Contents

Wstęp

Załadowanie potrzebnych bibliotek

Do przetestowania naszych zbiorów będziemy potrzebowli następujących bibliotek:

```
library("dplyr")
library("plotly")
source("spectral.R")
library("dplyr")
library("plotly")
# dplyr overwrite groups, so:
groups <- igraph::groups
```

Kod funkcji

Implementajca pszczególnych funkcji algorytmu spektralnego znajdują się w pliku *spectral.R*. Dane linijki kodu komentowałem w trakcie pisania. Ewentualne inne rozwiązania danego zadania zamieściłem w komentarzach. Głównym powodem dla który wybierałem jeden sposób było przede wszystkim szybkość działania poszczególnego sposobu.

Funkcja *Mnn*

```
Mnn <- function(X, M){
  # calculate the distance beetween two points and save it as a matrix
  distOutput <- as.matrix(dist(X), method = "euclidean")

  # order the matrix
  orderedOutput <- apply(distOutput, 2, order)
  # in first column is the same column value (1 - 1) so we want to delete it
  orderedOutput <- orderedOutput[-1, ]

  # choose only this rows, which are the closest
  # t function to transpose result
  S <- t(orderedOutput[1:M, ])

}
```

Funkcja *Mnn_graph*

W tej funkcji dane składowe łączyłem za pomocą pętli *while*. Po tesstach zauważyłem, iż nie jest to wolny sposób. Jednym z decyzji, jakie podjąłem, było łączenie danych składowych (gdy liczba składowych jest większa niż 1). Postanowiłem, iż najlepszym rozwiązaniem (a zarazem najłatwiejszym) będzie połączenie poszczególnych składowych łącząc krawędzie o najniższych liczbach.

```
Mnn_graph <- function(S){
  # convert into adjacency matrix
  G <- matrix(0, nrow = nrow(S), ncol = nrow(S))

  for(row in 1:nrow(S)) {
    for(col in 1:ncol(S)) {
      G[row, S[row, col]] <- 1
      G[S[row, col], row] <- 1
    }
  }

  # creating a graph from a adjacency matrix
  ourGraph <- graph_from_adjacency_matrix(G, mode = c("undirected"),
                                          weighted = NULL, diag = FALSE)

  # calculating number of graph component
  comp <- components(ourGraph)
  componentsGroups <- groups(comp)
  componentsNumber <- length(componentsGroups)

  # if the number of component is bigger than 1 we add some edges
  while (componentsNumber != 1) {
    G[componentsGroups[[componentsNumber]][1],
      componentsGroups[[componentsNumber-1]][1]] <- 1
    G[componentsGroups[[componentsNumber-1]][1],
      componentsGroups[[componentsNumber]][1]] <- 1
    componentsNumber <- componentsNumber - 1
  }

  G
}
```

Funkcja *Laplacian_eigen*

```
Laplacian_eigen <- function(G, k){
  stopifnot(k > 1)

  # creating graph to calculate a degree of a vertex
  # (optional solution: sum of a row or a column)
  ourGraph <- graph_from_adjacency_matrix(G, mode = c("undirected"),
                                          weighted = NULL, diag = FALSE)

  # first solution
  # calculating a degree of a vertex
  #vertexDegree <- degree(ourGraph)

  # using diag function create D matrix
  #D = diag(vertexDegree, nrow(G), ncol(G))
  #L = D - G
```

```
  # second solution
  L <- laplacian_matrix(ourGraph)

  #stopifnot(isSymmetric(L))
  #eigenStructure <- eigen(L, symmetric = TRUE) # <- too slow

  # SA - the smallest(leftmost) values
  eigenStructure <- eigs_sym(L, 10 * k, which = "SA")

  vectorNumbers <- k + 1

  E <-  eigenStructure$vectors[, (ncol
                                  (eigenStructure$vectors) - k + 1)
                               :ncol(eigenStructure$vectors)]

  # alternatives (when we use a eigen function):
  #E <- eigenStructure$vectors[, order(eigenStructure$values,
  # decreasing = FALSE)[1:vectorNumbers]]

  E
}
```

Funkcja *spectral_clustering*

Efektem finalnym jest funkcja *spectral_clustering*. Wykorzystuje ona wcześniej zaimplementowane funkcje oraz korzysta dodatkowo z funkcji *kmeans*, która w sposób losowy wybiera punkt początkowy. Rodzi to pewne problemy - testując daną funkcję najlepiej to wykonać kilka razy a następnie obliczyć średnią z danych eksperymnetów. Ze wzgędu na skomplikowaność zadania postanowiłem tego nie wykonywać (badałem jedną próbę).

```
spectral_clustering <- function(X, M, k){
  S <- Mnn(X, M)
  G <- Mnn_graph(S)
  E <- Laplacian_eigen(G, k)
  kmeans(E, k)$cluster
}
```

Zapis danych zbiorów w postaci *.data* oraz *.labels0*

Wszystkie moje zbiory danych są zapisane w folderze *myBenchmark*.

```
# save data to .data and .label0 files

# first data

write.table(as.matrix(firstDataset[, 1:2]),
            file = "../myBenchmark/benchmark1.data",
            row.names=FALSE, na="",col.names=FALSE, sep=" ")
```

```
write.table(as.matrix(firstDataset[, 3]),
            file = "../myBenchmark/benchmark1.labels0",
            row.names=FALSE, na="",col.names=FALSE, sep=" ")
```

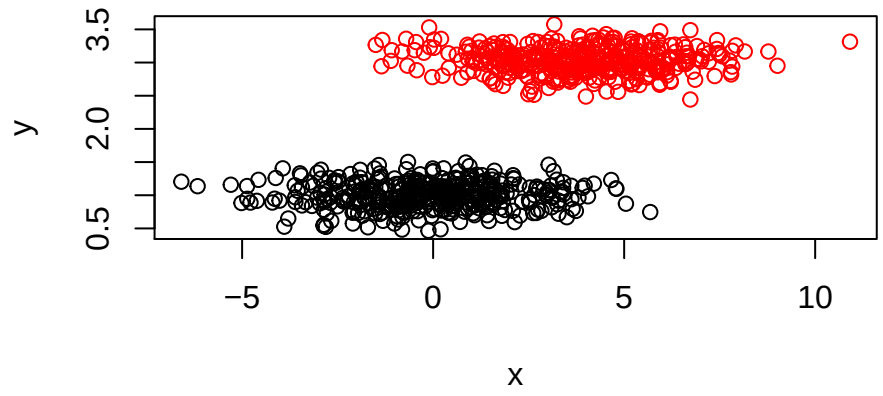
Pozostałe zbiory zapisujemy w podobny sposób.

Pierwszy zbiór

Kod tworzący zbiór

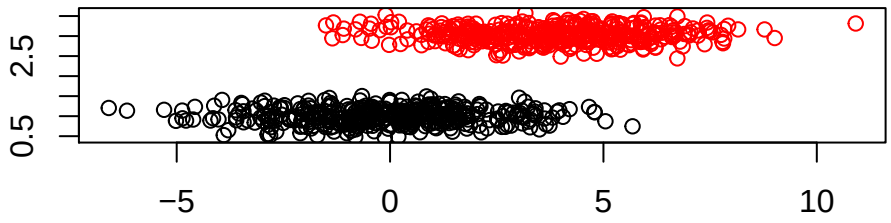
```
firstDataset <- {
  # first cluster
  n <- 400

  # first cluster is around point (1, 1)

  firstCluster <- data.frame(
    x <- rnorm(n, mean = 0, sd = 2),
    y <- rnorm(n, mean = 1, sd = 0.2)
  )
  names(firstCluster) <- c("x", "y")
  firstCluster <- firstCluster %>% mutate(class=factor(1))

  # second cluster
  # second cluster is around point (3, 3)

  secondCluster <- data.frame(
    x <- rnorm(n, mean = 4, sd = 2),
    y <- rnorm(n, mean = 3, sd = 0.2)
  )
  names(secondCluster) <- c("x", "y")
  secondCluster <- secondCluster %>% mutate(class=factor(2))

  # bind our data
  firstDataset = bind_rows(firstCluster, secondCluster)

}
```

Ilustracja zbioru



Testy

Ilustracja testów



Indeks Fowlkesa-Mallowsa

```
round(as.numeric(FM_index(label, calculatedLabel)), decimalPlaces)
```

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

Indeks Randa (skorygowany)

```
round(as.numeric(mclust::adjustedRandIndex(label, calculatedLabel)), decimalPlaces)
```

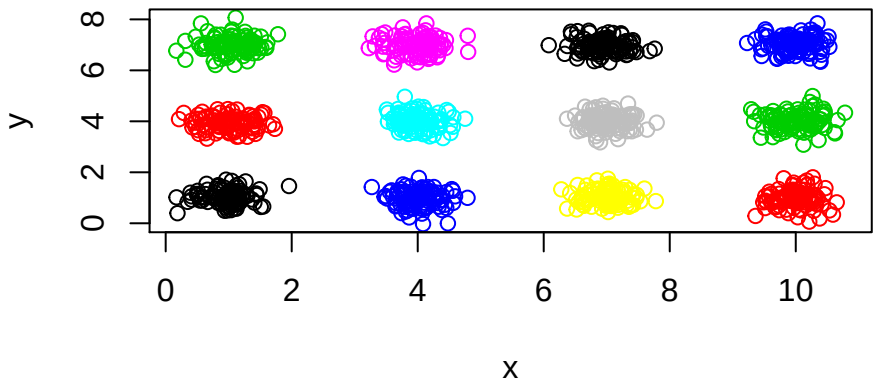
```
## [1] 1
```

Drugi zbiór

Kod tworzący zbiór

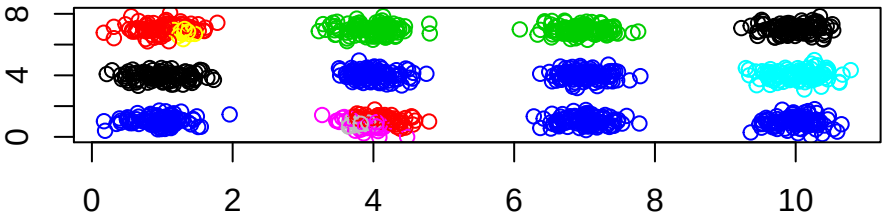
```
secondDataset <- {
  clusterNumber = 12
  n <- 100

  # in this dataset is 2-d data with clusterNumber clusters

  finalDataset <- data.frame(x = numeric(),
                             y = numeric(),
                             label = numeric())

  # creating a rectangle with some characteristic neighbourhoods
  for (i in 1:clusterNumber){
    temp <- data.frame(
      x <- rnorm(n,
                 mean = 3 * (i-1) %/% as.integer(sqrt(clusterNumber)) + 1,
                 sd = 0.3),
      y <- rnorm(n,
                 mean = 3 * (i-1) %% as.integer(sqrt(clusterNumber)) + 1,
                 sd = 0.3),
      label <- list(rep(i,n))
    )
    names(temp) <- c("x", "y", "label")

    finalDataset <- rbind(finalDataset, temp)
  }

  finalDataset

}

X <- secondDataset[, 1:2]
label <- secondDataset[, 3]
```

Ilustracja zbioru



Testy

Ilustracja testów



Indeks Fowlkesa-Mallowsa

```
round(as.numeric(FM_index(label, calculatedLabel)), decimalPlaces)
```

```
## [1] 0.664
```

Indeks Randa (skorygowany)

```
round(as.numeric(mclust::adjustedRandIndex(label, calculatedLabel)), decimalPlaces)
```

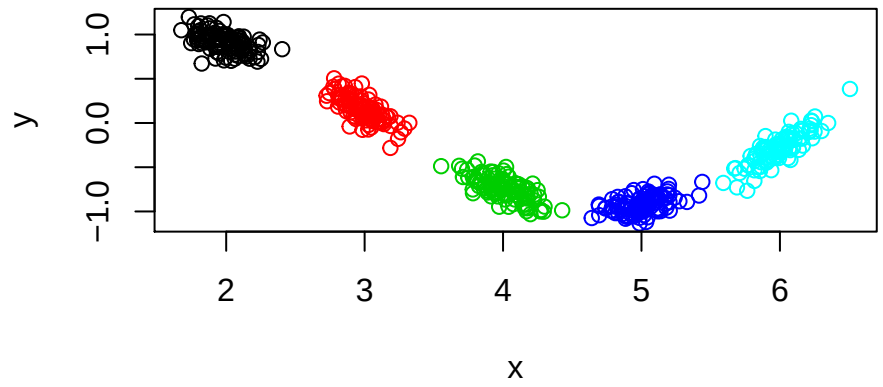
```
## [1] 0.585
```

Trzeci zbiór

Kod tworzący zbiór

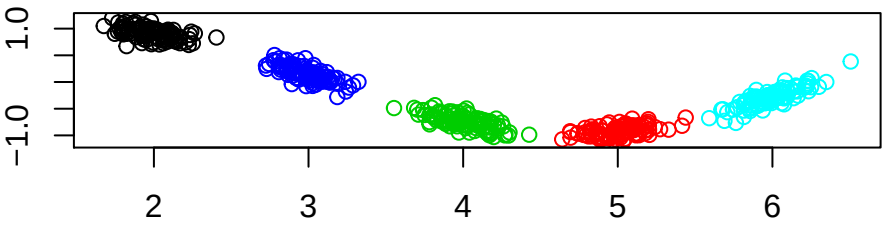
```
thirdDataset <- {
  clusterNumber = 5
  n <- 100

  finalDataset <- data.frame(x = numeric(),
                             y = numeric(),
                             label = numeric())

  for (i in 1:clusterNumber){
    temp <- data.frame(
      x <- rep(i:i+1, n) + rnorm(n, mean = 0, sd = 0.15),
      y <- sin(x) + rnorm(n, 0, sd = 0.1),
      label <- list(rep(i,n))
    )
    names(temp) <- c("x", "y", "label")

    finalDataset <- rbind(finalDataset, temp)
  }

  finalDataset

}
```

Ilustracja zbioru



Testy

Ilustracja testów



Indeks Fowlkesa-Mallowsa

```
round(as.numeric(FM_index(label, calculatedLabel)), decimalPlaces)
```

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

Indeks Randa (skorygowany)

```
round(as.numeric(mclust::adjustedRandIndex(label, calculatedLabel)), decimalPlaces)
```

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

Czwarty zbiór

Kod tworzący zbiór

```
fourthDataset <- {
  clusterNumber = 5
  n <- 100

  finalDataset <- data.frame(x = numeric(),
                             y = numeric(),
                             z = numeric(),
                             label = numeric())

  for (i in 1:clusterNumber){
    temp <- data.frame(
      x <- rep(1:5, n) + rnorm(n, mean = 0, sd = 0.5),
      y <- rep(1:5, n) + rnorm(n, mean = 0, sd = 0.5),
      z <- 2.5 * i + rnorm(n, mean = 0, sd = 0.1),
      label <- list(rep(i,n))
    )
    names(temp) <- c("x", "y", "z", "label")

    finalDataset <- rbind(finalDataset, temp)
  }

  finalDataset
}


X <- fourthDataset[, 1:3]
label <- fourthDataset[, 4]
```

Ilustracja zbioru

Niestety pliki *.Rmd* kompilując do formatu *.pdf* nie zapisuje wykresów (kompulując do *html* jest to możliwe, lecz plik miał się kompilować do formatu *pdf*). Odpowiednie testy można jednak otworzyć w *RStudio*.

Ilustracja testów

Niestety pliki *.Rmd* kompilując do formatu *.pdf* nie zapisuje wykresów (kompulując do *html* jest to możliwe, lecz plik miał się kompilować do formatu *pdf*). Odpowiednie testy można jednak otworzyć w *RStudio*.

Indeks Fowlkesa-Mallowsa

```
round(as.numeric(FM_index(label, calculatedLabel)), decimalPlaces)
```

```
## [1] 0.79
```

Indeks Randa (skorygowany)

```
round(as.numeric(mclust::adjustedRandIndex(label, calculatedLabel)), decimalPlaces)
```

```
## [1] 0.736
```