

Mājas darbs 2-2: dimensiju redukcija

2. Uzdevums - dimensiju redukcija ar algoritmu pēc izvēles (PaCMAP)

Algoritma realizācija

Izvēlēts PaCMAP algoritms, jo, lasot [kursa materiālos ielikto rakstu](#), lai saprastu visus piedāvātos algoritmus un atšķirības starp tiem, mājas darba autoram radusies izteikta pārliecība, ka tas tik tiešām varētu būt labākais no dimensiju redukcijas algoritmiem.

Atšķirībā no t-SNE, kam platformā R pieejama implementācija, ko var tiešā veidā instalēt kā R pakotni, PaCMAP pagaidām, šķiet, pieejams tikai Python. Taču tas nav šķērslis rakstot mājas darbu R markdown platformā, jo starp R pakotnēm ir pieejama Python saskarne. Tā kā pamatā visas apjomīgas darbības ar datiem gan R, gan Python veic ar C/C++ rakstītām skaitļošanas bibliotēkām, pēc saskarnes izveidošanas pārējais kods izskatās pēc parasta R skripta - Python modulim tiek izveidots atbilstošs R objekts ar visām tā metodēm un atribūtiem. Jāseko līdzī tikai datu tipu sakritībai - lai Python saprastu ienākošos datus kā *numpy.ndarray* objektu, nepieciešams R *dataframe* objektu izteikt matricas formā. Tāpat nepieciešams konkrēti norādīt, ja kāda skaitliska vērtība ir vesels, nevis reāls skaitlis.

Datu apstrāde

Izveido saskarni ar Python vidi (nepieciešams pirms tam instalēt *pacmap* pakotni izvēlētajai Python izpildvidei):

```
library(reticulate)
use_virtualenv("~/repos/homework/DIA_MD/.venv")
pacmap <- import("pacmap")
```

Ielasa datus:

```
library(foreign)
data <- read.arff("ionosphere.arff")
```

Deklarē funkcijas datu apgriešanai, klašu attēlošanai krāsās un PaCMAP algoritma izsaukšanai:

```
trim <- function(df){
  df[,1:length(df[1,])-1]
}

colors <- function(df) {
  sapply(df, function(x) {
    if (x == "b") {
      "blue"
    } else {
      "red"
    }
  })
}

pac_wrapper <- function(mat, col,init="pca", k=10, MN_ratio=0.5,
                        FP_ratio=2, var="default") {
  embedding <- pacmap$PaCMAP(MN_ratio = MN_ratio,FP_ratio = FP_ratio,
                             n_neighbors = as.integer(k))
  fit<-embedding$fit_transform(mat, init=init)
  plot(fit,col=col,xlab="comp 1",ylab="comp 2",
       main=sprintf("PaCMAP variable = %s init = %s k = %d MN = %.2f FP = %.2f",
                    var, init, k, MN_ratio, FP_ratio))
}
```

Apgriež datus, saglabā krāsas:

```
col_row <- colors(data[,length(data[1,])])  
mat <- as.matrix(trim(data))
```

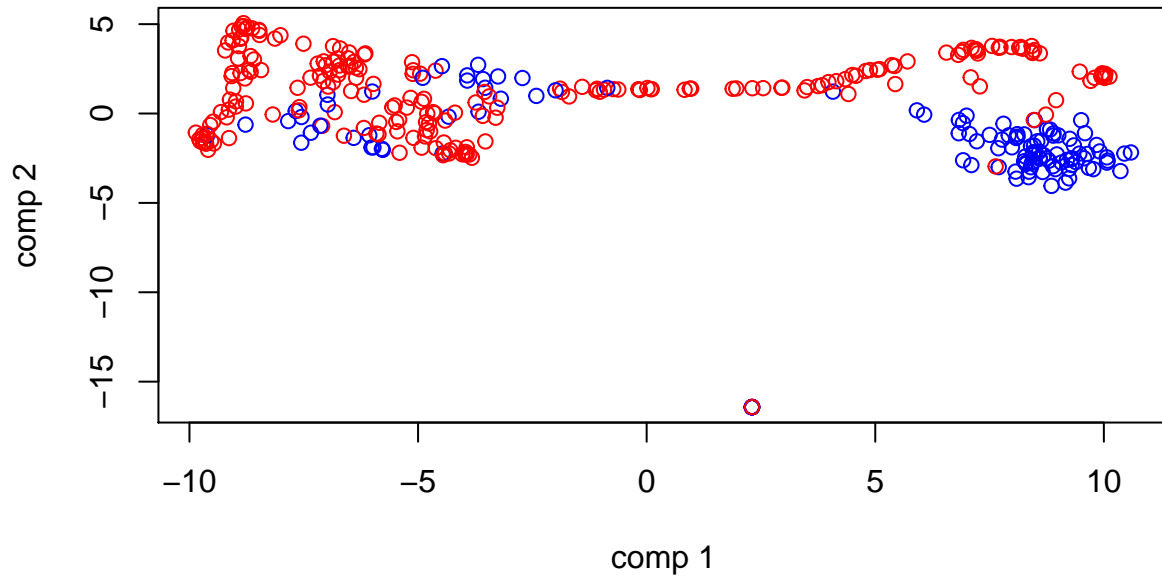
Algoritmam pieejami sekojošie hiperparametri:

- *init* - 2-dimensionālā punktu mākoņa sākotnējo vērtību iegūšanas metode. Pēc noklusējuma PCA, iespējams iegūt arī nejauši ģenerētas vērtības. Principā algoritms ir robusts pret inicializāciju, taču PCA it kā sniedzot nedaudz labākus rezultātus un ātrāku konvergenci. Citiem algoritmiem (konkrēti, TriMap un UMAP) inicializācijas metodei ir liela nozīme, jo optimizācijas process nespēj tikt galā ar zināmiem izkliedes režīmiem (tāli punkti pārāk tuvu, u.t.t);
- *n_neighbours* - tuvāko kaimiņu skaits. Šie ir punkti, pēc kuriem optimizācijas solī tiek rēķināta *loss* funkcijas kaimiņu komponente (tuvāk - labāk). Izvēlēti, aprēķinot *n_neighbours* + 50 tuvākajiem kaimiņiem pēc Eiklīda distances speciālu normalizēto distanci un izvēloties pēc mazākās;
- *MN_ratio* - koeficients, kā nosaka vidēji tālo punktu skaitu. Katram punktam nejauši atrod *n_neighbours* * *MN_ratio* vidēji tālos punktus, pēc kuriem rēķina *loss* vidēji tālo komponenti (sākumā tuvāk - labāk. Pēc tam ignorē);
- *FP_ratio* - koeficients tālo punktu skaitam. Pēc tālajiem punktiem rēķina tālo punktu komponenti (tālāk - labāk).

Uzzīmējot grafikus, pēc kārtas mainot katru no hiperparametriem:

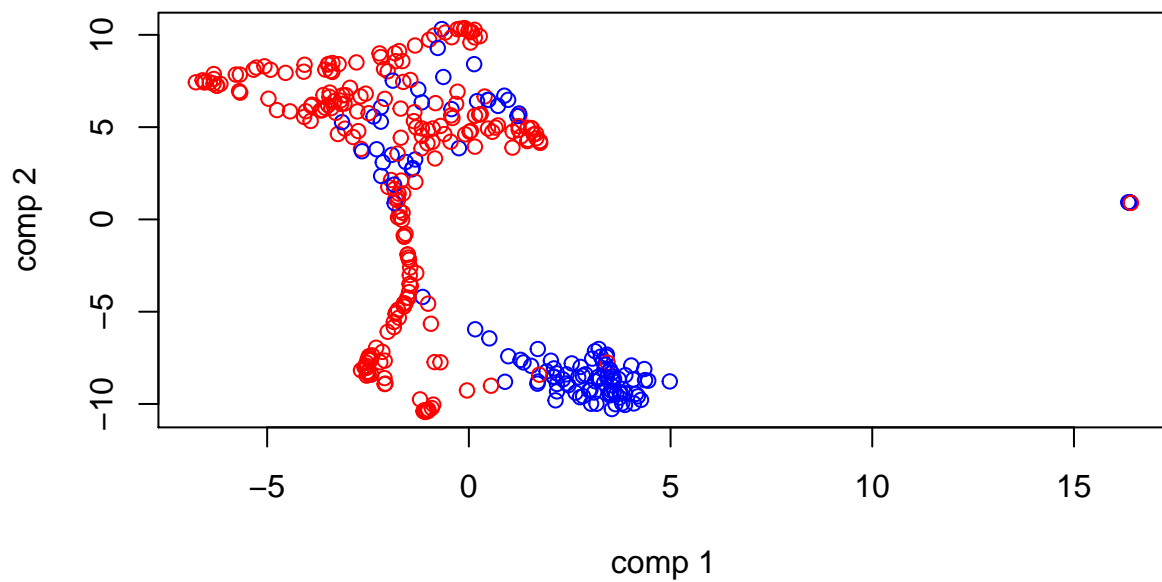
```
pac_wrapper(mat, col_row, init="pca", var="init") # pca is default
```

PaCMAP variable = init init = pca k = 10 MN = 0.50 FP = 2.00



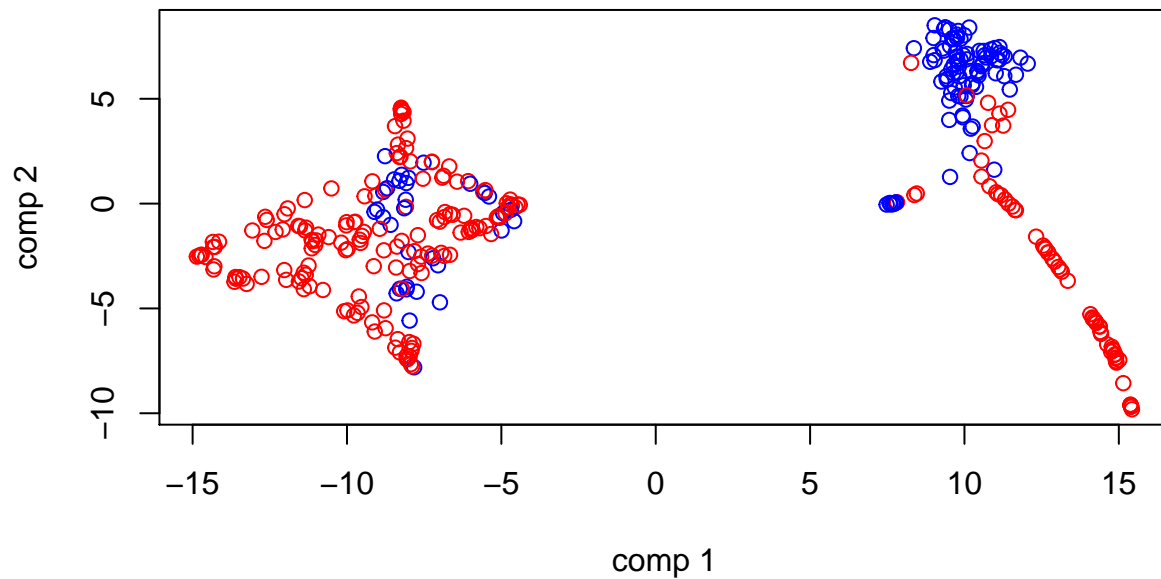
```
pac_wrapper(mat, col_row, init="random", var="init")
```

PaCMAP variable = init init = random k = 10 MN = 0.50 FP = 2.00



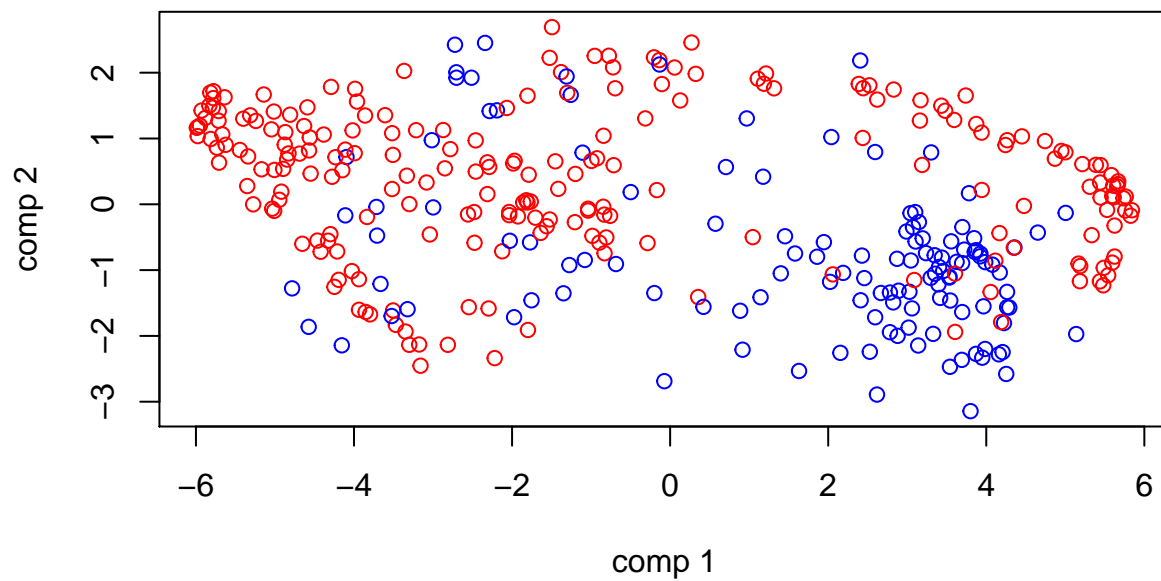
```
pac_wrapper(mat, col_row, k=5, var="k")
```

PaCMAP variable = k init = pca k = 5 MN = 0.50 FP = 2.00



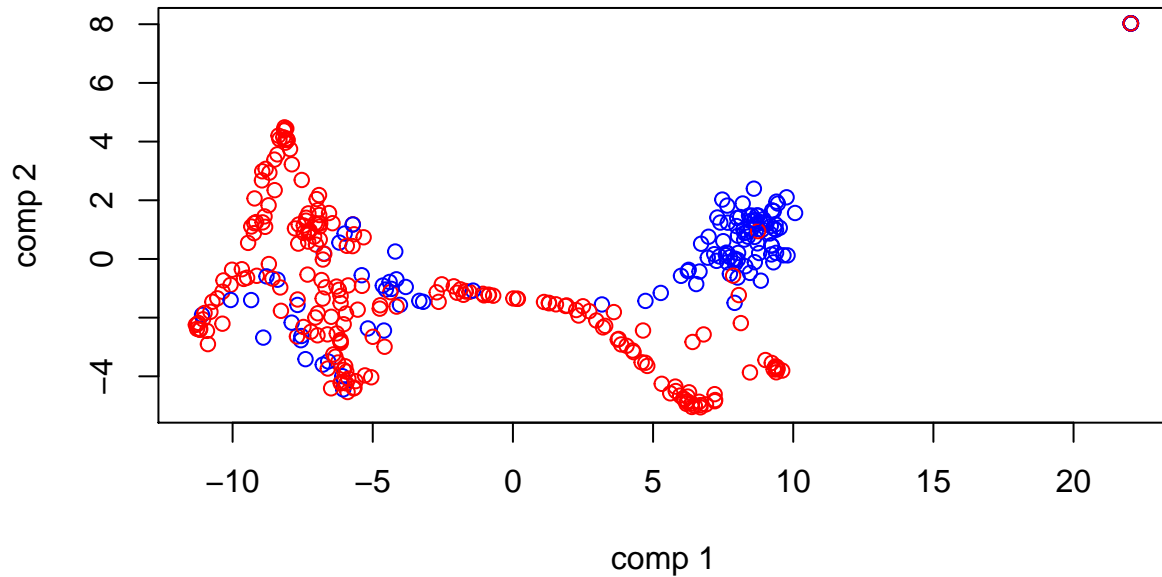
```
pac_wrapper(mat, col_row, k=50, var="k")
```

PaCMAP variable = k init = pca k = 50 MN = 0.50 FP = 2.00



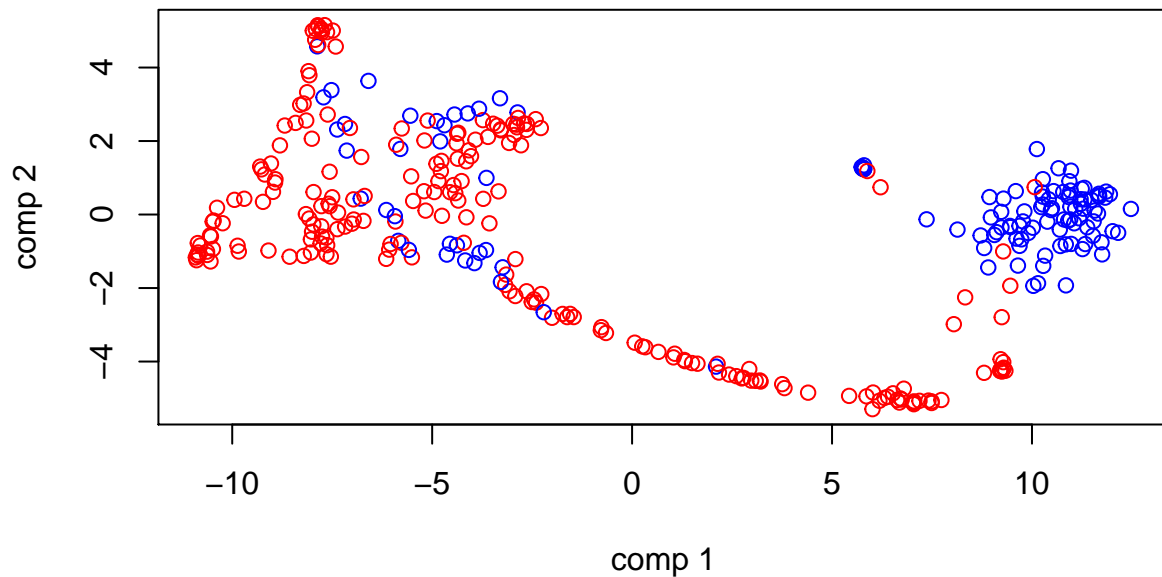
```
pac_wrapper(mat, col_row, MN_ratio=0, var="MN")
```

PaCMAP variable = MN init = pca k = 10 MN = 0.00 FP = 2.00



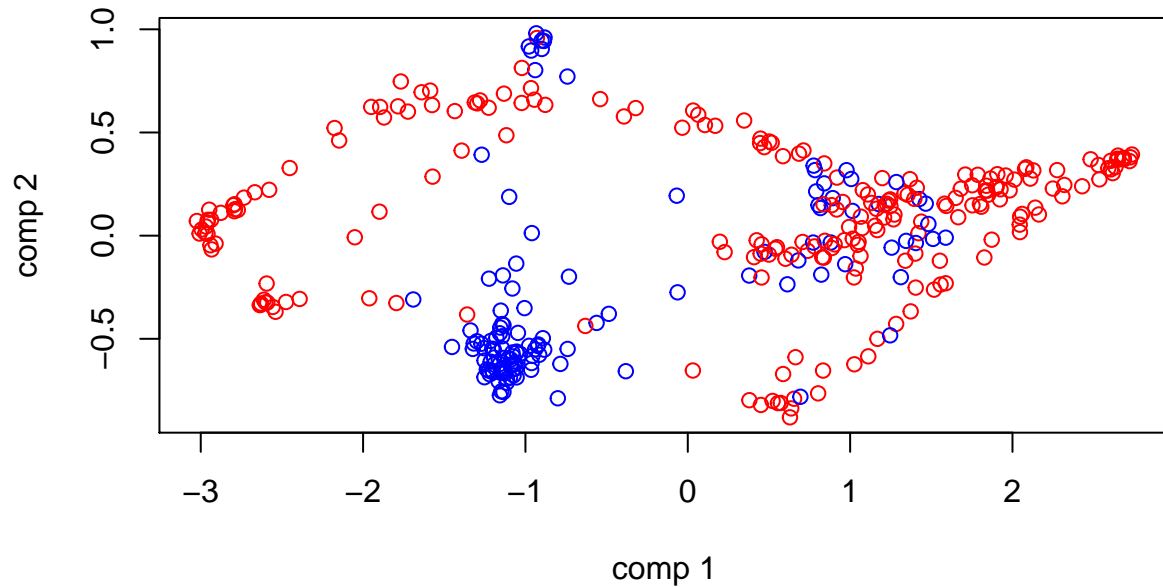
```
pac_wrapper(mat, col_row, MN_ratio=2, var="MN")
```

PaCMAP variable = MN init = pca k = 10 MN = 2.00 FP = 2.00



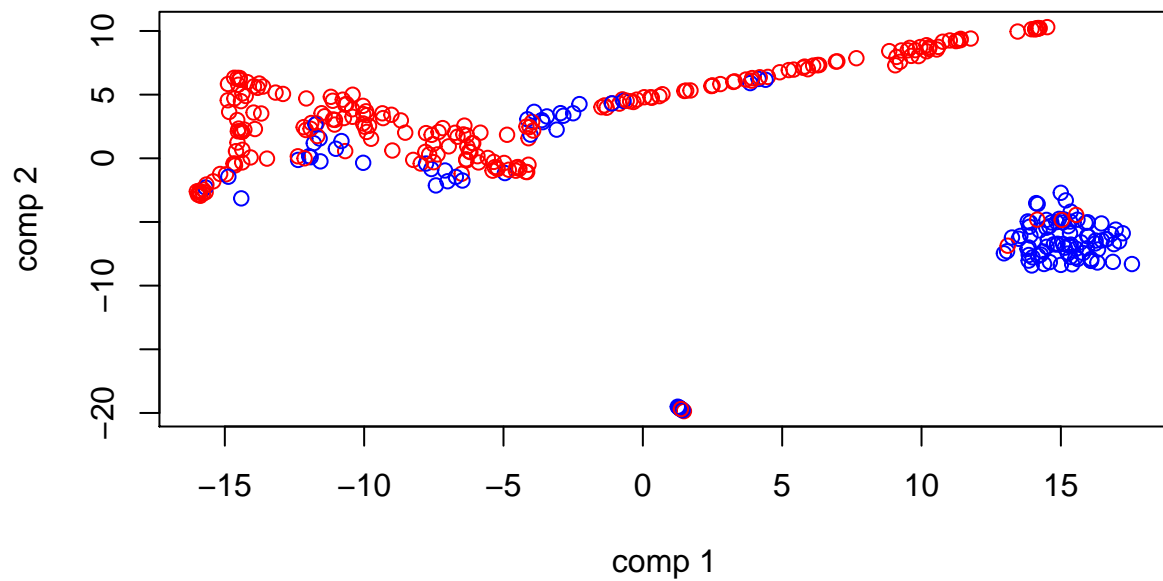
```
pac_wrapper(mat, col_row, FP_ratio=0.1, var="FP") # can't be less than 1
```

PaCMAP variable = FP init = pca k = 10 MN = 0.50 FP = 0.10



```
pac_wrapper(mat, col_row, FP_ratio=4, var="FP")
```

PaCMAP variable = FP init = pca k = 10 MN = 0.50 FP = 4.00



Secinājumi

Grūti spriest par iegūtā rezultāta kvalitāti, salīdzinot ar t-SNE, jo mājas darba autoram nekas nav zināms par apstrādājamo datu kopu, taču var novērot, ka rezultāts ir atšķirīgs - sarkanā klase vēl joprojām veido struktūru ar klasteri un garu “asti”, taču zilā klase pie gandrīz visām hiperparametru kombinācijām veido divus izteiktus klasterus.

Par hiperparemetriem var secināt sekojošo:

- *init* - īpašas nozīmes nav. Vismaz šai datu kopai, rezultāti izskatās līdzīgi neatkarīgi no inicializācijas metodes;
- *n_neighbours* - ietekmē visu trīs apstrādāto punktu klašu izmērus. Pie pārāk maza apstrādāto punktu skaita, datu veido nošķirtus klasterus - netiek pietiekami “savilkti kopā” iterācijas procesā. Pie pārāk liela, rezultējošais punktu mākonis ir daudz “izpludinātāks” - skaidri redzamas struktūras sāk pazust un punkti vienmērīgi izkliedējas plaknē;
- *MN_ratio* - vizuāli grūti spriest, taču iespējams, ka parādās līdzīga sakarība kā ar *n_neighbours*, tikai mazāk izteikti (pievērsot uzmanību klasteru blīvumam un formulai, kas dota rakstā);
- *FP_ratio* - pie neliela tālo punktu skaita, klasteri vēl joprojām parādās, taču tiek sliktāk nošķirti. Pie liela tālo punktu skaita, klasteri tiek ļoti tālu aizbīdīti viens no otra, iespējams - pārāk tālu.