Projekt II

Filip Axelsson & Julia Holmgren

2021-02-19

Uppgift 1

a)
Skillnaden mellan övervakad och oövervakad inlärning är att vid övervakad är målet ofta att prediktera. Genom att data som består av $D = (x_i, y_i)$ används och delas upp i tränings- och testset för att utvärdera hur bra prediktionerna generaliseras på "ny data". I oövervakad inlärning så är målet att hitta okänd struktur i data och består endast av input iskilland från övervakad som består av output också.

b)
Datasetet som ska undersöka heter iris som beskriver längd och bredd på foderblad och kronblad olika arters av blommor. I datasetet består av 150 observationer. I *Table 1* visas ett utdrag av datasetet.

```
# Läs in data
df <- iris[, 1:4]


df %>%
  head() %>%
  knitr::kable(caption = "Utdrag från data")
```

Table 1: Utdrag från data

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4

Uppgift 2

a)
Målet med att K-means är att tilldela varje datapunkt till ett kluster så att det minimerar kvadratsumman av avståndet mellan datapunkterna till det klustercenter som ligger närmast. Detta görs genom att sätta använda följande struktur:
Antag att det finns N stycken observationer i ett D-dimensionellt Euklidiskt rum, där $X_1, X_2, \ldots, X_N \in R^D$. K kommer representera antalet kluster, det antas även att K är känt, μ_K är en vektor som representerar centerpunkterna för de K kluster. Genom att introducera en dummyvariabel $r_{nk} \in (0, 1)$ det vill säga

$$r_{nk} = \begin{cases} 1 & \text{Om } x_n \text{ tillhör kluster K} \\ 0 & \text{Om } x_n \text{ inte tillhör kluster K} \end{cases}$$

Detta leder till att målet kan omformuleras till att hitta $\{r_{nk}\}$ och $\{\mu_k\}$ som minimerar följade:

$$J = \sum_{n}^{N} \sum_{k}^{K} r_{nk} \cdot ||x_n - \mu_k||^2$$

Denna summa minimeras genom att iterativ process som går igenom två steg i varje iteration. Processen börjar med att gisa μ_k sedan görs följande steg:

1. minimera J med avseende på r_{nk}, där μ_k är fixt.
2. minimera J med avseende på μ_k, där r_{nk} är fixt.

Dessa två steg upprepas tills att J har konvergerat. Det är även värt att notera att dessa två steg motsvarar Expectation och Maximization stegen i EM-algoritmen. Mer detaljerad information kring stegen kan beskrivas enligt följande:
1:a steget
På grund av att J är formulerad som en linjär funktion utav r_{nk} kan $\underset{r_{nk}}{\text{argmin}}\ J$ beräknas exakt. Det vill säga för varje x_n där $n = 1, 2 \ldots, N$ väljs det klustercenter som ligger närmast

$$r_{nk} = \begin{cases} 1 & \text{Om } k = \underset{j}{\text{argmin}}\ ||x_n - \mu_j||^2 \\ 0 & \text{Annars} \end{cases}$$

2:a steget
I detta steg kommer μ_k att optimeras medan r_{nk} är fixt. Eftersom J är en kvadratisk funktion utav μ_k kan den minimeras genom att derivera med avseende på μ_k och sätta uttrycket till lika med 0. Detta ger följande

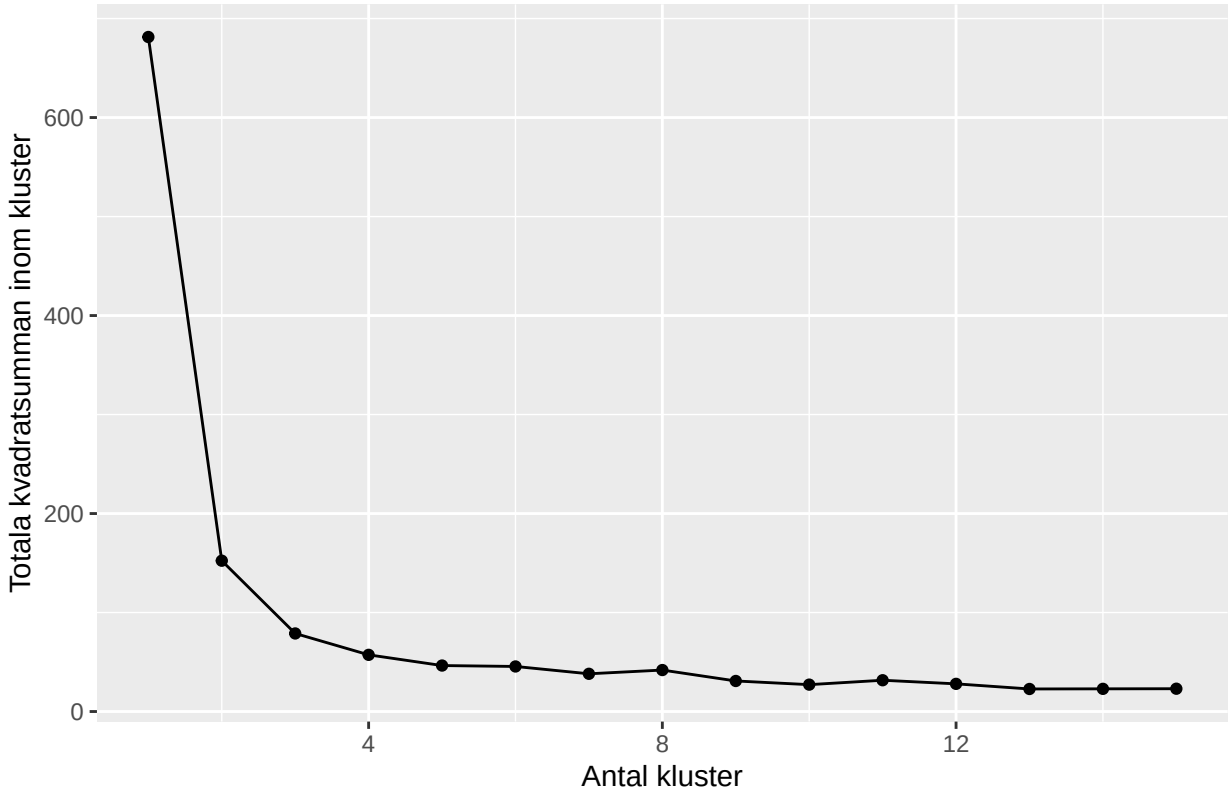
$$2\sum_{n}^{N} r_{nk}(x_n - \mu_k) = 0 \Leftrightarrow \mu_k = \frac{\sum_n^N r_{nk} \cdot x_n}{\sum_n^N r_{nk}}$$

b)
Härnäst så bestäms K genom att iterativt utförs kmeans och plotar en elbowplot där k plotas emot totala kvadratsumman inom kluster.

```
elbowplot <- function(data, k, title){
  withinss <- c()
  for(i in 1:k){
    withinss <- append(withinss,kmeans(data,i)[["tot.withinss"]])
  }
  as_tibble(cbind(c(1:k),withinss)) %>%
    ggplot(aes(x=V1,y=withinss)) +
      geom_point() +
      geom_line() +
      labs(x = "Antal kluster", y = "Totala kvadratsumman inom kluster") +
      ggtitle(title)
}
elbowplot(df,15,"Figur 1: Elbowplot")
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

Figur 1: Elbowplot

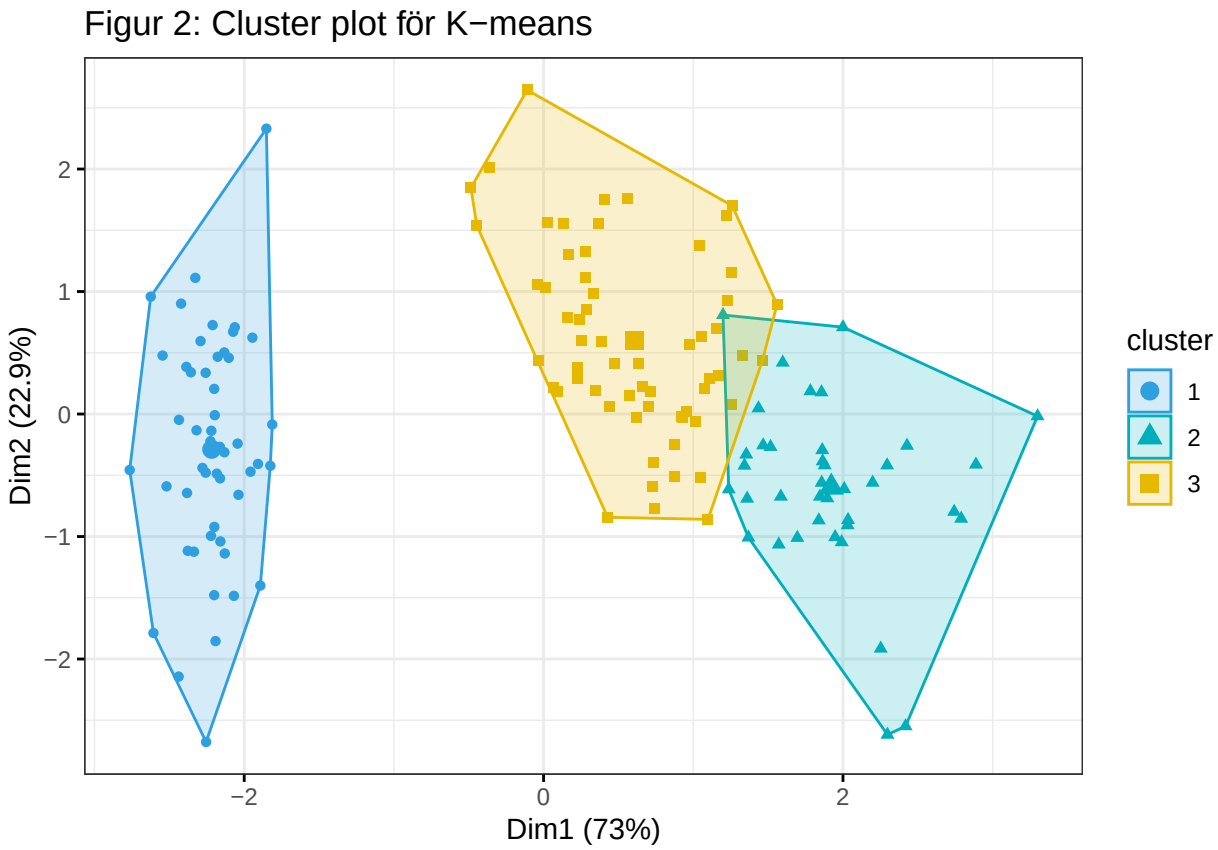


I figur 1 kan det observeras att k = 3 verkar vara det mest lämpade k:et. Detta bestäms genom att den totala kvadratsumman inom kluster inte förändras mycket genom att öka k större än tre.

c) Enligt plotten nedaför *Figur 2* visar det sig att utan färgläggning och områdesbeskrivning som har kunnats läggas till med hjälp utav PCA, så är det svårt att urskilja att det skulle finnas 3 kluster.

```
# k-means
kmeans_iris <- df %>%
  kmeans(3)
```

```
fviz_cluster(kmeans_iris, data = df[, -5],
             palette = c("#2E9FDF", "#00AFBB", "#E7B800"),
             geom = "point",
             ellipse.type = "convex",
             ggtheme = theme_bw()
             ) +
  ggtitle("Figur 2: Cluster plot för K-means")
```



Uppgift 3

a)
Guassian Mixtures (GMM) antas data att komma ifrån normalfördelingar. Detta kan förklaras genom att se X_k som en slumpvariabel i kluster k vilket ger att $X_k \sim N(\mu_k, \Sigma_k)$, där μ_k representerar medelvärdevektorn och Σ_k är kovariansmatrisen. Deb blandade normalfördelningen skrivs enligt följande:

$$f(x) = \sum_{k}^{K} \pi_k N(x|\mu_k, \Sigma_k)$$

Där π_k är "sannolikheten" att x tillhör kluster k.

I EM-algoritmen för GMM antas det att data kommer ifrån k, multivariata normalförelningar där målet kommer vara att anpassa data till modellen, dock är ett problem att modellen är okänd. Detta försöker dock EM-algortimen att lösa genom att iterativt gissa föredelning och sedan uppdatera parameterna. Detta görs genom att maximera likelihood-funktionen och processen utförs tills konvergens har uppnåtts. Mer djupgående på hur EM-algoritmen fungerar kan delas upp i tre steg.

1. Först måste en gissning på modelparametrarna μ_k, Σ_k och π_k utföras
2. E-steget: Beräkna väntevärdet av likihooden för data givet de nuvarande parametrar.
3. M-steget: Uppdatera paramerarna så att de maximerar likelihooden.

Detta upprepas tills att konvergens har uppnåtts.

b)
Koden denna utför GMM för k=3.

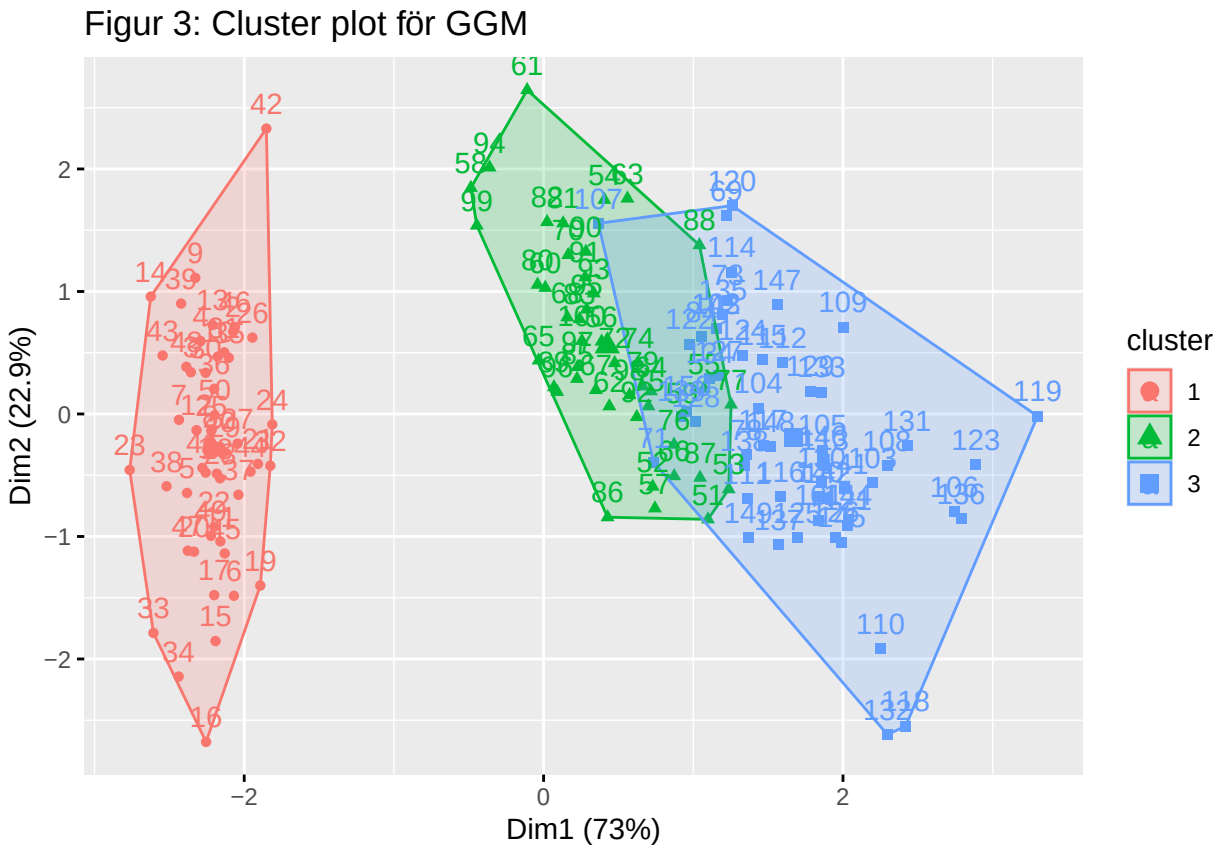
```
# Gaussian Mixture Models
ggm_iris <- df %>% Mclust(G=3)

summary(ggm_iris)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VEV (ellipsoidal, equal shape) model with 3 components:
##
##  log-likelihood   n df       BIC       ICL
##        -186.074 150 38 -562.5522 -566.4673
##
## Clustering table:
##  1  2  3
## 50 45 55
```

c) Enligt *Figur 3* visar det sig att två kluster överlappar varandra ganska mycket, sanna plott visar även klusterformerna verkar vara spfärsika vilket GMM ofta tenderar att genrera.

```
fviz_cluster(ggm_iris, data= df) +
  ggtitle("Figur 3: Cluster plot för GGM")
```



d)
I denna uppgift ska vi utvärdera hur bara klustringen fungerar. Genom att analysera *Table 2* visar det sig att endast fem observationer placerades fel, vilket kan anses som ett bra resultat.

```r
# d)
#Procentuellt antal korrekt placerade observationer

r<- cbind(sum(ggm_iris[["classification"]] == as.numeric(iris[,5]))/length(as.numeric(iris[,5])),sum(ggm
# Antal fel
f <- cbind((length(as.numeric(iris[,5]))-(sum(ggm_iris[["classification"]] == as.numeric(iris[,5]))))/le

d <- as_tibble(rbind(r,f))
rownames(d) <- c("Rätt placerade","Fel placerade")
```

```
## Warning: Setting row names on a tibble is deprecated.
```

```r
d %>%
  knitr::kable(caption = "Kontroll utav vilka observationer som är placerad i korrekt kluster",
               col.names = c("Procent", "Antal"),
               row.names = T)
```

Table 2: Kontroll utav vilka observationer som är placerad i korrekt kluster

	Procent	Antal
Rätt placerade	0.9666667	145
Fel placerade	0.0333333	5

e)
Härnäst bortser vi från de punkter som har mindre än 95% chans att tillhöra ett kluster och utvärderar hur korrekt gmm klassifierade datasetet. Resultetat kan visualiseras i *Table 3*, vilket ger att endast en observation placerades fel. Detta innebär att det finns endast en obseravton som hade en sannolikhet över 95% som placerades fel.

```r
# e)
#Procentuellt antal korrekt placerade observationer
filter_gmm <- as_tibble(cbind(ggm_iris[["classification"]],ggm_iris[["uncertainty"]],as.numeric(iris[,5]
  filter(V2 <= 0.05)

r2<-cbind(sum(filter_gmm$V1==filter_gmm$V3)/length(filter_gmm$V1),sum(filter_gmm$V1==filter_gmm$V3))
#Antal fel
f2<-cbind((length(filter_gmm$V1)-sum(filter_gmm$V1==filter_gmm$V3))/length(filter_gmm$V1),(length(filte

d2 <- as_tibble(rbind(r2,f2))
rownames(d2) <- c("Rätt placerade","Fel placerade")
```

```
## Warning: Setting row names on a tibble is deprecated.
```

```r
d2 %>%
  knitr::kable(caption = "Kontroll utav vilka observationer som är placerad korrekt och har <95 procent
               col.names = c("Procent", "Antal"),
               row.names = T)
```

Table 3: Kontroll utav vilka observationer som är placerad korrekt och har <95 procent slh att tillhöra rätt kluster.

	Procent	Antal
Rätt placerade	0.9929078	140

	Procent	Antal
Fel placerade	0.0070922	1

f)
K-means algoritmen och EM algoritmen för Guasian mixtures visar sig vara likartade. I K-means algoritmen utförs en så kallad "hard" klustring där varje observation är associerad till ett unikt kluster. I EM algrotimen utförs istället en "soft" klustring som är baserad på en posteriori sannolikhet. Första steget och andra steget i K-means representerar E-steget och M-steget i EM algoritmen. Ett speicalltillfälle då GMM och K-means är "samma", är då en "hard" klustring version utav GMM med en generell kovariansmatris används, detta är även känd som elliptical K-means. Ytterliggare skillnader mellan dessa två metoder är att i K-means generas ingen sannolikhet att en observation tillhör ett kluster, samt att datamängdens fördelning är okänd. Om man vill veta vad sannolikheten att en obseravtion tillhör ett specifikt kluster, och formen på varje kluster så är GMM en lämplig metod att använda.

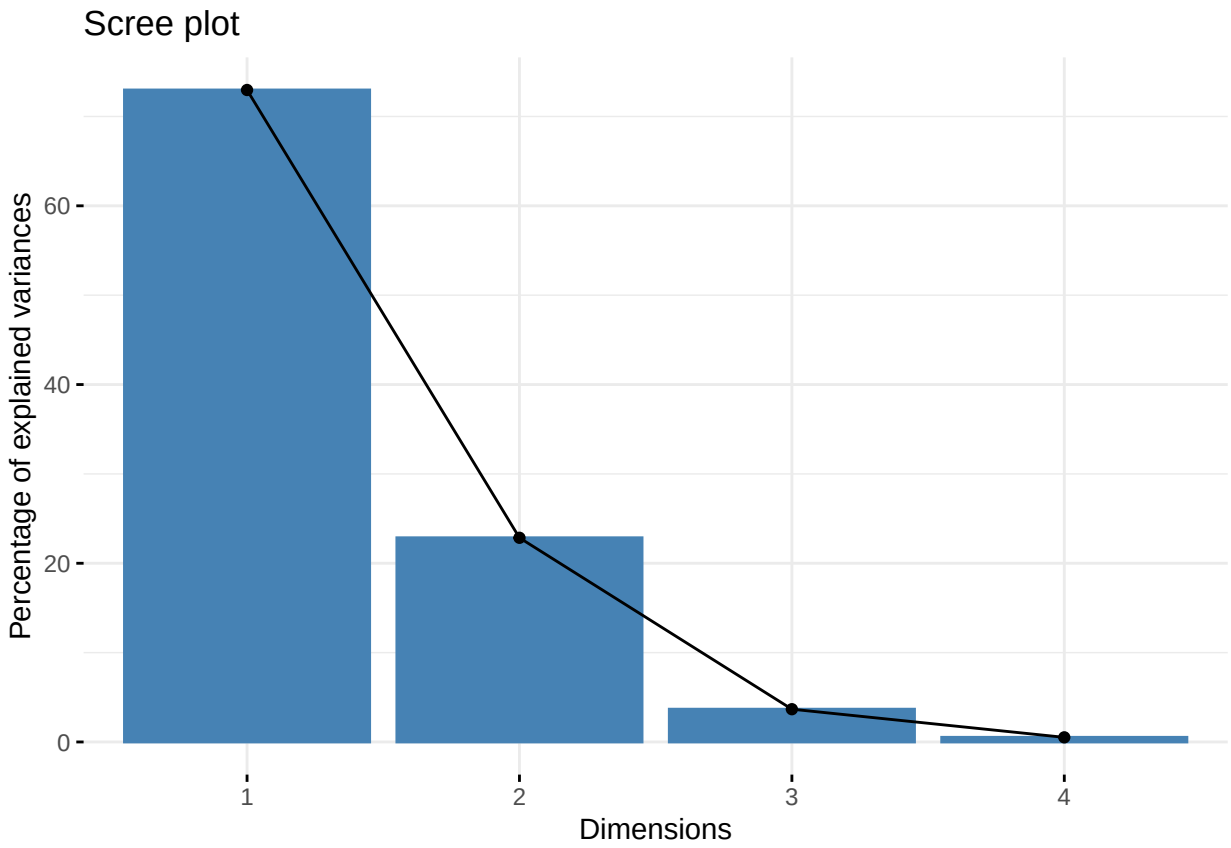
Uppgift 4

a)
PCA eller Principalkomponentanalys tranformerar data till en lägre dimension så en ortogonal projektion av datapunkterna till ett delrum, vilket ska uppfylla vilkoret att variansen av de projicerade datapunkterna maximeras. Ett alternativt synsätt är att minimera summan av avståndet projektionen. Kan även vara värt att notera att skillnaden mellan denna metod och regression är vinkeln som används för mäta avståndet till projektionen. Normalisering är viktigt när PCA ska användas detta på grund av det är en maximering utav varaiansen som ska utföras. b)
Vid utförande av PCA på de fyra första kolummnerna i datasetet Iris, där data är skalad och centriserad innan analys ger att första kompnentens förklar 73% av variansen i data, detta beräknas i koden nedaför men kan även visualiseras i plotten nedanför *Figur 4*.

```
# # PCA
 irisPCA <- princomp(df, cor = TRUE) # cor = TRUE, the data will be centered and scaled before the anal

var_first_comp <- 1.7083611^2/(0.9560494^2+ 0.3830886^2+ 0.1439265^2+1.7083611^2)

fviz_eig(irisPCA)
```



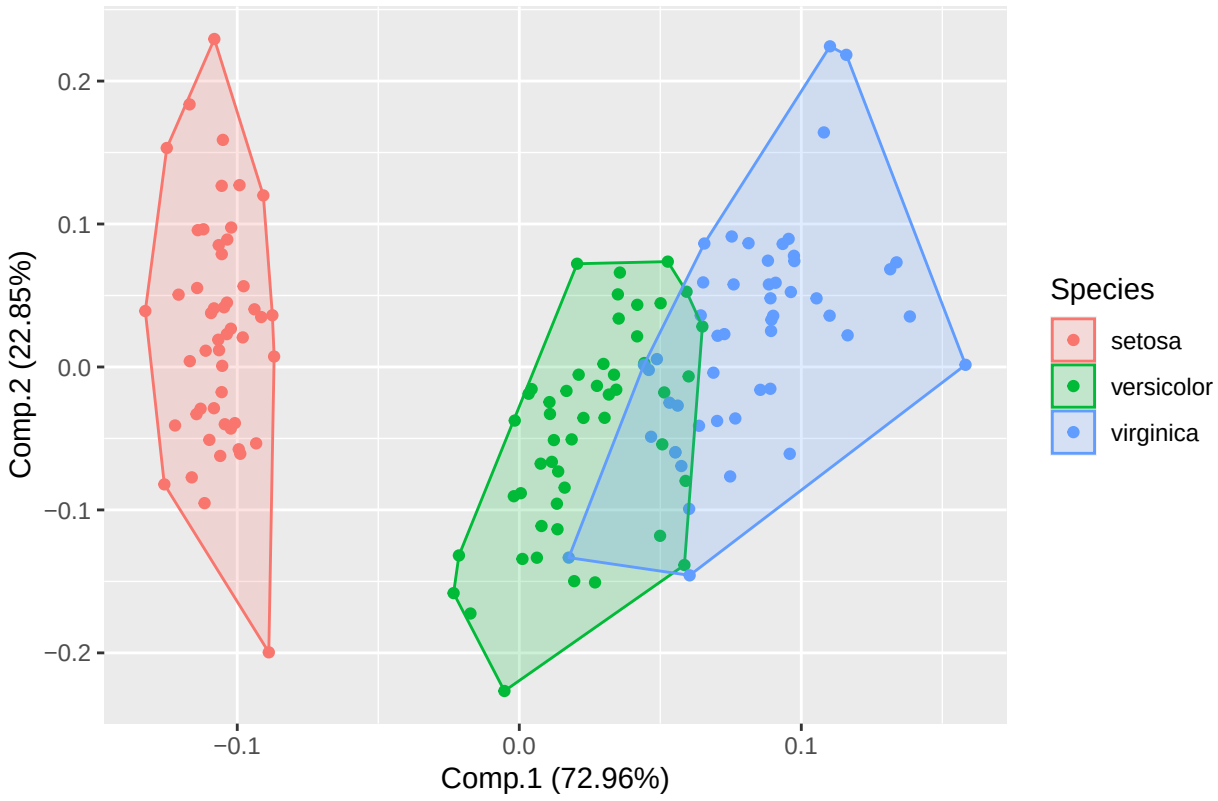
c) Enligt plotten nedanför *Figur 5* kan det analyseras att klusterna även här överlappar varandra.

```
library(ggpubr)
library(ggfortify)
autoplot(irisPCA, data = iris, colour = 'Species', frame = TRUE) +
  ggtitle("Figur 5: Cluster plot för PCA")
```

```
## Warning: `select_()` is deprecated as of dplyr 0.7.0.
## Please use `select()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## Warning: `group_by_()` is deprecated as of dplyr 0.7.0.
## Please use `group_by()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

Figur 5: Cluster plot för PCA



Uppgift 5

a)
Tidigare har vi använt neurala nätverk för övervakad inlärning där syftet med nätverket har varit att prediktera en output y givet variabler X. Målet nu är att använda neurala nätverk för dimensionreduktion. Genom att föreställa sig ett nätverk med D inputs, D outputs samt M gömda noder och där $M < D$. Utifrån detta tränas nätverket mot inputvariablerna, vilket i sig betyder att nätverker försöker mappa varje inputvariabel mot sig själv, vilket även kallas för "autoassociative mapping". Då antelet gömda noder är färre än antalet inputsvariabler kan det inte återskapas exakt. Detta leder till att det neurala nätverkets paramterarar, vikterna W bestäms genom att minimera en så kallad "error function" som ska uppfatta vilken grad av inputen och rekonstruktionen som inte matchas. Det som ska minimeras kan skrivas enligt följande:

$$E(W) = \frac{1}{2}\sum_{n=1}^{N} ||y(x_n, W) - x_n||^2$$

Vid tillfället då de gömda noderna har linjära aktiveringsfunktioner kan det visas att ovstående ekvation har ett unikt minium vilket är den porjektion utav data till ett M-dimensionellt rum vilket består utav de M första principalkomponenterna, nätverket är alltså ekvivalent med PCA. Om det läggs till lager av gämda noder kan det utföras icke-linjära dimensionreduktion vilket även kallas autoencoder.

b) Genom att utföra en autoencoder-modell, där målet är att hitta en 2-dim utav data där de tre arterna är separerade, vi använder oss av koden nedanför.

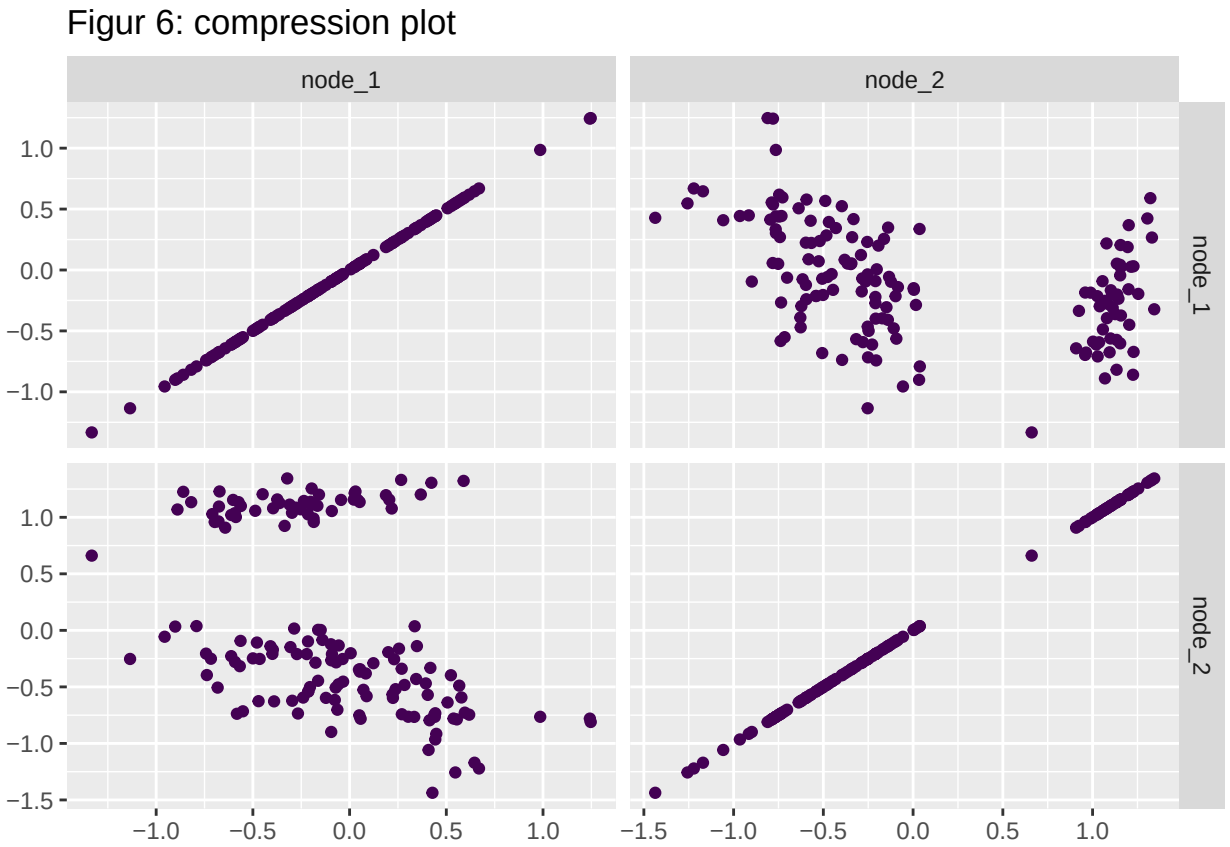
```
set.seed(2021)
# Autoencoder
 X <- as.matrix(df)
 irisAE <- autoencoder(X, c(10,2,10), loss.type = 'squared',
```

```
                  activ.functions = c('tanh','linear','tanh'),
                  batch.size = 8, optim.type = 'sgd',
                  n.epochs = 1000, val.prop = 0)
```

```
## Artificial Neural Network:
##   Layer - 4 nodes - input
##   Layer - 10 nodes - tanh
##   Layer - 2 nodes - linear
##   Layer - 10 nodes - tanh
##   Layer - 4 nodes - linear
## With squared loss and SGD optimizer
## Training progress:
## [|--------------------------------------------------] 0% - Training loss: 4.85755[|--------------------
```

c) I *Figur 6* visas en 2-dim bild utav data

```
set.seed(2021)
 compression_plot(irisAE, X)+ ggtitle("Figur 6: compression plot")
```



d) Vid jämföresle med *Figur 5* som visar en plott för PCA och *Figur 6* fås ett liknade svar, det vill säga ett kluster är seprarat från de två andra. Detta leder till att PCA och autoencoder ger ungefär samma svar i för detta datamaterial.

e)
PCA är begränsad till linjär mappning medan autoencoder kan ha icke-linjära komponenter (enoder/decoders). Vid tillfället då endast ett lager i autoencoder med linjär transfomeringsfunktion är i princip identisk till PCA. Det finns inga direkta riktlinjer storleken ("bottleneck"/"flaskhals") för laggerna i autoencoder, PCA används ofta som en riktlinje för antal kluster, k. Autoenconder tenderar även att fungera bättre än PCA vid små k, detta innebär att samma precession kan uppnås vid färra komponenter och ett litet dataset.

Uppgift 6

I detta projekt så har 4 styckna kluster algoritmer undersöks på Iris-datasetet, K-means, Gaussian Mixtures, PCA och Autoencoders. I uppgift 1 så undersöks k-means och med hjälp av en elbowplot bestäms antal kluster. Därefter plotas klusterna. Nästa kluster algoritm som undersöks är Gaussian Mixture som klustrar "soft" så den tilldelar med hjälp av normalfördeningar sannolikheter att varje punkt tillhör ett kluster. Därefter plotades klustringen och resultatet utvärderades av klustringen och de som är korrekt med 95% sannoliket. Resultat var många korrekt klassificerade. Den tredje algoritmen som undersöktes var PCA som transformeras data till en ortogonal projektion av datapunkterna till ett delrum där variansen maximeras. Där resultatet blev att första kompnentens förklar 73% av variansen. PCA lyckades inte separera klusterna från varandra. Den sista kluster algoritmen som testades var autoencoder som använder neuralnetworks för att klustra fördelen med denna algoritmen är att den kan mapa icke-linjärt. I projektet lyckades inte autoencoder seperera klusterna. Sammanfattningsvis så lyckades inte någon av de olika metoderna att dela upp data i tre tydliga kluster, utan främst till två.