Exercise Set 3 - solutions and grading

Please read the following before doing the peer reviews:

- Problem sheet at https://moodle.helsinki.fi/pluginfile.php/3324196/mod_folder/content/0/DATA11002-2020-E3.pdf?forcedownload=1
- General grading instructions at https://moodle.helsinki.fi/mod/page/view.php?id=2072012
- Step-by-step instructions for the peer review week at https://moodle.helsinki.fi/mod/page/view.php?id=2084536

Please participate to one peer review session during 8-10 December and submit your peer reviews on 11 December, at latest.

We use here the short-hand notation $[n] = \{1, \ldots, n\}$.

Problem 16

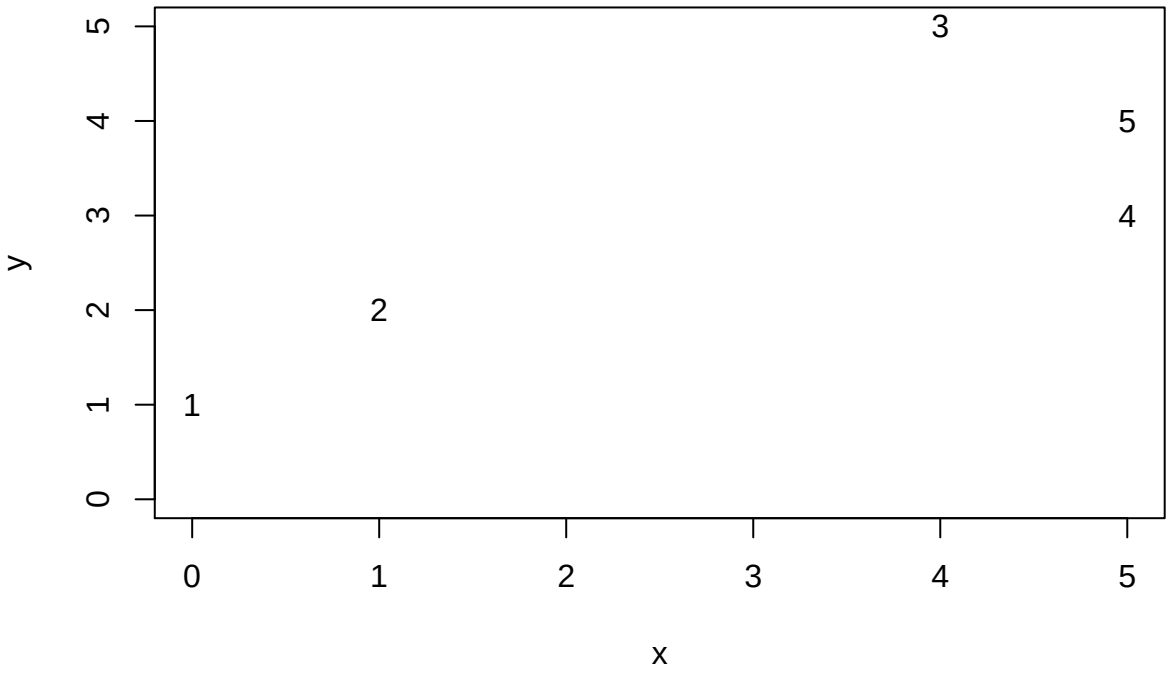
[20% points]

Objectives: k-means loss and Lloyd's algorithm

```
set.seed(42)
```

```
data <- data.frame(x=c(0,1,4,5,5),y=c(1,2,5,3,4))
```

```
plot(c(0,5),c(0,5),type="n",xlab=expression(x),ylab=expression(y))
text(data,rownames(data))
```



Task a

> What kind of tasks can we use the Lloyd's (k-means) algorithm for? Explain what the *inputs* and *outputs* of the algorithm are. How to interpret the results?

The Lloyd's algorithm is used to cluster data points.

The algorithm assumes that our data is composed of n p-dimensional data vectors $x_i \in \mathbb{R}^p$, where $i \in [n]$.

The algorithm takes as an input:

- the n data vectors,
- number of cluster centroids or prototypes k, and
- the initial cluster centroids $\mu_j \in \mathbb{R}^p$, where $j \in [k]$ (or alternatively: initial assignment of data points into clusters $z_i \in \{1, \ldots, k\}$, where $i \in [n]$).

The algorithm gives as an output:

- the final cluster centroids $\mu_j \in \mathbb{R}^p$, , where $j \in [k]$, and/or
- assignment of data points into clusters $z_i \in [k]$, where $i \in [n]$.

Intuitively, the points within cluster should be nearby and two points in different clusters should be further away from each other.

Task b

> Define the objective (or cost) function that the Lloyd's algorithm tries to minimize. What can be said about the value of the objective function during the two stages of each iteration of Lloyd's algorithm?

The Lloyd's algorithm attempts to greedily minimize the total within-cluster squared loss given by

$$L_{k-means} = \sum_{j=1}^{k} \sum_{i \in S_j} |x_i - \mu_j|^2 = \sum_{i=1}^{n} |x_i - \mu_{z_i}|^2,$$

where the items in cluster $j \in [k]$ are given by $S_j = \{i \in [n] \mid z_i = j\}$.

The Lloyd's algorithm consists of repeated application of the following two iterations:

a. Assign each point to the nearest cluster centroid, or $z_i \leftarrow \arg\min_{z \in [k]} |x_i - \mu_z|$ for all $i \in [n]$.
b. Update cluster centroids to the mean of datapoints in the cluster, or $\mu_j \leftarrow \sum_{i \in S_j} x_i / |S_j|$ for all $j \in [k]$.

Lets then look how the loss function changes in the iterations above.

Iteration a. For a given $i \in [n]$ the iteration either leaves the loss unchanged or then (because we take minimum) it reduces the loss. I.e., iteration a cannot increase the loss.

Iteration b. For a given $j \in [k]$ the sum $\sum_{i \in S_j} |x_i - \mu_j|^2$ is minimized if μ_j is the mean of data vectors x_i in S_j. Therefore, iteration b cannot increase the loss either.

(We can write $\sum_{i \in S_j} |x_i - \mu_j|^2 = \sum_{i \in S_j} \sum_{k=1}^{p} (x_{ip} - \mu_{jp})^2$. The pth component of the loss is minimal at the root of the derivative, or $\partial\Big(\sum_{i \in S_j} (x_{ip} - \mu_{jp})^2\Big)/\partial \mu_{jp} = 2\sum_{i \in S_j} x_{ip} - 2|S_j|\mu_{jp} = 0$, or $\mu_{jp} = \sum_{i \in S_j} x_{ip}/|S_j|$. The quadratic sum is therefore minimised when μ_j is the mean vector.)

Summarizing, the value of the loss function does not increase at any iteration of Lloyd's algorithm. If the loss function is unchanged after two iterations the Lloyd's algorithm stops and the final centroid vectors and/or cluster membership indices are output.

Task c

> Consider the following set of data points in $\mathbb{R}^2$: $x_1 = (0, 1)$, $x_2 = (1, 2)$, $x_3 = (4, 5)$, $x_4 = (5, 3)$, $x_5 = (5, 4)$. Sketch the run of the Lloyd's algorithm using $K = 2$ and initial prototype (mean)

vectors $\mu_1 = (0, 2)$ and $\mu_2 = (2, 0)$. Draw the data points, cluster prototype vectors, and cluster boundary after each iteration until convergence.

Also, write down calculation procedure and the cluster memberships as well as prototype vectors after each iteration.

We can characterize the state of the algorithm by the prototype vectors μ_1 and μ_2 and the cluster membership indices $z \in \{1, 2\}^5$. Lets make implementation of Lloyd's algorithm. (**Grading:** You do not need to do implementation for full points. It is enough run the algorithm "by hand" and produce the same output than the implementation below.)

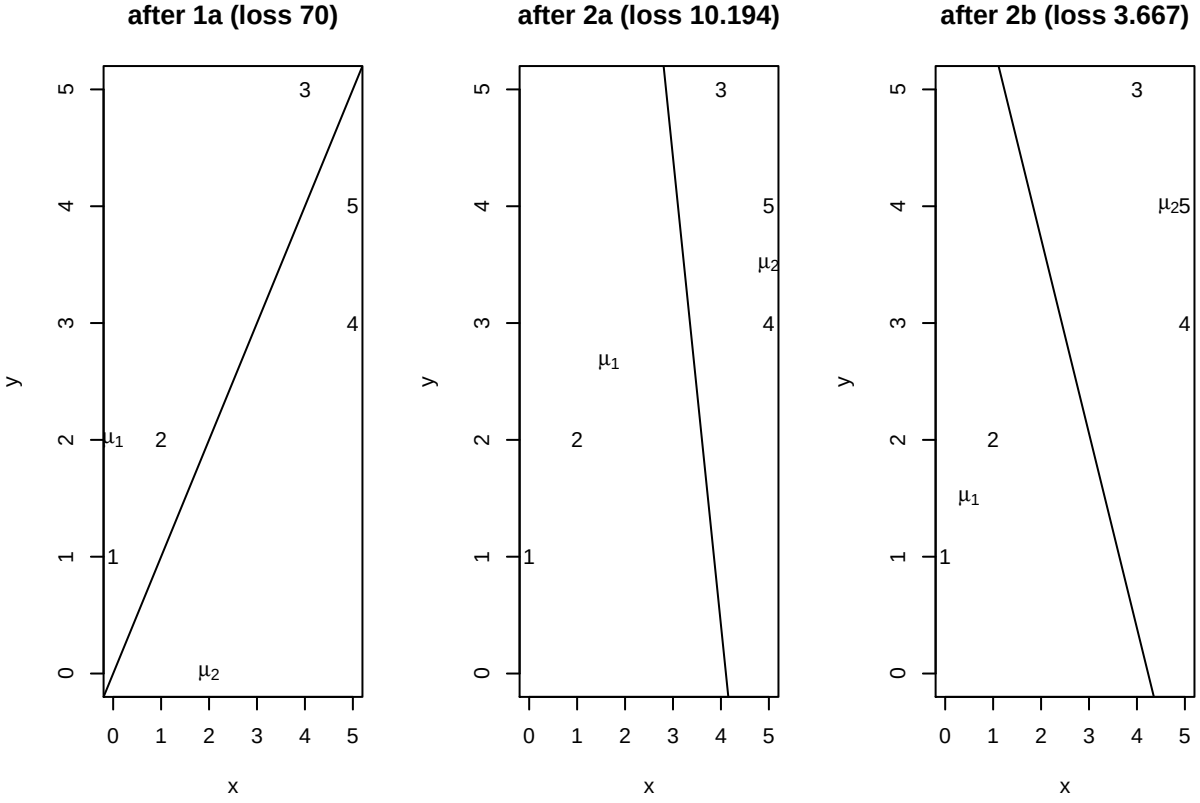
```
data <- as.matrix(data)

updatez <- function(mu) {
  ## Assign each datapoint to the nearest centroid
  apply(data,1,function(x) if(sum((x-mu[1])^2)<sum((x-mu[2])^2)) 1 else 2)
}
updatemu <- function(z) {
  ## Update centroids to the means of datapoints
  data.frame(mu1=apply(data[z==1,],2,mean),mu1=apply(data[z==2,],2,mean))
}
loss <- function(mu,z) {
  ## Total within-cluster loss
  sum(sapply(1:nrow(data),function(i) sum((data[i,]-mu[z[i]])^2)))
}
lloyds <- function(mu,z=rep(0,nrow(data))) {
  i <- 1
  cat(sprintf("initially:    mu1 <- (%.3f,%.3f)  mu2 <- (%.3f,%.3f)\n\n",
              mu[1,1],mu[2,1],mu[1,2],mu[2,2]))
  while(TRUE) {
    znew <- updatez(mu)
    ## stop if z remains unchanged
    if(all(znew==z)) break
    z <- znew

    cat(sprintf("iteration %da: z <- (%d,%d,%d,%d,%d)\n",
                i,z[1],z[2],z[3],z[4],z[5]))
    cat(sprintf("              loss = %.3f\n\n",loss(mu,z)))

    mu <- updatemu(z)

    cat(sprintf("iteration %db: mu1 <- (%.3f,%.3f)  mu2 <- (%.3f,%.3f)\n",
                i,mu[1,1],mu[2,1],mu[1,2],mu[2,2]))
    cat(sprintf("              loss = %.3f\n\n",loss(mu,z)))

    i <- i+1
  }
  list(mu=mu,z=z)
}

## Set initial mu as instructed and run Lloyd's algorithm
a <- lloyds(mu=data.frame(mu1=c(0,2),mu2=c(2,0)))
```

```
## initially:    mu1 <- (0.000,2.000)  mu2 <- (2.000,0.000)
##
## iteration 1a: z <- (1,1,1,2,2)
```

```
##               loss = 70.000
##
## iteration 1b: mu1 <- (1.667,2.667)  mu2 <- (5.000,3.500)
##               loss = 17.833
##
## iteration 2a: z <- (1,1,2,2,2)
##               loss = 10.194
##
## iteration 2b: mu1 <- (0.500,1.500)  mu2 <- (4.667,4.000)
##               loss = 3.667
```

Above the run of the algorithm is described, when we have initialised the initial prototypes to $\mu_1 = (0, 2)$ and $\mu_2 = (2, 0)$. In iteration a, we assign each point to the nearest centroid and in iteration b we update the cluster centoids to the means of data vectors in the cluster. We report the total within-cluster loss after each of the iterations. We notice that after 4 steps the algorithm has converged to the final state having $\mu_1 = (0.5, 1.5)$, $\mu_2 = (4.667, 4)$, and $z = (1, 1, 2, 2, 2)$. The initial loss is 70 and the final loss 3.667.

The cluster centroids and the cluster boundary after the iterations 1a, 2a, and 3a, respectively, are shown as follows:



Grading

Points: max. 20 (a: 5 b: 7 c: 8)

Problem 17

[20% points]

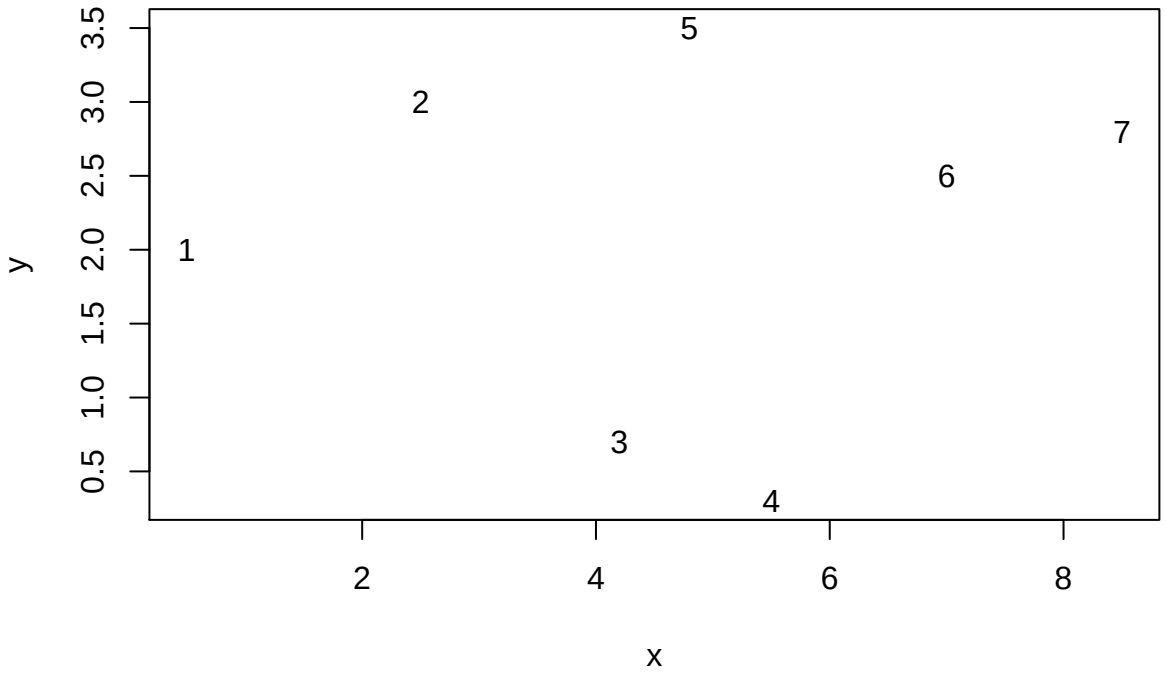
Objectives: understanding hierarchical clustering algorithms

We consider *hierarchical clustering* on a toy data set $D = \{(x_i, y_i)\}_{i=1}^{7}$, where $(x_i, y_i) \in \mathbb{R}^2$. You should be able to do this problem with a pen and paper.

The data points are shown in the table and figure below.

```
data2 <- data.frame(x=c(0.5,2.5,4.2,5.5,4.8,7.0,8.5),
                    y=c(2.0,3.0,0.7,0.3,3.5,2.5,2.8))
```

	x	y
1	0.5	2.0
2	2.5	3.0
3	4.2	0.7
4	5.5	0.3
5	4.8	3.5
6	7.0	2.5
7	8.5	2.8



The matrix of Euclidean distances between the data points is given below.

	1	2	3	4	5	6	7
1	0.00	2.24	3.92	5.28	4.55	6.52	8.04
2	2.24	0.00	2.86	4.04	2.35	4.53	6.00
3	3.92	2.86	0.00	1.36	2.86	3.33	4.79
4	5.28	4.04	1.36	0.00	3.28	2.66	3.91
5	4.55	2.35	2.86	3.28	0.00	2.42	3.77
6	6.52	4.53	3.33	2.66	2.42	0.00	1.53
7	8.04	6.00	4.79	3.91	3.77	1.53	0.00

Task a

> Sketch a run of the basic agglomerative hierarchical clustering algorithm to this data using the single link (min) notion of dissimilarity between clusters. Visualise the result as a dendrogram. Draw a sketch of the above figure in your answer sheet and indicate in this how the algorithm forms clusters step-by-step. It is not necessary to show detailed calculations, but indicate the cost of the join that you select.

The idea of agglomerative hierarchical clustering is that we start with all points in singleton clusters. We then start merging the clusters so that at each iteration we merge the two clusters with the smallest dissimilarity, until all points are in one cluster.

We will first sketch the run of the algorithm with single link dissimilarity between the clusters, which is defined by the smallest distance between any two points in the two clusters or $L_{single}(A, B) = \min_{i \in A, j \in B} d(i, j)$.

Initial clusters: $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}\}$

Iteration 1. The clusters $A_1 = \{3\}$ and $B_1 = \{4\}$ have the smallest distance $L_{single}(A_1, B_1) = d(3, 4) = 1.36$. Merge A_1 and B_1. Clusters: $\{\{1\}, \{2\}, \{3, 4\}, \{5\}, \{6\}, \{7\}\}$

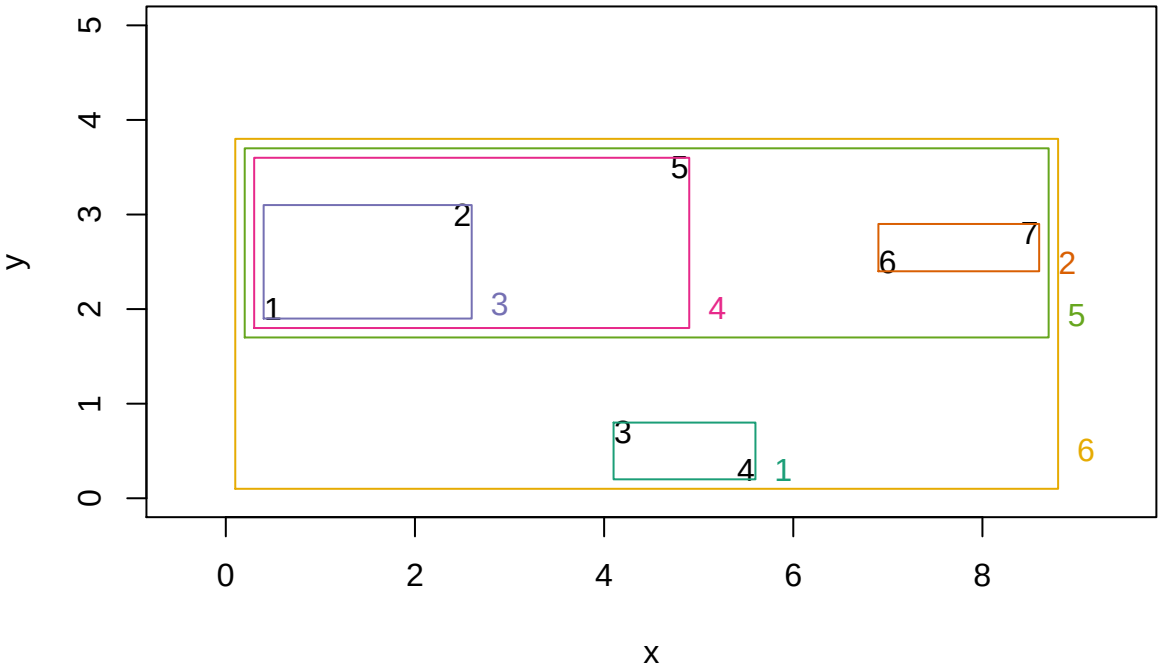
Iteration 2. The clusters $A_2 = \{6\}$ and $B_2 = \{7\}$ have the smallest distance $L_{single}(A_2, B_2) = d(6, 7) = 1.53$. Merge A_2 and B_2. Clusters: $\{\{1\}, \{2\}, \{3, 4\}, \{5\}, \{6, 7\}\}$

Iteration 3. The clusters $A_3 = \{1\}$ and $B_3 = \{2\}$ have the smallest distance $L_{single}(A_3, B_3) = d(1, 2) = 2.25$. Merge A_3 and B_3. Clusters: $\{\{1, 2\}, \{3, 4\}, \{5\}, \{6, 7\}\}$

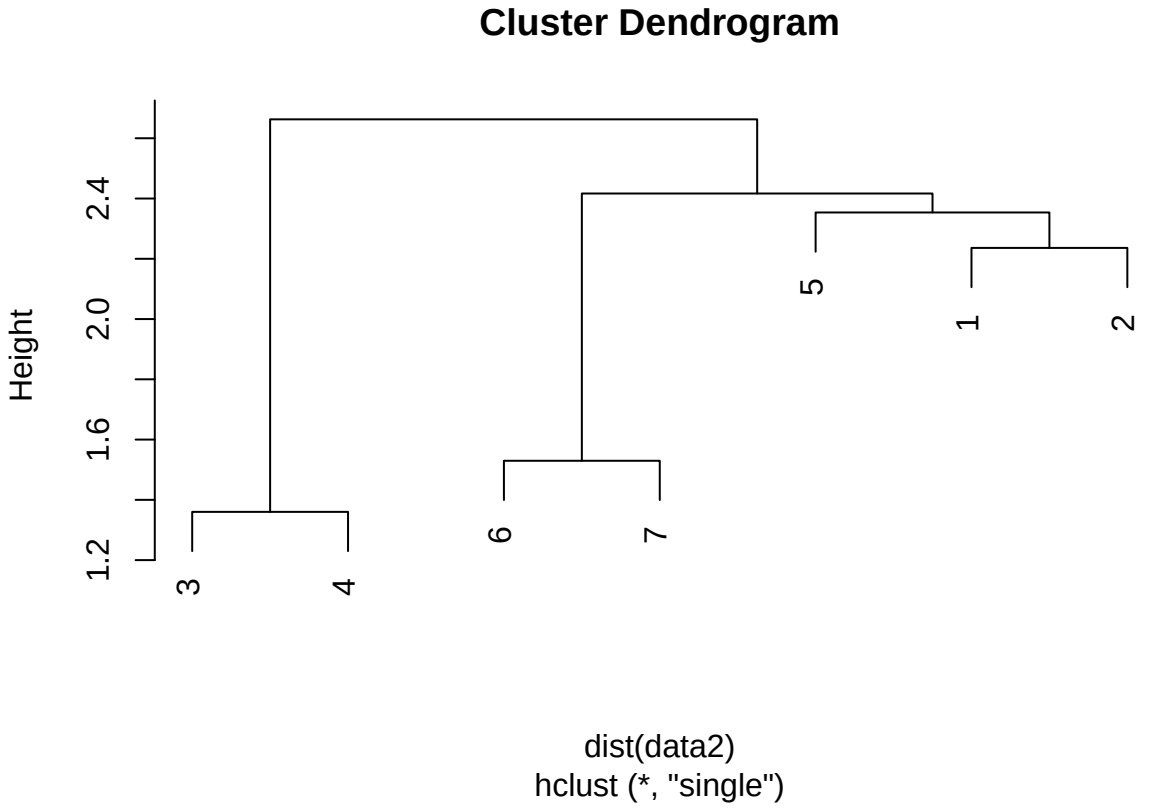
Iteration 4. The clusters $A_4 = \{1, 2\}$ and $B_4 = \{5\}$ have the smallest distance $L_{single}(A_4, B_4) = d(2, 5) = 2.35$. Merge A_4 and B_4. Clusters: $\{\{1, 2, 5\}, \{3, 4\}, \{6, 7\}\}$

Iteration 5. The clusters $A_5 = \{1, 2, 5\}$ and $B_5 = \{6, 7\}$ have the smallest distance $L_{single}(A_5, B_5) = d(5, 6) = 2.42$. Merge A_5 and B_5. Clusters: $\{\{1, 2, 5, 6, 7\}, \{3, 4\}\}$

Iteration 6: The remaining clusters $A_6 = \{1, 2, 5, 6, 7\}$ and $B_6 = \{3, 4\}$ have the distance $L_{single}(A_6, B_6) = d(4, 6) = 2.66$. Merge A_6 and B_6. Clusters: $\{\{1, 2, 3, 4, 5, 6, 7\}\}$



Dendrogram:



Notice that the height of the branches matches the linkage function values at each join!

Task b

Repeat the task a, but using complete link (max) dissimilarity. Compare the results.

The complete link clustering is defined by the linkage $L_{complete}(A, B) = \max_{i \in A, j \in B} d(i, j)$.

Initial clusters: $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}\}$

Iteration 1. The clusters $A_1 = \{3\}$ and $B_1 = \{4\}$ have the smallest distance $L_{complete}(A_1, B_1) = d(3, 4) = 1.36$. Merge A_1 and B_1. Clusters: $\{\{1\}, \{2\}, \{3, 4\}, \{5\}, \{6\}, \{7\}\}$

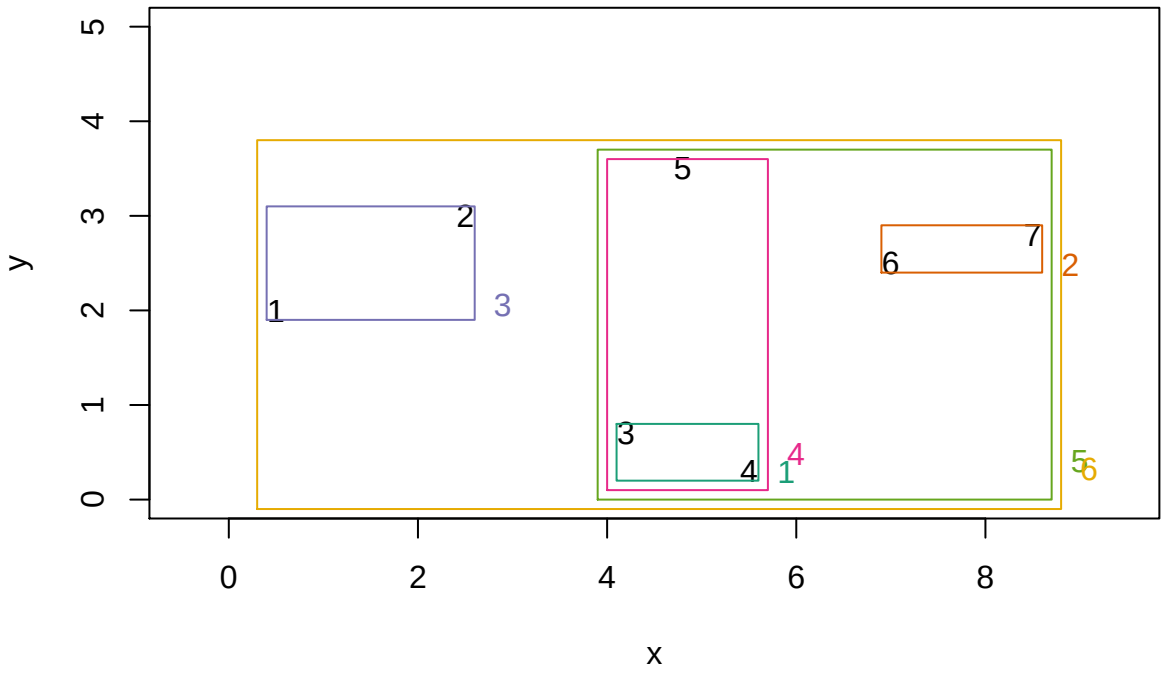
Iteration 2. The clusters $A_2 = \{6\}$ and $B_2 = \{7\}$ have the smallest distance $L_{complete}(A_2, B_2) = d(6, 7) = 1.53$. Merge A_2 and B_2. Clusters: $\{\{1\}, \{2\}, \{3, 4\}, \{5\}, \{6, 7\}\}$

Iteration 3. The clusters $A_3 = \{1\}$ and $B_3 = \{2\}$ have the smallest distance $L_{complete}(A_3, B_3) = d(1, 2) = 2.25$. Merge A_3 and B_3. Clusters: $\{\{1, 2\}, \{3, 4\}, \{5\}, \{6, 7\}\}$

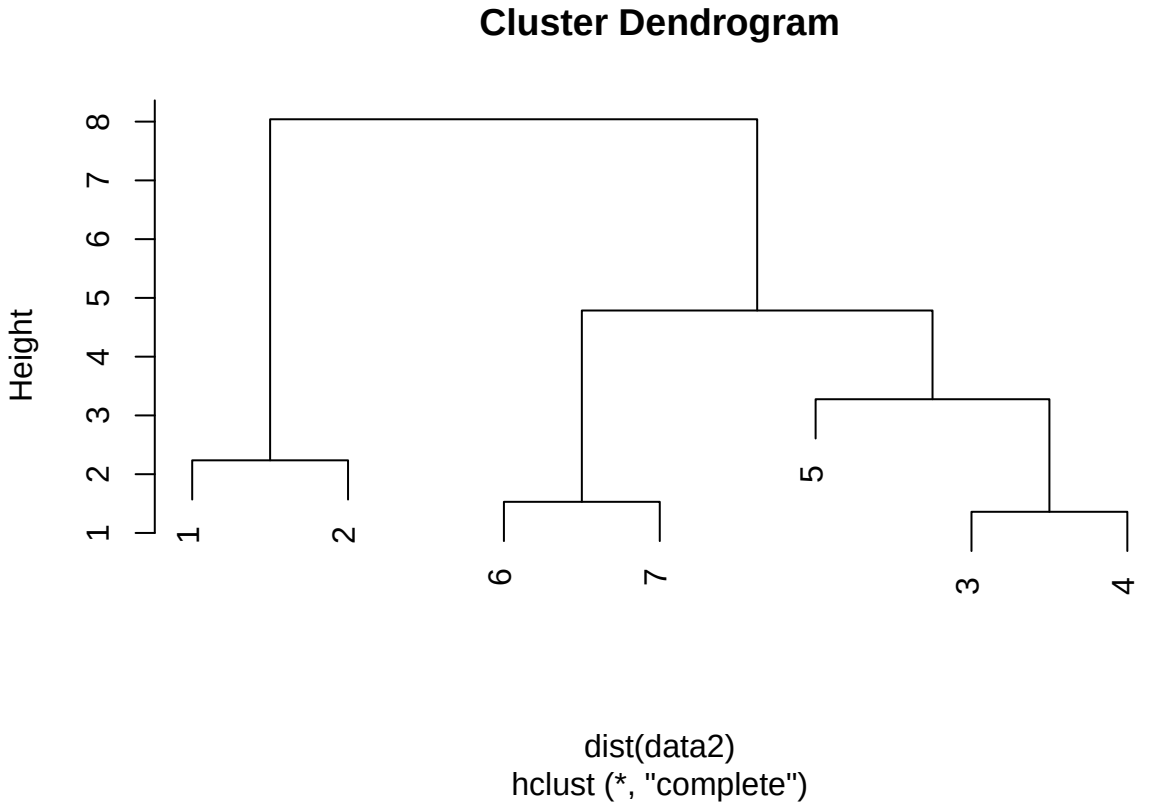
Iteration 4. The clusters $A_4 = \{3, 4\}$ and $B_4 = \{5\}$ have the smallest distance $L_{complete}(A_4, B_4) = d(4, 5) = 3.28$. Merge A_4 and B_4. Clusters: $\{\{1, 2\}, \{3, 4, 5\}, \{6, 7\}\}$

Iteration 5. The clusters $A_5 = \{3, 4, 5\}$ and $B_5 = \{6, 7\}$ have the smallest distance $L_{complete}(A_5, B_5) = d(3, 4) = 4.79$. Merge A_5 and B_5. Clusters: $\{\{1, 2\}, \{3, 4, 5, 6, 7\}\}$

Iteration 6: The remaining clusters $A_6 = \{1, 2\}$ and $B_7 = \{3, 4, 5, 6, 7\}$ have the distance $L_{complete}(A_6, B_6) = d(1, 7) = 8.04$. Merge A_6 and B_6. Clusters: $\{\{1, 2, 3, 4, 5, 6, 7\}\}$



Dendrogram:



Notice that the height of the branches matches the linkage function values at each join!

The complete linkage clustering tends to result in clusters of roughly the same cluster diameter (diameter is the maximum distance between any two points within a cluster), if the tree would be cut at any level.

Grading

Points: max. 20 (a: 10, b: 10)

Problem 18

[25% points]

Objectives: practical application of k-means and hierarchical clustering

For this problem you can use the term project dataset in file `npf_train.csv`. Your task is to cluster the data matrix, where the rows are given by the days and columns by various observations during that day. In this task you need only the class variable (`class4`) and the real valued mean measurements (variable names ending with `.mean`). You should end up with a dataset of 430 rows and 50 real valued variables and one class variable as columns. The R code to produce the data is given below for reference (in the last line we strip out the redundant `.mean` from the variable names!).

```
npf <- read.csv("npf_train.csv")
## choose variables whose names end with ".mean" together with "class4"
vars <- colnames(npf)[sapply(colnames(npf),
                             function(s) nchar(s)>5 && substr(s,nchar(s)-4,nchar(s))==".mean")]
npf <- npf[,c(vars,"class4")]
## strip the trailing ".mean" to make the variable names prettier
colnames(npf)[1:length(vars)] <- sapply(colnames(npf)[1:length(vars)],
                                        function(s) substr(s,1,nchar(s)-5))
vars <- colnames(npf)[1:length(vars)]
```

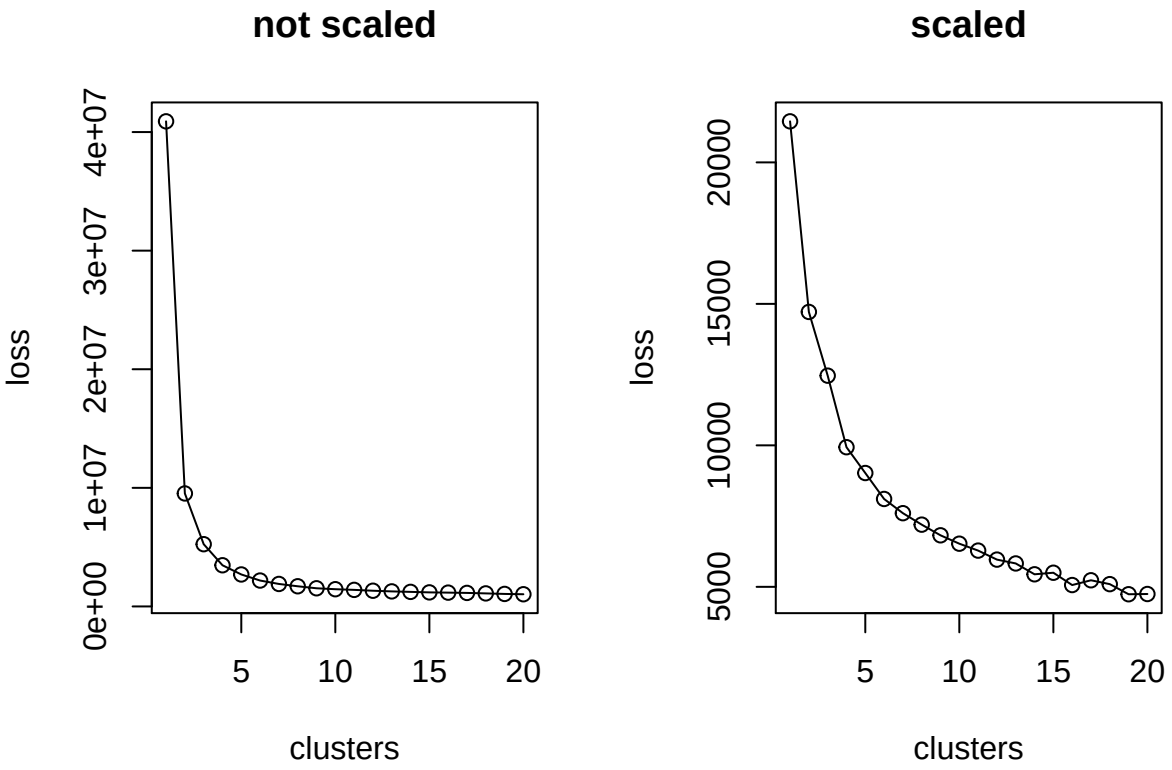
Unless otherwise instructed, please scale the variables to zero mean and unit variance.

Task a

Cluster the rows of the data matrix and plot the k-means loss as a function of the number of clusters from 1 to 20. Should you normalise the columns and what effect does the normalization of the columns have? You can use a library function such as `kmeans` in R.

```
loss0 <- sapply(1:20,function(i)
  kmeans(npf[,vars],i,iter.max=1000,nstart=20,
         algorithm="Lloyd")$tot.withinss)
loss1 <- sapply(1:20,function(i)
  kmeans(scale(npf[,vars]),i,iter.max=1000,nstart=20,
         algorithm="Lloyd")$tot.withinss)
```

```
par(mfrow=c(1,2))
plot(1:20,loss0,xlab="clusters",ylab="loss",main="not scaled")
lines(1:20,loss0)
plot(1:20,loss1,xlab="clusters",ylab="loss",main="scaled")
lines(1:20,loss1)
```



Above, you can find the k-means losses as a function of number of clusters when the variables have not been normalised (left) and when they have been normalised (right), respectively. When variables have not been normalised then the squared loss is dominated by the few variables with high values and high standard deviation, resulting to the faster drop in the left hand curve. The variables with small numerical values are effectively ignored. This difference is caused mostly by units of measurements (e.g., the pressure is in thousands of millibars while the temperature in tens degrees). Because it makes no sense to compare measurements of differnet units (e.g., pressure vs temperature) directly it makes sense to normalise the variables so that the effect of their absolute magnitude is removed and only the effects due to their correlations remain.

Task b

> By using the Lloyd's algorithm and scaled variables, cluster the rows into four clusters and make a confusion matrix (contingency table where the rows are the true classes, columns the cluster indices, and entries the counts of rows). Order the rows and columns so that the sum of diagonal entries in your contingency table is maximized. Where are most "errors" made, if you would use your clusters as a rudimentary classifier (i.e., you would associate each of the clusters with a class)?
>
> Then try how using some smart initialisation such as kmeans++ affects your results.

```
set.seed(42)

library(clue)

cl <- kmeans(scale(npf[,vars]),
             centers=4,algorithm="Lloyd",nstart=1000,iter.max=100)

## Create confusion matrix between the known classes (class 4) and
## cluster indices.
tt <- table(npf$class4,cl$cluster)
```

```
## Find a permutation of cluster indices such that they
## best match the classes in class4.
tt <- tt[,solve_LSAP(tt,maximum=TRUE)]

print(tt)
```

```
##
##              4   2   1   3
##   Ia         0  16   3   7
##   Ib         0  55  22   6
##   II         0  58  35  13
##   nonevent   3  12  89 111
```

We first cluster the normalised data 1000 times and pick a solution with the smallest loss. Then we match the cluster with the known classes by using the Hungarian algorithms. Cluster number 4 is matched with class `Ia`, clustern number 2 is matched with class `Ib`, cluster number 1 with class `II`, and cluster number 3 with class `nonevent`. If we would the clustering as a classifier we would obtain accuracy of 0.4674419. The classifier is not very good.

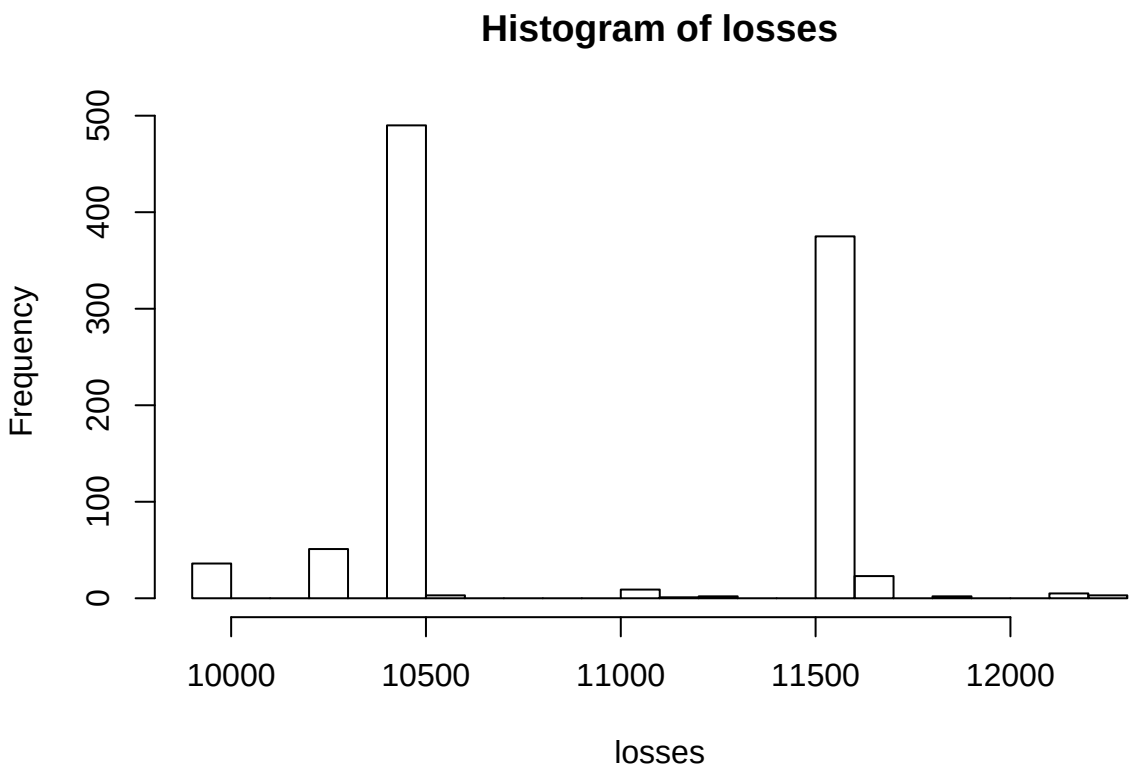
We notice that most of the "errors" are made when the class `nonevent` is "classifed" as `II` (89 occurrences), and when class `II` is classified as `Ib` (58 occurrences).

Task c

> Repeat the clustering of task b above with 1000 different random initialisations (for example, picking in random 4 data vectors for your initial centroids in the Lloyd's algorithm). Make histogram of the losses. What is the minimum and maximum k-means losses for your 1000 random initialisations? How many initialisations would you expect to need to have to obtain one reasonably good loss (say, a solution with a loss within 1% of the best loss out of your 1000 losses), for this dataset and number of clusters?

Lets first do 1000 clustering with random initialisations (4 randomly sampled data vectors as initial prototypes).

```
set.seed(42)

npf_scaled <- scale(npf[,vars])
losses <- sort(replicate(1000,kmeans(npf_scaled,
                                     centers=npf_scaled[sample.int(nrow(npf_scaled),4),],
                                     algorithm="Lloyd",iter.max=100)$tot.withinss))
hist(losses,breaks=20)
```



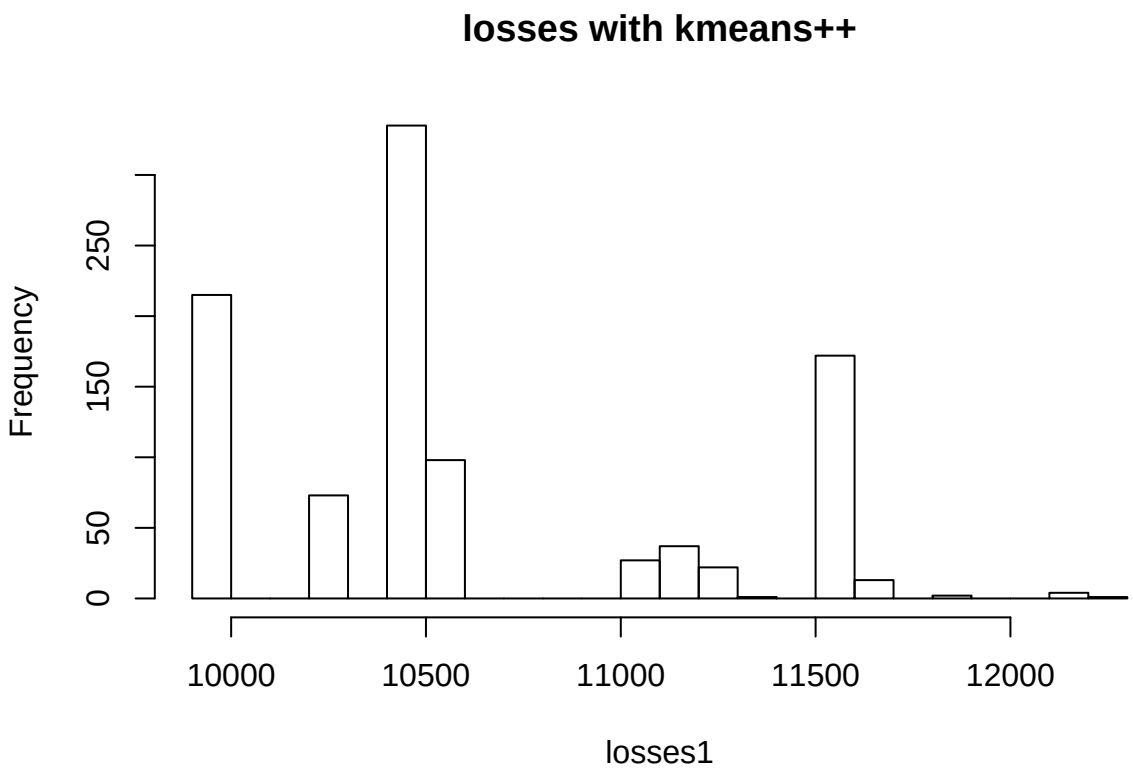
The minimal loss is 9934.2868062 and maximal 1.2230623×10^4.

There are about 36 solutions within 1% of the minimal loss of 9934.2868062. You would therefore expect that about 27.7777778 random initialisations would yield at least one good solution.

```
# kmeansppcenters - Finds initial kmeans++ centroids
# Arguments:
# x        numeric matrix, rows correspond to data items
# k        integer, number of clusters
# Value:
# centers  a matrix of cluster centroids, can be fed to kmeans
#
# Reference: Arthur & Vassilivitskii (2007) k-means++: the
# advantages of careful seeding. In Proc SODA '07, 1027-1035.
kmeansppcenters <- function(x,k) {
  x <- as.matrix(x)
  n <- nrow(x)
  centers <- matrix(NA,k,ncol(x))
  p <- rep(1/n,n)
  for(i in 1:k) {
    centers[i,] <- x[sample.int(n,size=1,prob=p),]
    dd <- rowSums((x-(rep(1,n) %o% centers[i,]))^2)
    d <- if(i>1) pmin(d,dd) else dd
    if(max(d)>0) { p <- d/sum(d) }
  }
  centers
}
```

```
set.seed(42)
losses1 <- sort(replicate(1000,kmeans(npf_scaled,
                                      centers=kmeansppcenters(npf_scaled,4),
```

```
                                        algorithm="Lloyd",iter.max=100)$tot.withinss))
hist(losses1,breaks=20,main="losses with kmeans++")
```



With k-means++ initialisation, the minimal loss is 9934.2868062 and maximal 1.2230623×10^4.

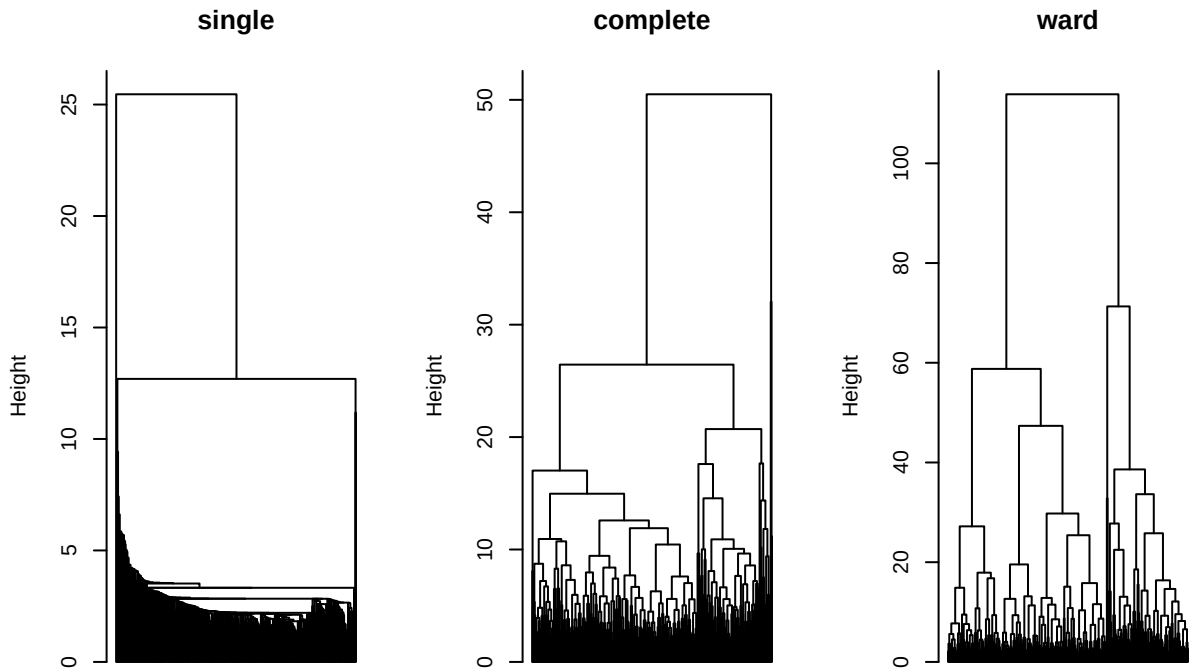
With k-means++, there are about 215 solutions within 1% of the minimal loss of 9934.2868062. You would therefore expect that about 4.6511628 random initialisations would yield at least one good solution.

Above, we also see the typical behaviour of single linkage clustering which tends to for a "chain" of clusters by merging singletons to a larger cluster.

Task d

> Try clustering the same data with agglomerative hierarchical clustering with at least 2 different linkage functions. Produce a dendrogram and corresponding flat clustering (e.g., by splitting the dendrogram with `cutree`) and compare their properties (e.g., comparing sizes of clusters, by looking at confusion matrices). Find and report at least one interesting feature or reproduce some of the properties of hierarchical clustering (e.g., differences between the linkage functions) discussed in the lecture. (Hint: See Sections 10.5.2 and 10.6.2 of James et al. for examples in R.)

```
d <- dist(scale(npf[,vars]))
hc.min <- hclust(d,"single")
hc.max <- hclust(d,"complete")
hc.ward <- hclust(d,"ward.D2")
par(mfrow=c(1,3))
plot(hc.min,labels=FALSE,hang=-1,main="single")
plot(hc.max,labels=FALSE,hang=-1,main="complete")
plot(hc.ward,labels=FALSE,hang=-1,main="ward")
```




```
hc.min4 <- cutree(hc.min,k=4)
hc.max4 <- cutree(hc.max,k=4)
hc.ward4 <- cutree(hc.ward,k=4)

tt.min4 <- table(cl$cluster,hc.min4)
tt.min4 <- tt[,solve_LSAP(tt.min4,maximum=TRUE)]
tt.max4 <- table(cl$cluster,hc.max4)
tt.max4 <- tt[,solve_LSAP(tt.max4,maximum=TRUE)]
tt.ward4 <- table(cl$cluster,hc.ward4)
tt.ward4 <- tt[,solve_LSAP(tt.ward4,maximum=TRUE)]
```

Above, you can find the hiearchical clustering using single, complete, and Ward linkage respectively. Cut the tree for each clusterings so that we end up with 4 flat clusters and compare them to the previous k-means result by using the Hungarian algorithm. The sum of diagonals of the confusion matrices between different hierarchical clustering variants and k-means are as follows: single 0.1232558, complete 0.2581395, Ward 0.3930233. As expectd, the Ward method gives us a solution that is closest to the k-means result

Grading

Points: max. 25 (a: 6 b: 6 c: 6 d: 7)

Problem 19

[25% points]

Objectives: uses of PCA

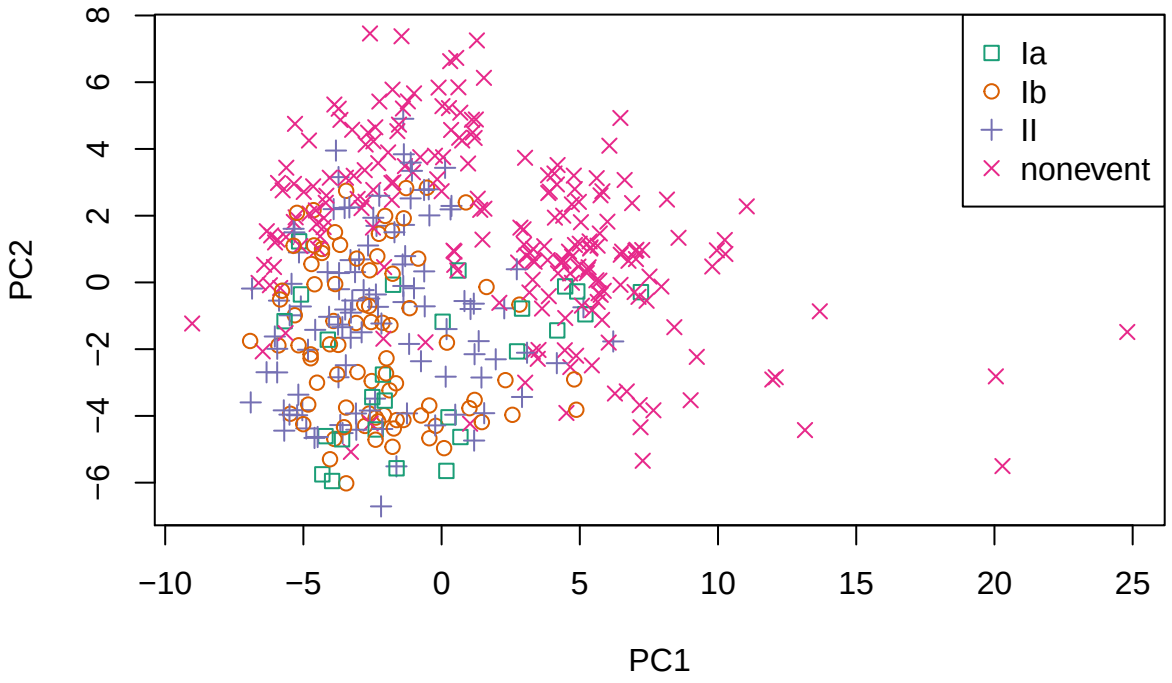
Continue with the same dataset as in Problem 18 above.

Task a

Make a PCA projection of the data into two dimensions. Indicate the class index (`class4`), e.g., by color and/or the shape of the glyph. Be sure to indicate which color/shape corresponds to which class, e.g., by `legend`.

```
pr <- prcomp(scale(npf[,vars]))
pr0 <- prcomp(npf[,vars],scale.=FALSE)
```

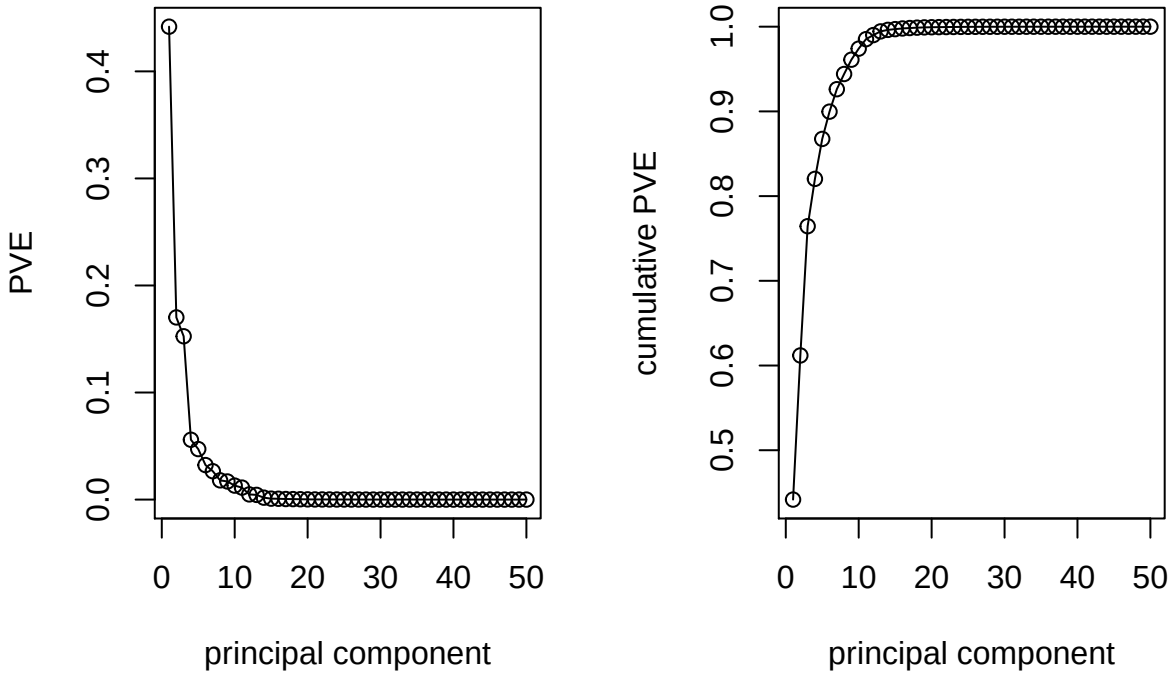
```
plot_npf <- function(x,...) {
  ## https://colorbrewer2.org/?type=qualitative&scheme=Dark2&n=4
  cols4 <- c("#1b9e77","#d95f02","#7570b3","#e7298a")
  plot(x[,1:2],col=cols4[npf$class4],pch=c(0,1,3,4)[npf$class4],...)
  legend("topright",levels(npf$class4),
         col=cols4,pch=c(0,1,3,4))
}
plot_npf(pr$x)
```



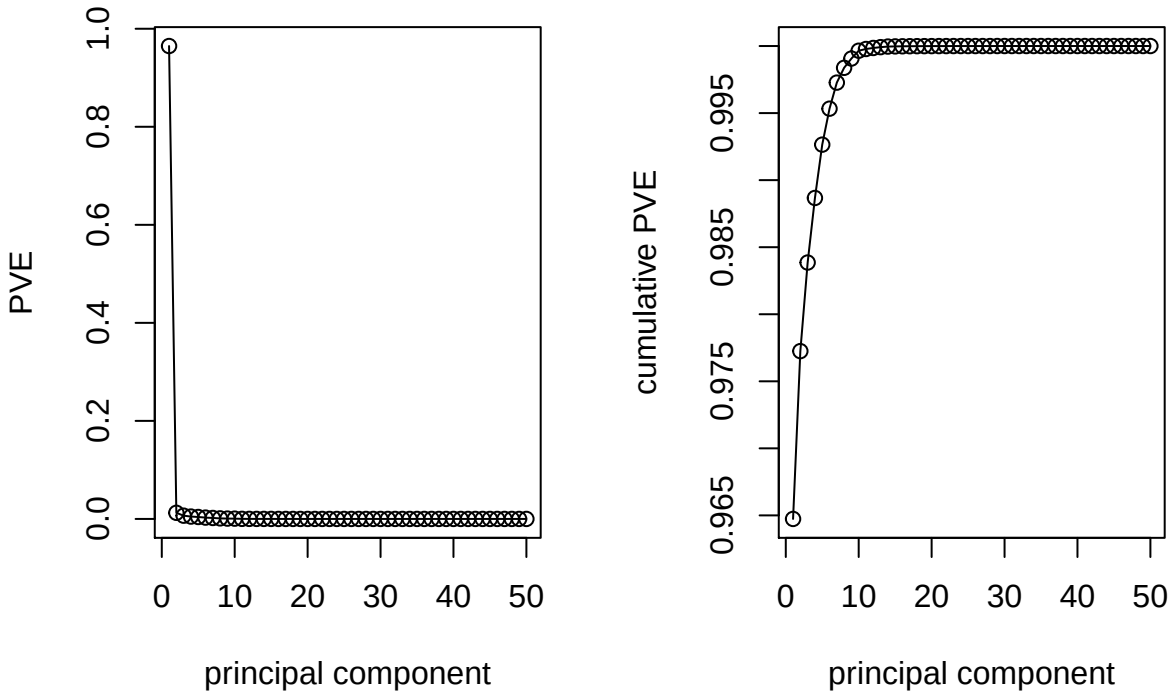
Task b

Compute and plot the proportion of variance explained and the cumulative variances explained for the principal components. Study the effects of different normalisations - at least compare difference if you do not scale the data at all vs. you normalize each variable to zero mean and unit variance! Why for unnormalized data it seems that fewer components explain a large proportion of the variance, as compared to the normalized data? (Hint: See Sec. 10.4 Fig 10.4 of James et al.)

```
par(mfrow=c(1,2))
plot(1:length(pr$sdev),pr$sdev^2/sum(pr$sdev^2),xlab="principal component",ylab="PVE")
lines(1:length(pr$sdev),pr$sdev^2/sum(pr$sdev^2))
plot(1:length(pr$sdev),cumsum(pr$sdev^2/sum(pr$sdev^2)),xlab="principal component",ylab="cumulative PVE
lines(1:length(pr$sdev),cumsum(pr$sdev^2/sum(pr$sdev^2)))
```




```
par(mfrow=c(1,2))
plot(1:length(pr0$sdev),pr0$sdev^2/sum(pr0$sdev^2),xlab="principal component",ylab="PVE")
lines(1:length(pr0$sdev),pr0$sdev^2/sum(pr0$sdev^2))
plot(1:length(pr0$sdev),cumsum(pr0$sdev^2/sum(pr0$sdev^2)),xlab="principal component",ylab="cumulative
lines(1:length(pr0$sdev),cumsum(pr0$sdev^2/sum(pr0$sdev^2)))
```



Above, you can find plots of proportion of variance explained (PVE, left) and cumulative PVE (right) for normalized (top) and unnormalized (bottom) data, respectively.

As with k-means clustering (see Problem 18a) if we do not normalize variance then we get a dominant contribution with variables that have high numerical values (because of the choice of units) and those variables

that have low numerical values may essentially be ignored.

Task c

> Pick one classification algorithm that is implemented in R or SciPy or in your other favourite environment that would work with this data and choose one of the challenge performance measures (binary accuracy, multiclass accuracy, or perplexity). Split the data in random into training and validation sets of equal sizes. Train your classification algorithm first without the dimensionality reduction on the training set and report the performance (=your chosen performance measure) on the validation set. Do the same on the data where the dimensionality has been reduced by the PCA (see task b above). How does the performance of your classifier vary with the (reduced) dimensionality and is there an "optimal" dimensionality which gives you the best performance on the validation set?

```
i_tr <- sample.int(nrow(npf),nrow(npf) %/% 2) # training set
i_va <- setdiff(1:nrow(npf),i_tr) # validation set
npf$class2 <- factor("nonevent",levels=c("nonevent","event"))
npf$class2[npf$class4!="nonevent"] <- "event"
```

```
npf.pr <- prcomp(scale(npf[,vars]))

acc <- sapply(1:50,function(i) {
  mean(ifelse(predict(glm(class2 ~ .,
                          data.frame(npf.pr$x[i_tr,1:i,drop=FALSE],
                                     class2=npf[i_tr,"class2"]),
                          family=binomial),
                      data.frame(npf.pr$x[i_va,1:i,drop=FALSE]))>=0,
              "event","nonevent")==npf[i_va,"class2"])
})
```

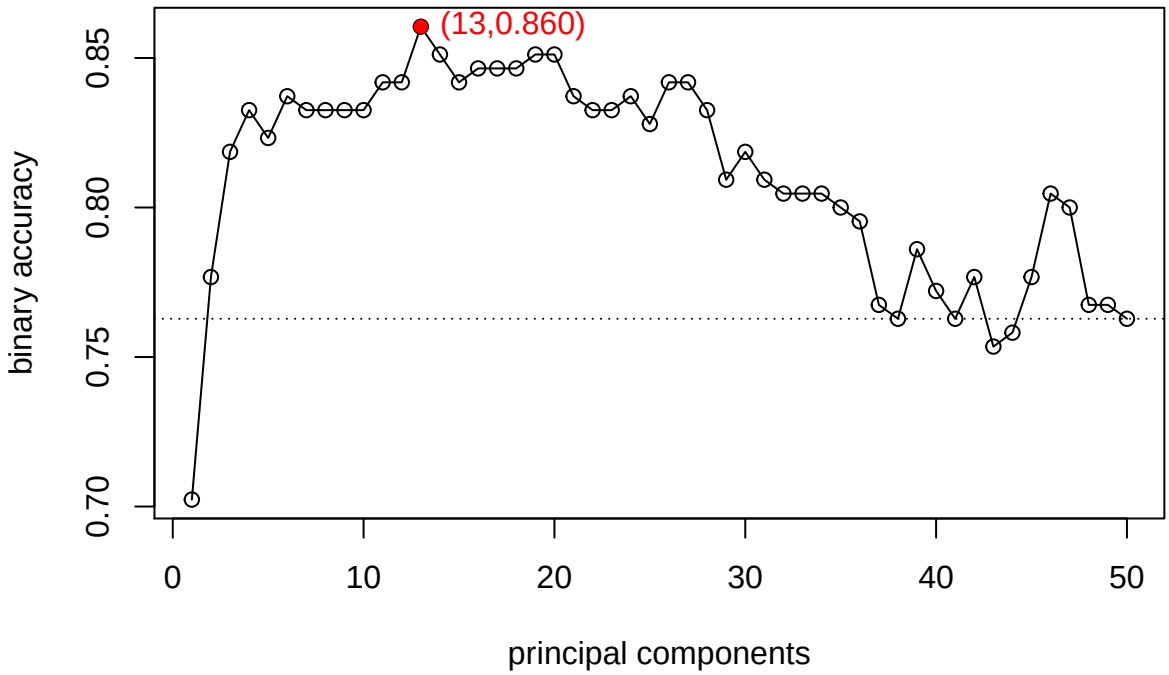
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
acc0 <- mean(ifelse(predict(glm(class2 ~ .,npf[i_tr,c(vars,"class2")],family=binomial),
                            npf[i_va,vars])>=0,
                    "event","nonevent")==npf[i_va,"class2"])
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
plot(1:50,acc,xlab="principal components",ylab="binary accuracy")
lines(1:50,acc)
i <- which.max(acc)
points(i,acc[i],pch=16,col="red")
text(i,acc[i],sprintf("(%d,%.3f)",i,acc[i]),pos=4,col="red")
abline(h=acc0,lty="dotted")
```



We choose logistic regression and binary classification accuracy and split the dataset in random into training and validation sets of equal sizes. The best validation set accuracy 0.8604651 is then given with 13 principal components. Without any dimensionality reduction the accuracy would be 0.7627907.

Task d (optional)

This is a bonus tasks for which no points are awarded.

> Repeat task a above, but this time with isometric multidimensional scaling (MDS) and t-distributed stochastic neighbor embedding (t-SNE).

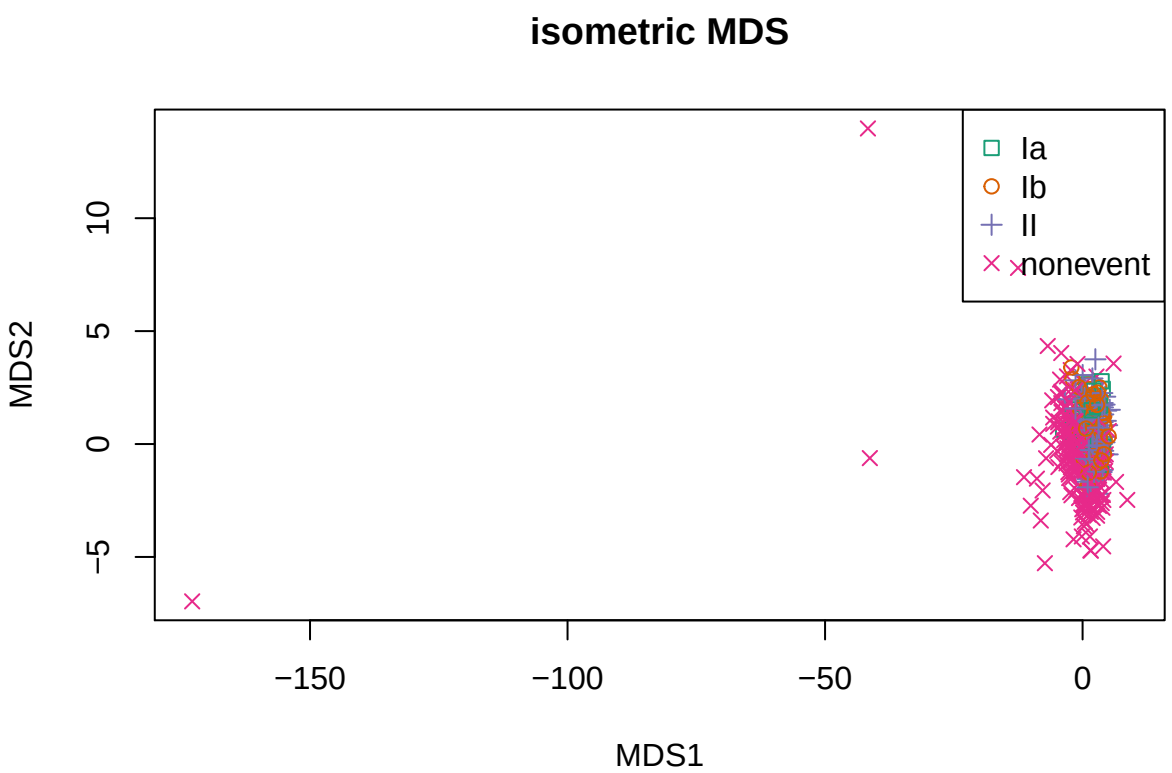
```
library(MASS)
library(tsne)
d <- dist(scale(npf[,vars]))
## cmdscale essentially does PCA. Both MDS and t-SNE are sensitive to initial
## configuration and the PCA initialisation generally leads to reasonable convergence.
## (Even though R isoMDS and tsne can handle bad initial config fine, we are being explicit here.)
y <- cmdscale(d,2)
## isometric MDS embedding coordinates
x.mds <- isoMDS(d,y=y)$points
```

```
## initial  value 16.386354
## iter   5 value 11.751958
## iter  10 value 10.773663
## iter  15 value 10.119689
## iter  20 value 9.548696
## iter  25 value 9.242584
## iter  30 value 9.001601
## iter  35 value 8.663005
## iter  40 value 7.997339
## iter  45 value 7.071515
## iter  50 value 6.713433
## final  value 6.713433
## stopped after 50 iterations
```

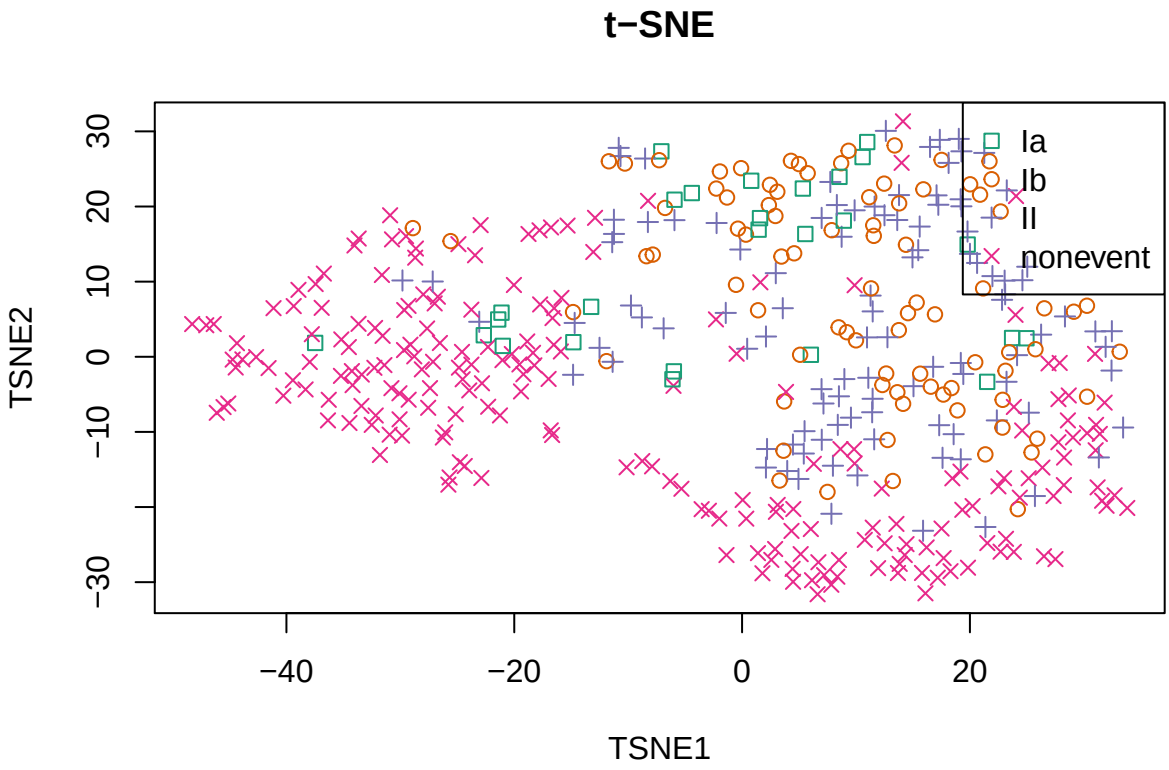
```
## t-SNE embedding coordinates
x.tsne <- tsne(d,initial_config=y,k=2)
```

```
## sigma summary: Min. : 0.529930984735273 |1st Qu. : 0.712023880116945 |Median : 0.770082433119843 |Mea
## Epoch: Iteration #100 error is: 0.557476940277133
## Epoch: Iteration #200 error is: 0.54243563263473
## Epoch: Iteration #300 error is: 0.537310241835897
## Epoch: Iteration #400 error is: 0.535269343159296
## Epoch: Iteration #500 error is: 0.534577516947113
## Epoch: Iteration #600 error is: 0.534223715864371
## Epoch: Iteration #700 error is: 0.534021907782654
## Epoch: Iteration #800 error is: 0.533887176630687
## Epoch: Iteration #900 error is: 0.533793086371863
## Epoch: Iteration #1000 error is: 0.533724025573137
```

```
plot_npf(x.mds,main="isometric MDS",xlab="MDS1",ylab="MDS2")
```




```
plot_npf(x.tsne,main="t-SNE",xlab="TSNE1",ylab="TSNE2")
```



Grading

Points: max. 25 (a: 8 b: 8 c: 9 d: 0)

Problem 20

[10% points]

Objectives: self-reflection, giving feedback of the course

Task a

Write a learning diary of the topics of lectures 1-12 and exercise sets 1-3.

The length of your reply should be 3-6 paragraphs of text. Guiding questions: What did I learn? What did I not understand? Was there something relevant for other studies or (future) work? Notice that this entry should typically be longer than your earlier learning diary entries in E1 and E2.

Grading

Points: max. 10

Please give full points if there is at least 3 paragraphs of the text and the discussions shows some insight. Please use the grading matrix as a guideline.