Mājas darbs 2-2: dimensiju redukcija

1. Uzdevums - dimensiju redukcija ar algoritmu pēc izvēles (t-SNE)

Datu apstrāde

Algoritms t-SNE izvēlēts kā pirmais no dotajiem un tāpēc, ka pieejama realizācija R platformā.

Vispirms ielasa datus:

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

Veic priekšapstrādi, lai nogrieztu klases un pēc tam varētu tās grafiski attēlot kā punktu krāsas:

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

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

trimmed <- trim(data)
col_row <- colors(data[,length(data[1,])])
```

Kods t-SNE aprēķinam un grafiskai attēlošanai:

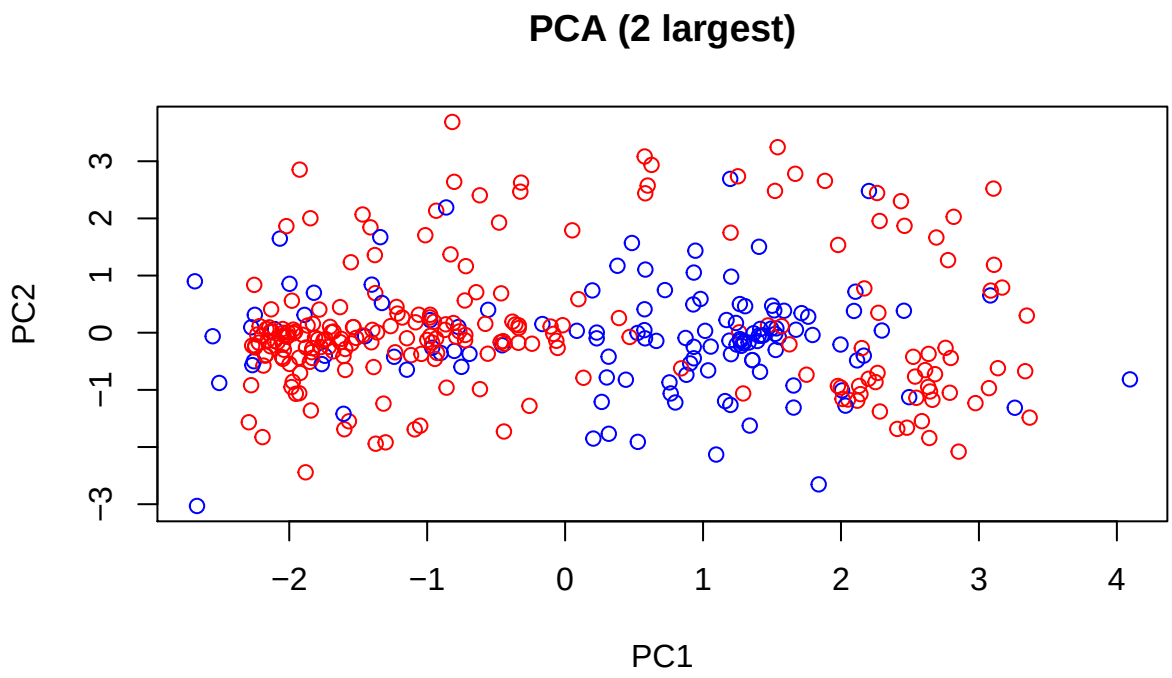
```
library(tsne)
plot.tsne <- function(df, perp=30, iter=400, class=NULL, plt=T, k=2, ret=F) {
  transformed <- tsne(df, k = k, perplexity = perp,
                      initial_dims = length(df[1,]), max_iter = iter)
  x <- transformed[,1]
  y <- transformed[,2]
  if (plt) {
    plot(x,y,col=class,xlab="comp1",
         ylab="comp2",
         main=sprintf("t-SNE perplexity = %d iter = %d", perp, iter))
  }
  if (ret) {
    transformed
  }
}
```

Kods PCA aprēķinam un attēlošanai (salīdzināšanas mērķiem):

```
plot.pca <- function(df, class){
  pca <- prcomp(df)
  plot(pca$x[,1:2],col=class,main="PCA (2 largest)")
  s <- summary(pca)
  s$importance[2,1:2]
}
```

PCA ar divām galvenajām komponentēm, 2 galveno komponenšu dispersijas proporcija:

```
v<-plot.pca(trimmed, col_row)
```




```
v
```

```
##     PC1     PC2
## 0.31344 0.12272
```

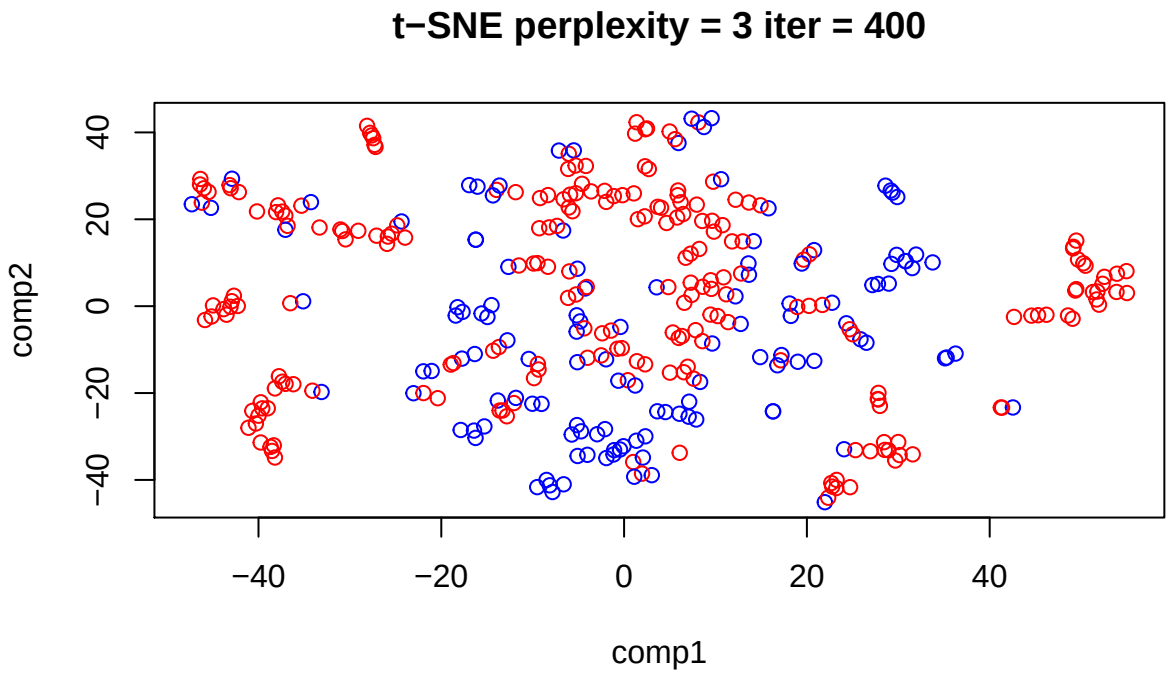
```
sum(v)
```

```
## [1] 0.43616
```

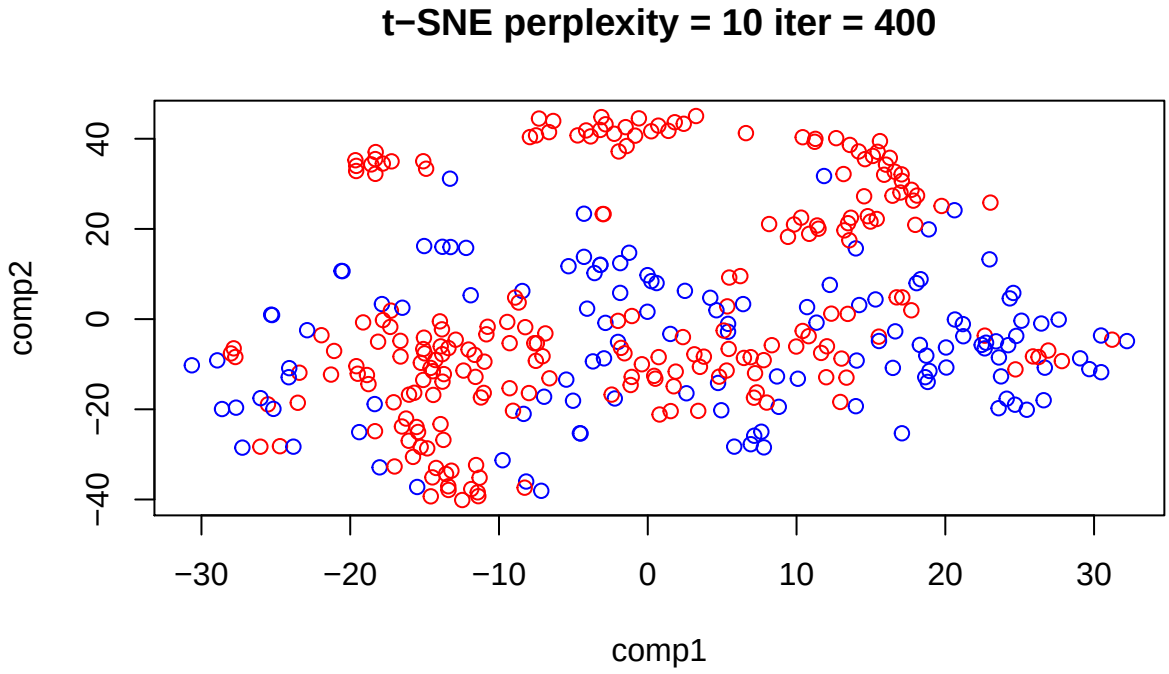
Šai t-SNE realizācijai pieejami divi galvenie hiperparametri - *perplexity* (konstante, pret kuru tiek skaitliski risināts vienādojums $2^{-\sum_j p_{j|i} log(p_{j|i})} = perplexity$, kur $p_{j|i}$ - izteiksme atkarīga no distancēm starp punktiem ar brīvo mainīgo σ_i; iegūtās nosacītās varbūtības tiek izmantotas, lai iegūtu līdzības metriku) un *max_iter* - gradientu optimizācijas algoritma iterāciju skaits. Funkcija automātiski drukā *loss* vērtību pēc katrām 100 optimizācijas algoritma iterācijām un uzreiz redzams, ka visām izvēlētām *perplexity* vērtībām *loss* vērtība ir faktiski minimizēta jau pēc 200 iterācijām. Pēc noklusējma iterāciju skaits ir 1000, šeit atstāts 400 un sīkāk nav pētīts, jo lielas izmaiņas nav novērotas. Ir pieejams arī *min_cost* parametrs, kas ļauj pārtraukt darbu pie izvēlētas *loss* vērtības, taču atkarībā no *perplexity* izskatās, ka tām ir dažādas asimptotes - tāpēc mainot *perplexity* šis parametrs nav īpaši noderīgs.

Zīmējot grafikus ar *perplexity* vērtībām {3,10,20,30,50,100}:

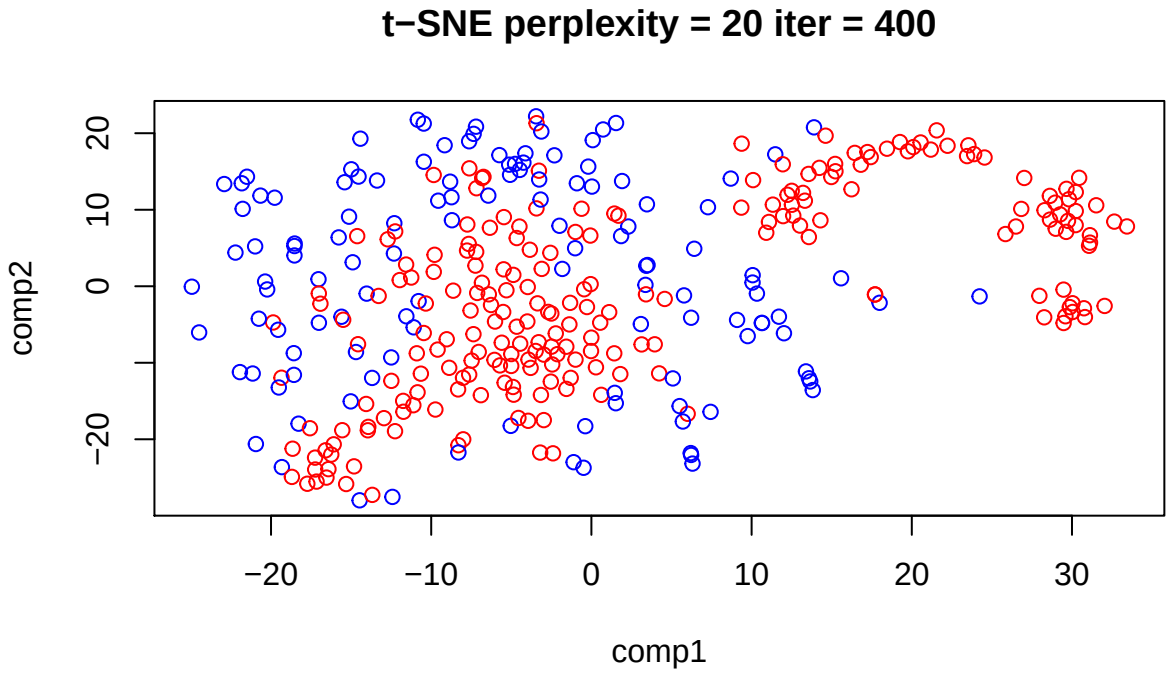
```
plot.tsne(trimmed,3,class=col_row)
```



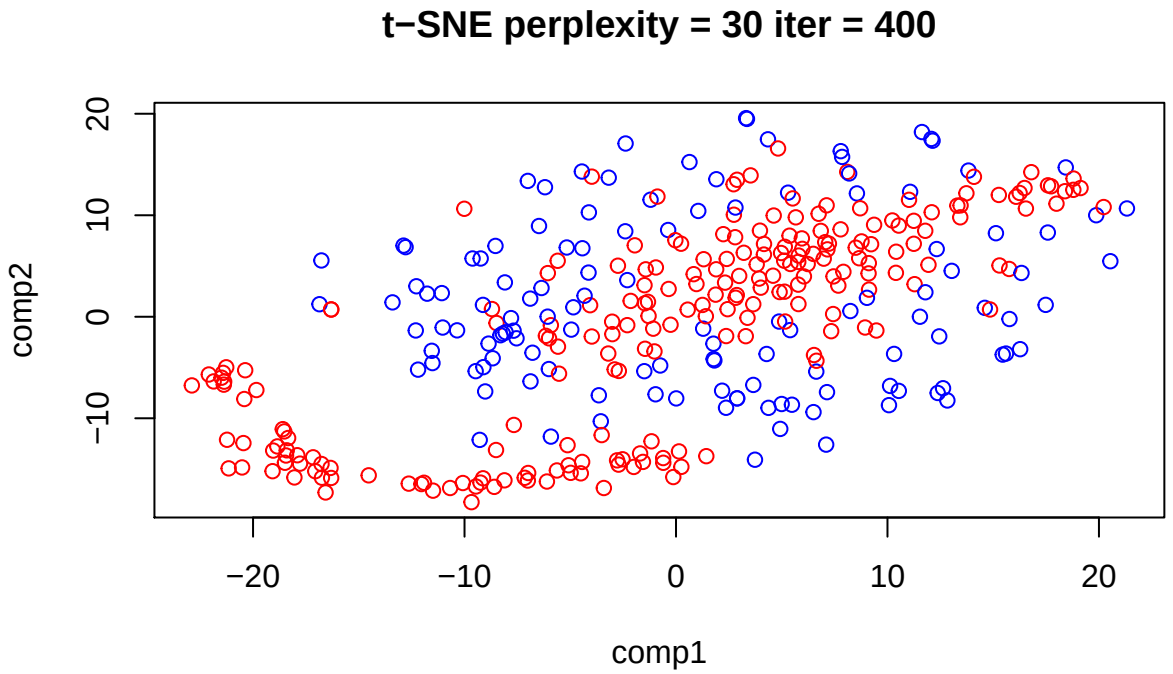

```
plot.tsne(trimmed,10,class=col_row)
```



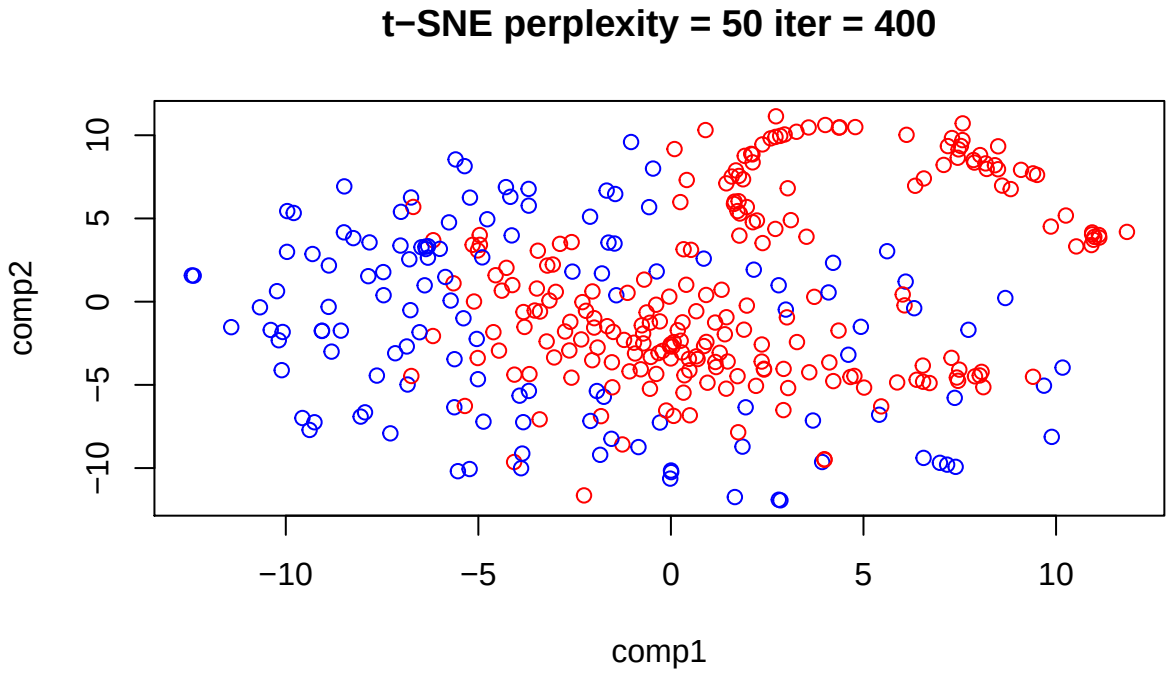
```
plot.tsne(trimmed,20,class=col_row)
```



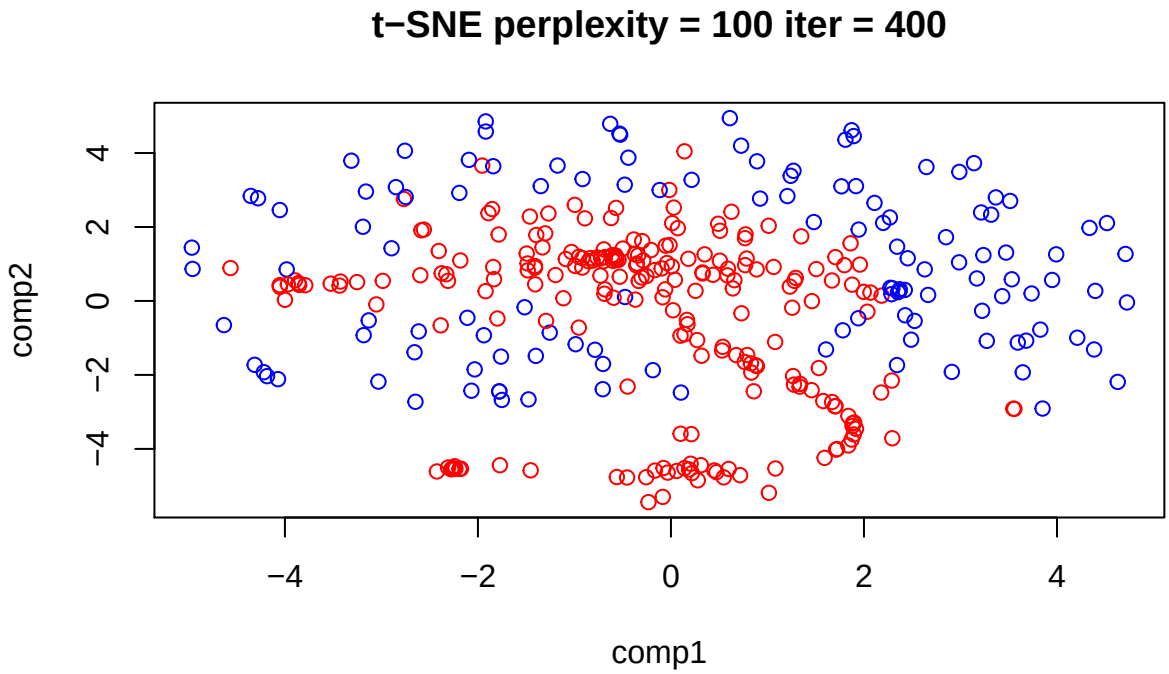

```
plot.tsne(trimmed,30,class=col_row)
```



```
plot.tsne(trimmed,50,class=col_row)
```




```
plot.tsne(trimmed,100,class=col_row)
```



Secinājumi

Ar visām *perplexity* vērtībām, izņemot 3 un 100, var novērot, ka sarkanā klase veido divus klasterus - vienu izstieptu un ārpus zilo punktu mākoņa, otru bez nekādas izteiktas formas taču ar zilās klases punktiem visapkārt. Pie vērtības 3 pēc dimensiju redukcijas iegūtais punktu mākonis veido tikai nelielus lokālus klasterus, kas škietami nejauši izkaisīti plaknē. Pie vērtības 100, abi sarkanās klases klasteri apvienojušies izliektas "T" formas struktūrā, kam apkārt ir zilās klases punkti. Zīmīgi, ka lokāli klases nav sevišķi sajauktas, taču izdarīt kaut kādu nozīmīgus spriedumus par struktūru bez zināšanām par datu kopu nav iespējams - un bez klašu anotācijas apšaubāmi, vai kāds klasterizācijas algoritms varētu konstatēt divu klašu klātbūtni.

Ar PCA iegūtais attēls ietver tikai 44% kopējās dispersijas, taču arī tajā novērojams kas līdzīgs t-SNE atrastajām struktūrām.

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 tajiem, 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 bilbiotēkām, pēc saskarnes izveidošanas pārējais kods izskatās pēc parasta R skripta - Python modulim tiek izveidots atbilstošs R objekts ar visām tā metodēm un atribūtiem. Jāseko līdzi 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:

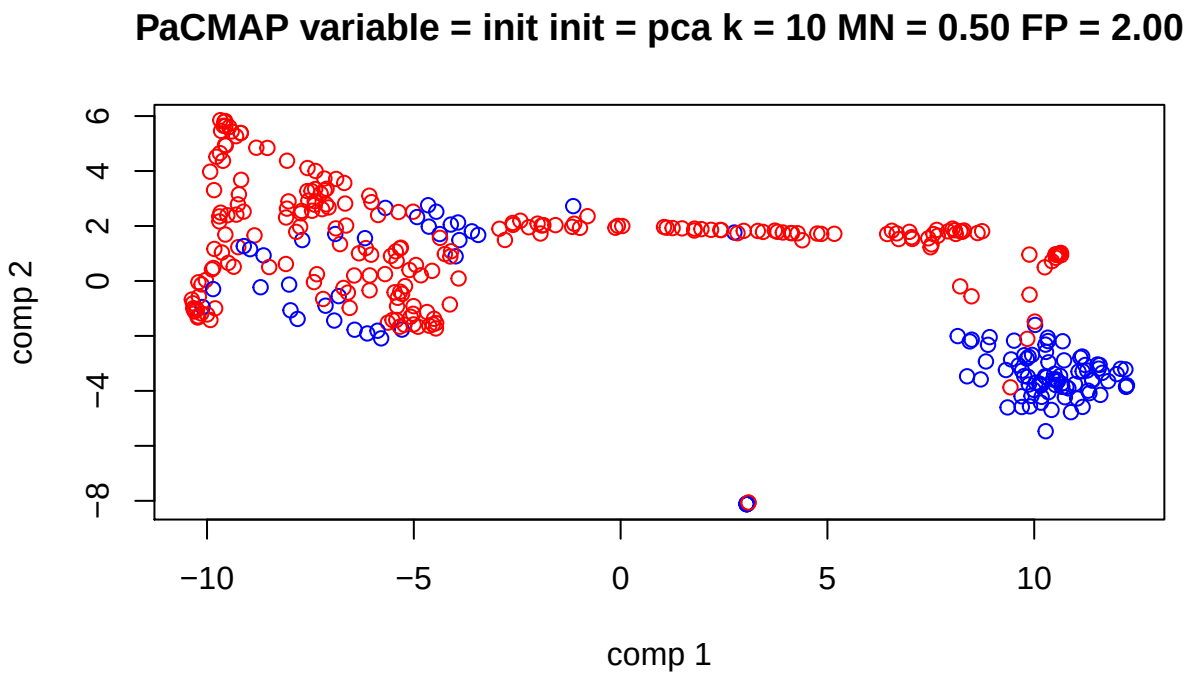
```
col_row <- colors(data[,length(data[1,])])
mat <- as.matrix(trim(data))
```

Algoritmam pieejami sekojošie hiperparametri:

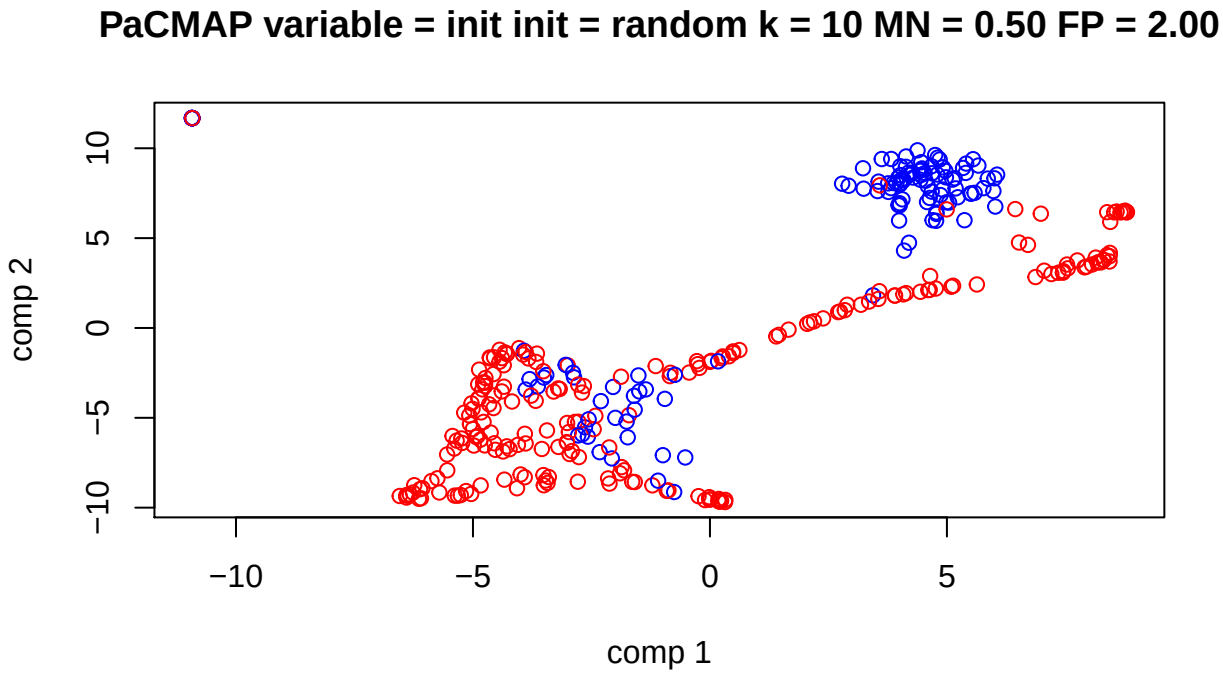
- *init* - 2-dimensionālā punktu mākoņa sākotnējo vērtību iegūšanas metode. Pēc noklusējuma PCA, iespējams iegūt arī nejauši ģenerētas vērtības. Principā algoritms ir robusts pret inicializāciju, taču PCA it kā sniedzot nedaudz labākus rezultātus un ātrāku konverģenci. Citiem algoritmiem (konkrēti, TriMap 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:

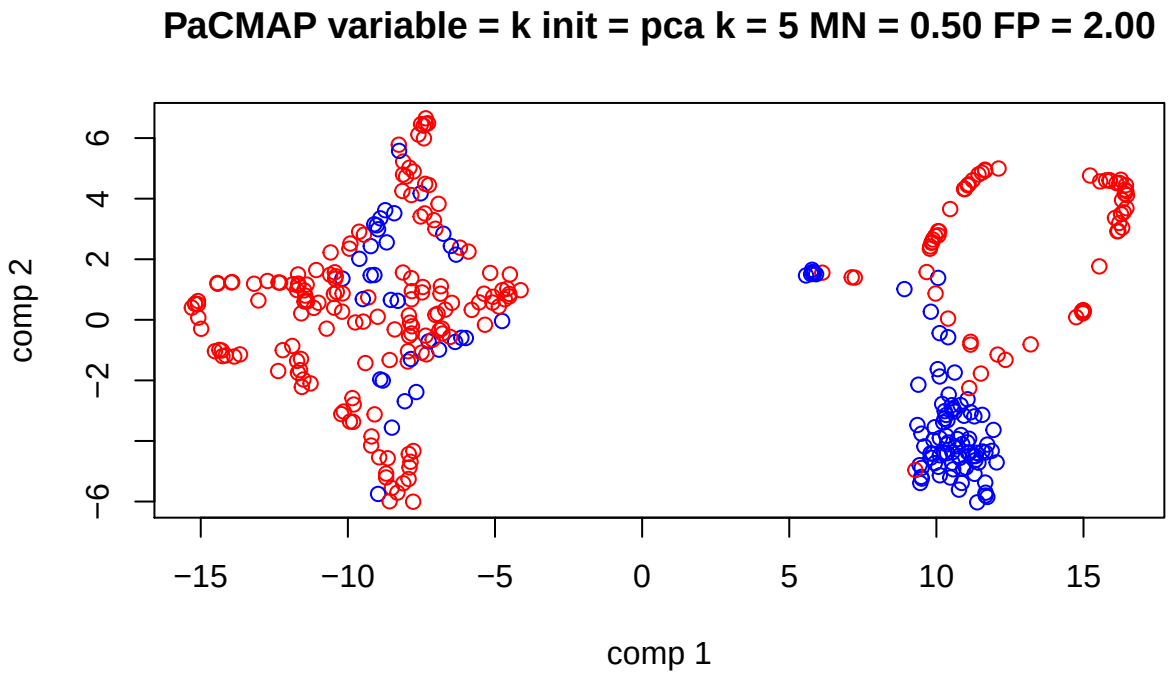
```
pac_wrapper(mat, col_row, init="pca", var="init") # pca is default
```



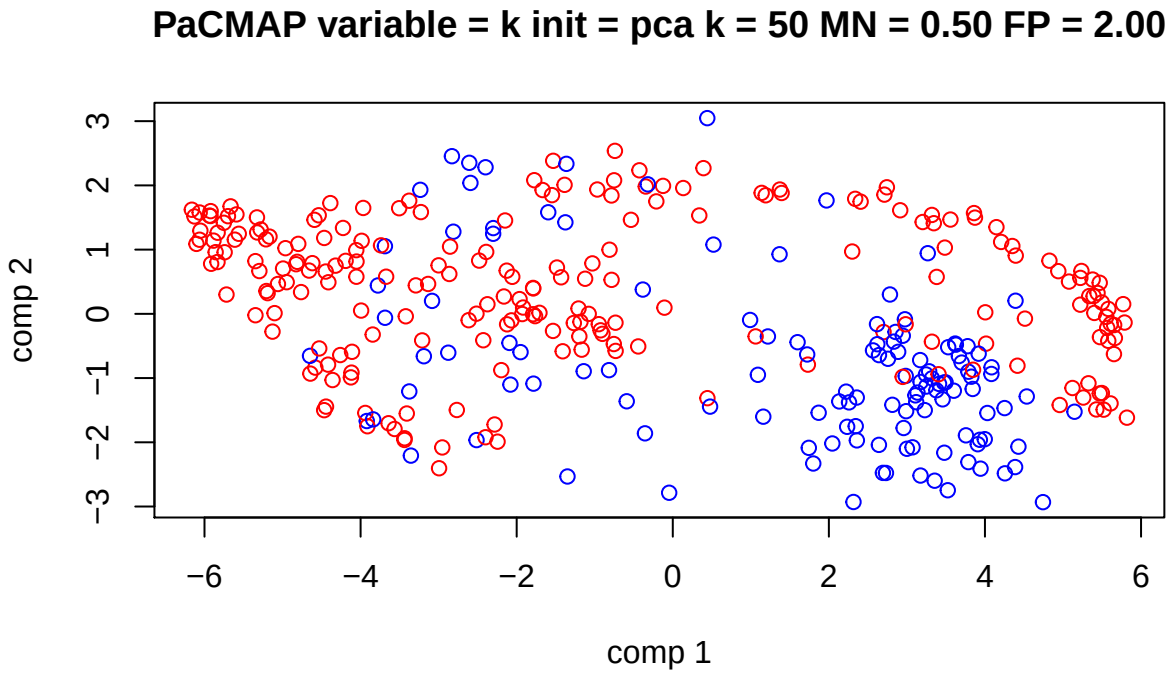

```
pac_wrapper(mat, col_row, init="random", var="init")
```



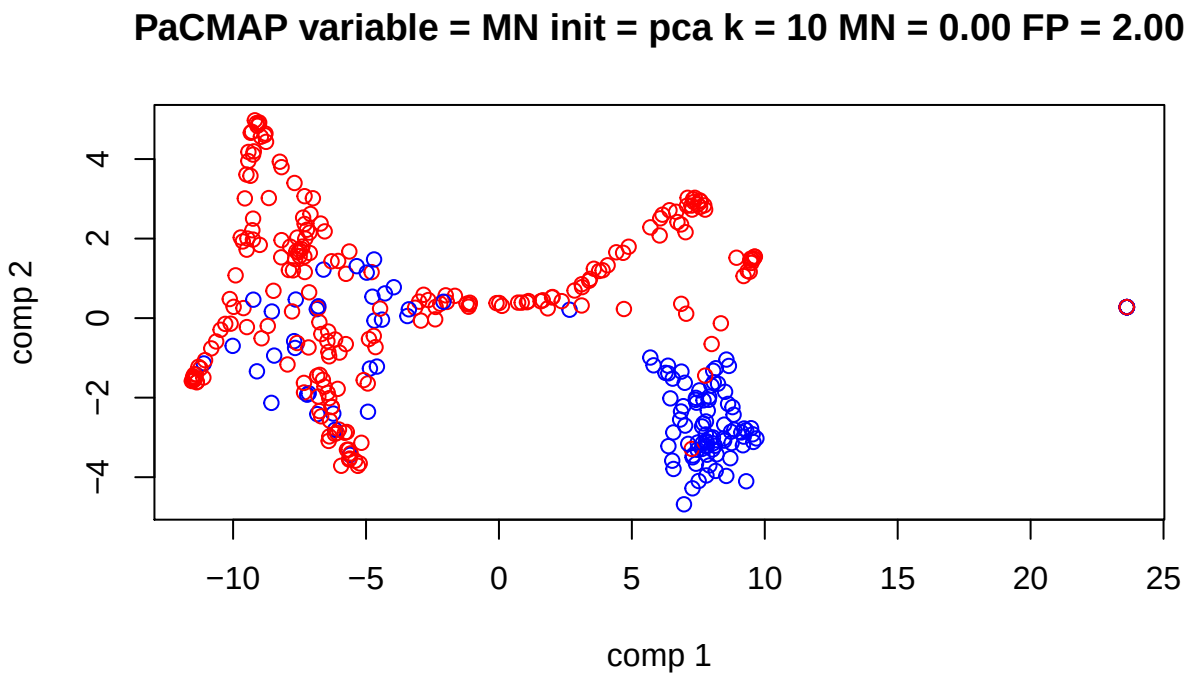
```
pac_wrapper(mat, col_row, k=5, var="k")
```



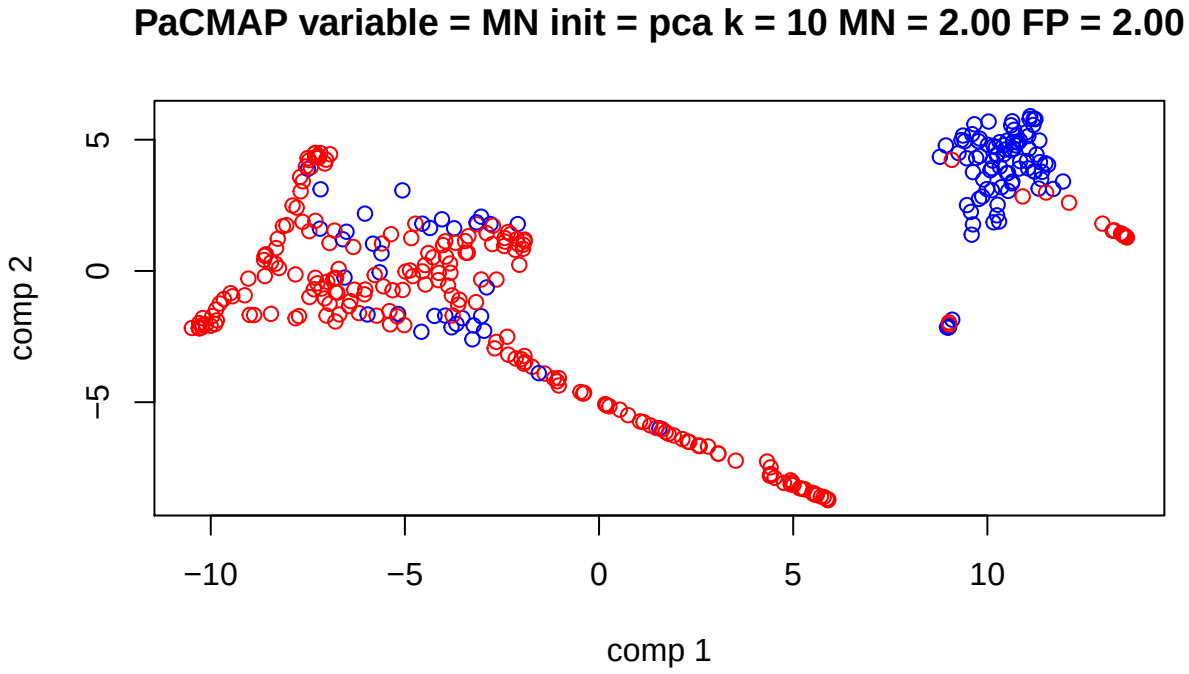

```
pac_wrapper(mat, col_row, k=50, var="k")
```



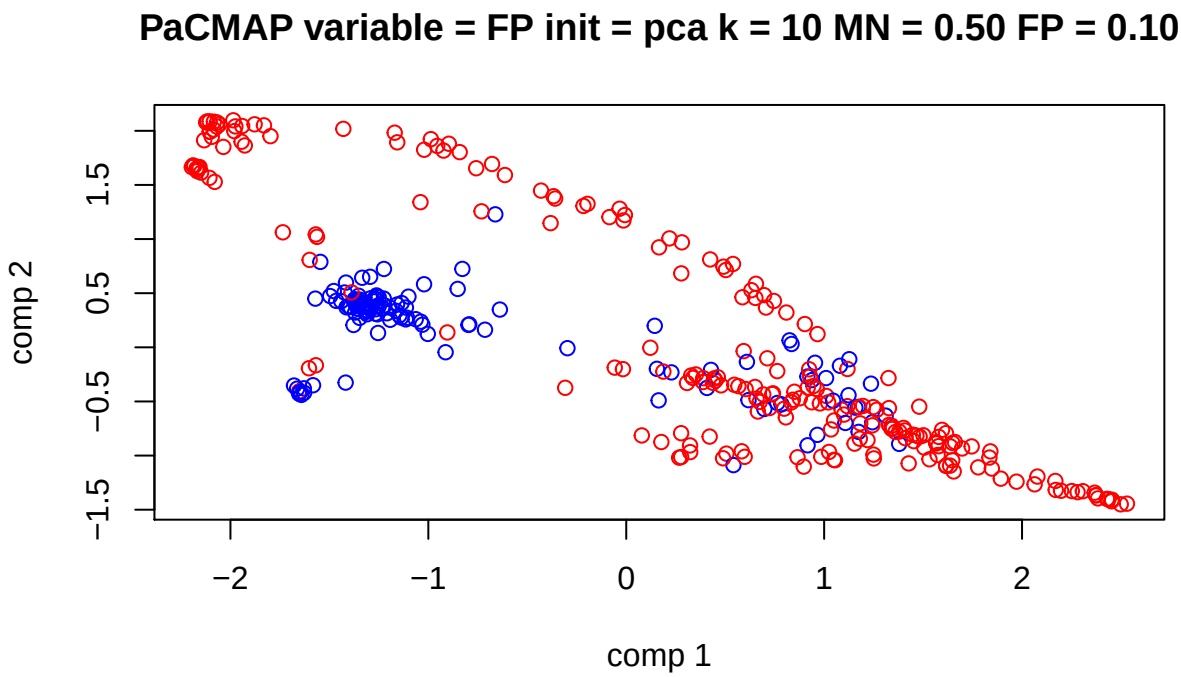
```
pac_wrapper(mat, col_row, MN_ratio=0, var="MN")
```



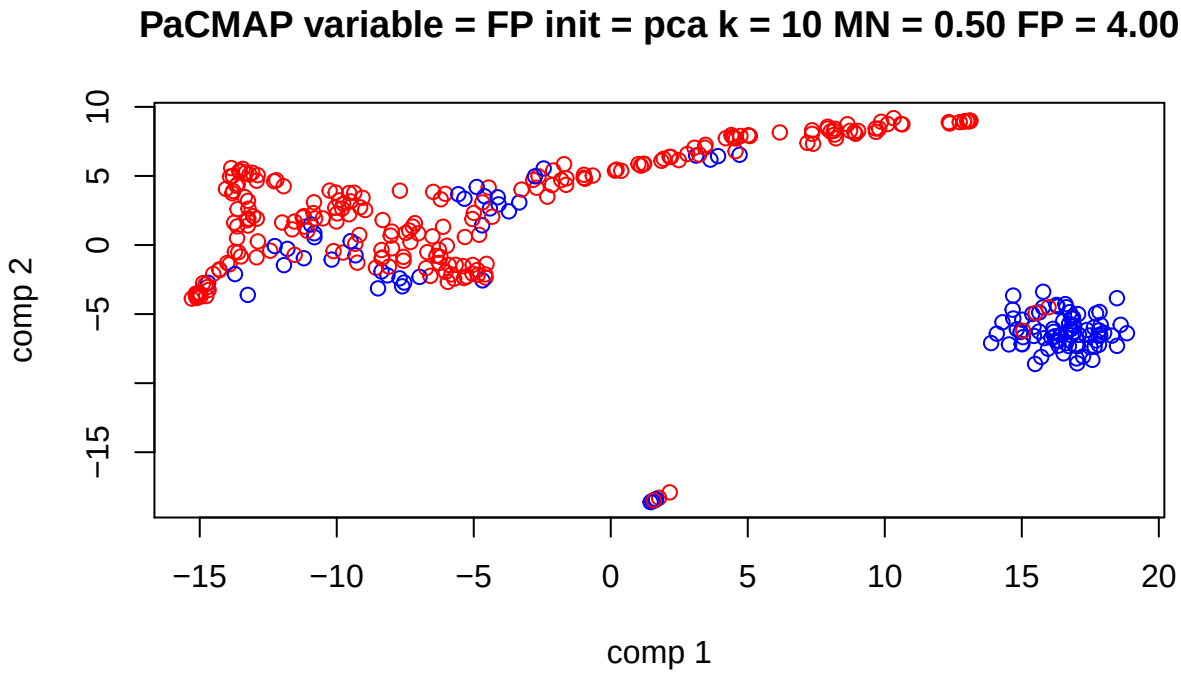

```
pac_wrapper(mat, col_row, MN_ratio=2, var="MN")
```



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




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



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. Vienīgais, ko var pateikt pilnīgi droši, ir tas, ka t-SNE ir daudz lēnāks algoritms.

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 līdzīka sakarība kā ar *n_neighbours*, tikai mazāk izteikti (pievēršot 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.