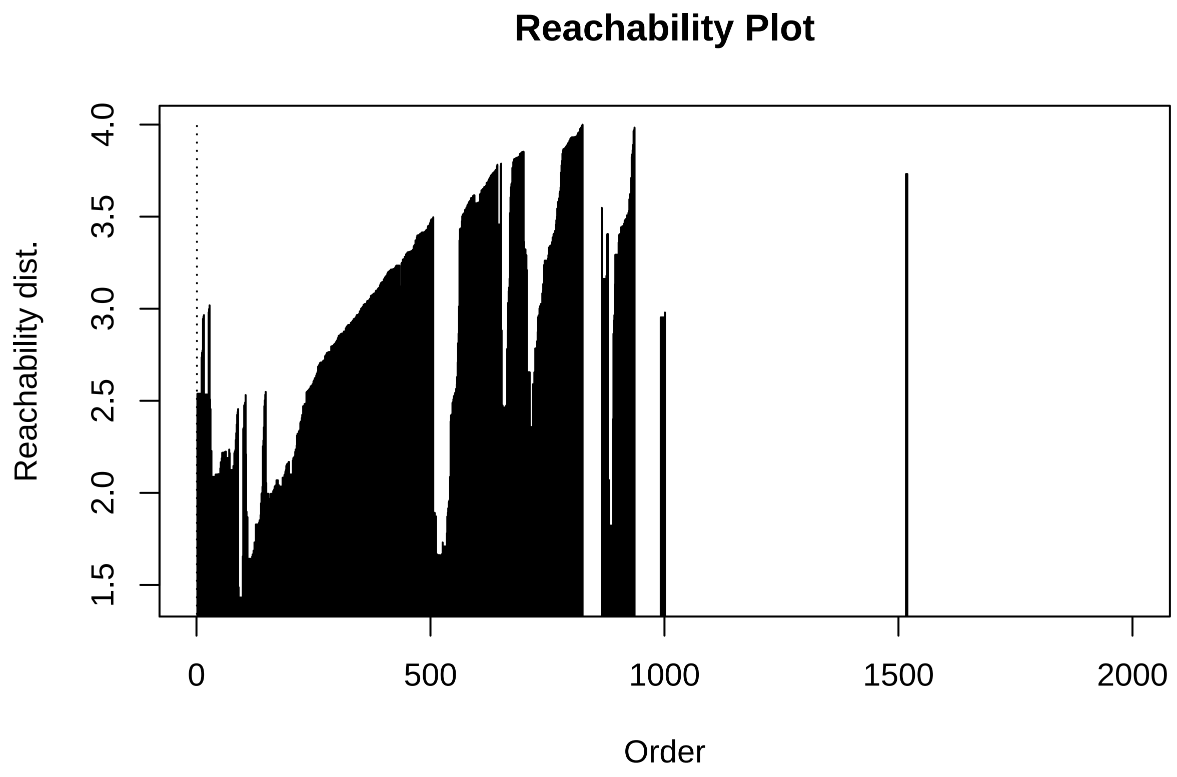


Slightly smaller eps at = 4

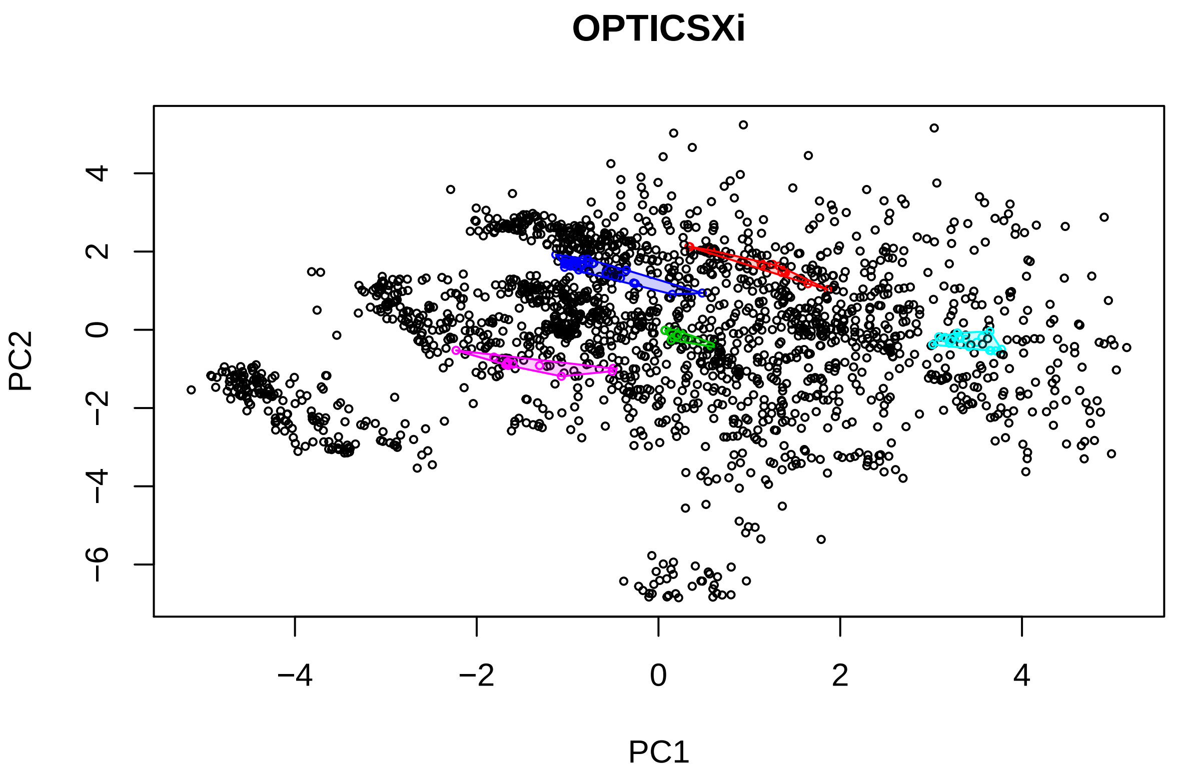
```r
op <- optics(cluster_features, eps = 4, minPts = 10)
opDBSCAN <- extractDBSCAN(op, eps_cl = .07)
hullplot(cluster_features, opDBSCAN, main = "OPTICS")
```



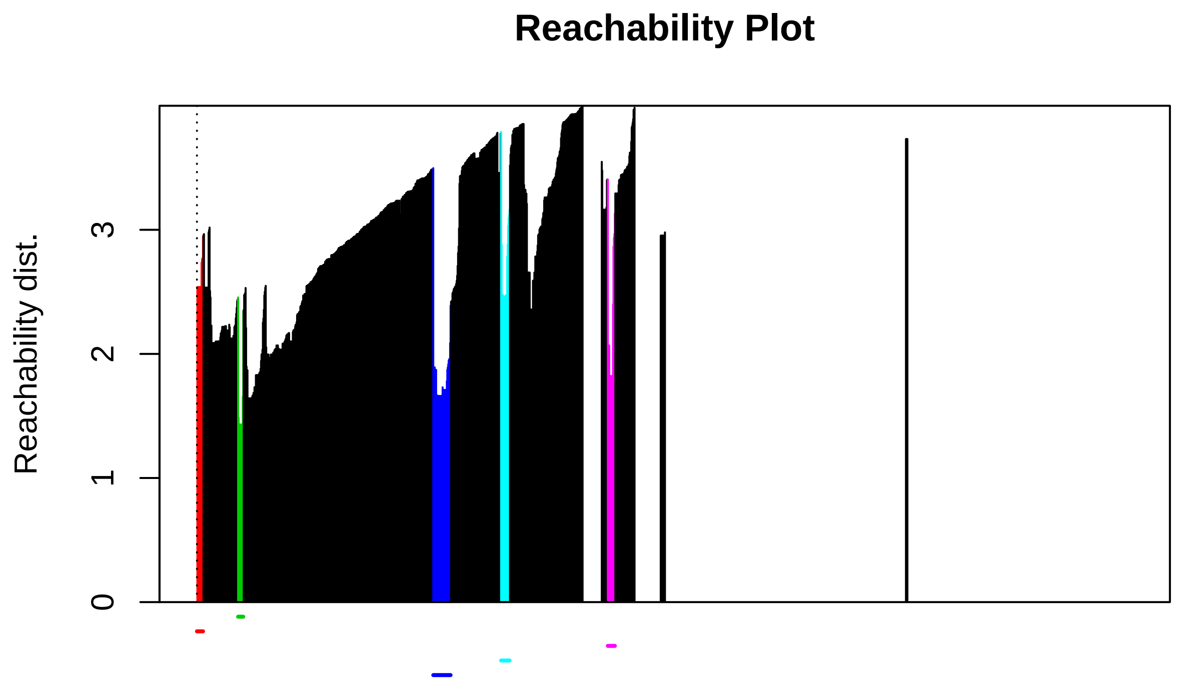
```r
plot(op)
```



```r
opXi <- extractXi(op, xi = 0.05)
hullplot(cluster_features, opXi, main = "OPTICSXi")
```


OPTICSXi
PC2
PC1
4
2
0
−2
−4
−6
−4
−2
0
2
4

```r
plot(opXi)
```


Reachability Plot
Reachability dist.
3
2
1
0