SMM636 Machine Learning (PRD2 A 2019/20)

R exercises 10: Unsupervised learning (Part II)

DR. Rui Zhu, DR. Feng Zhou

2020-03-27

In R exercise 10, you will know

- How to use ICA, kernel PCA and ISOMAP

Don't forget to change your working directory!

1 Independent component analysis (ICA)

We take the cocktail party data as an example.

We first generate the source signals and the mixing matrix to get the observed signals.

```
##### get source signals
S <- cbind(sin((1:1000)/20), rep((((1:200)-100)/100), 5))
##### get mixing matrix
M <- matrix(c(0.3, 0.7, -0.5, 0.6), 2, 2)
##### plot source signals
par(mfrow = c(1, 2))
plot(1:1000, S[,1], type = "l",xlab = "source 1",ylab="",cex.lab=1.5,
       cex.axis=1.5,lwd=2)
plot(1:1000, S[,2], type = "l", xlab = "source 2",ylab="",cex.lab=1.5,
       cex.axis=1.5,lwd=2)
```

Then we generate the observed signals as the mixture of the two source signals with the weights specified in the weight matrix M.

```
X = S %*% M
par(mfrow = c(1, 2))
plot(1:1000, X[,1], type = "l",xlab = "observation 1",ylab="",
       cex.lab=1.5,cex.axis=1.5,lwd=2)
```

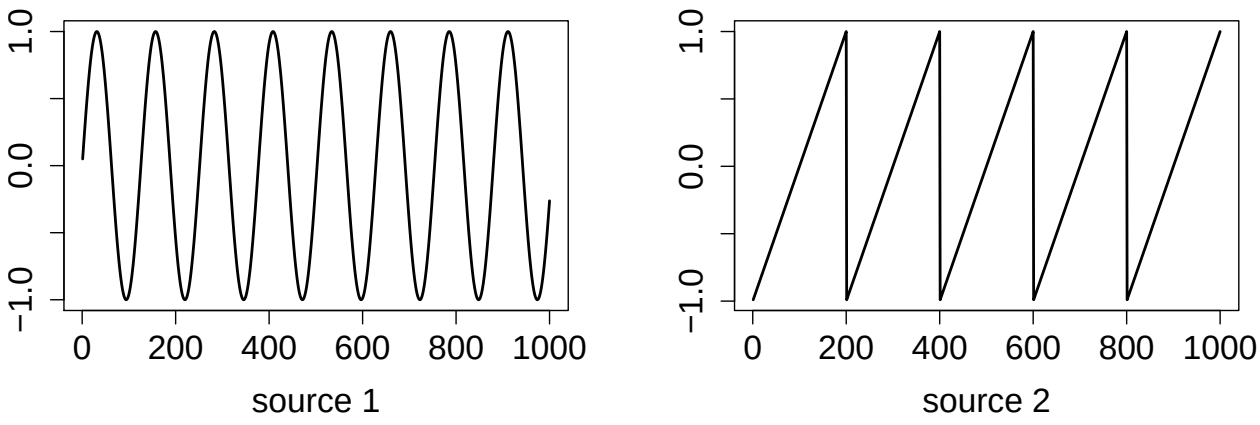

Figure 1: Source signals.

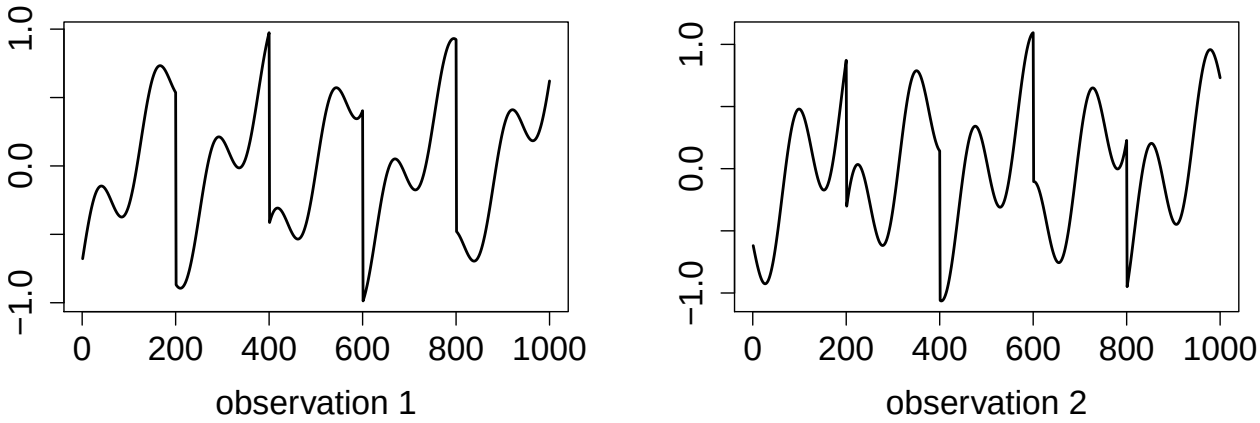

Figure 2: Observed signals.

```
plot(1:1000, X[,2], type = "l", xlab = "observation 2",ylab="",
       cex.lab=1.5,cex.axis=1.5,lwd=2)
```

Now we can use ICA to find the source signals based on the observations. We can use the `fastICA` function in the `fastICA` package provided by the authors of ICA.

```
#install.packages("fastICA")
library(fastICA)
set.seed(20)
latent = fastICA(X, 2, fun = "logcosh", alpha = 1,
            row.norm = FALSE, maxit = 200,
            tol = 0.0001, verbose = TRUE)
```

```
## Centering
## Whitening
## Symmetric FastICA using logcosh approx. to neg-entropy function
## Iteration 1 tol = 0.009539087
## Iteration 2 tol = 8.952314e-06
```

The output `latent` is a list and S in the list is the estimated source signals.

```
par(mfrow = c(1, 2))
plot(1:1000, latent$S[,1], type = "l", xlab = "Esource 1",ylab="",
       cex.lab=1.5,cex.axis=1.5,lwd=2)
plot(1:1000, latent$S[,2], type = "l", xlab = "Esource 1",ylab="",
       cex.lab=1.5,cex.axis=1.5,lwd=2)
```

2 Kernel principal component analysis

We use the simulated `flame.txt` data.

```
flame=read.table("flame.txt")
plot(flame[flame$V3==1,1],flame[flame$V3==1,2],cex=1.5,
     col="red",pch=16,xlim=c(0.5,14.2),ylim=c(14.45,27.80),
     xlab="X1",ylab="X2")
points(flame[flame$V3==2,1],flame[flame$V3==2,2],cex=1.5,col="blue",pch=17)
```

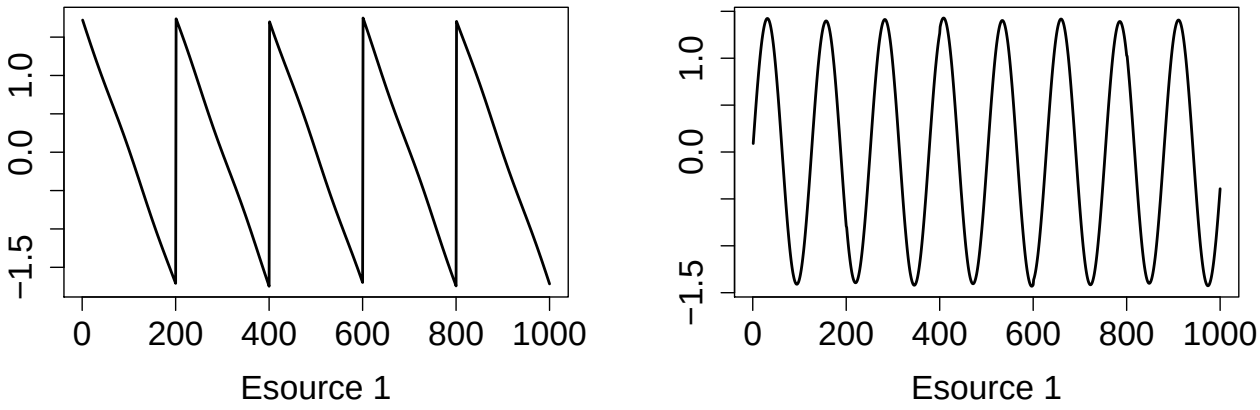

Figure 3: Estimated source signals.

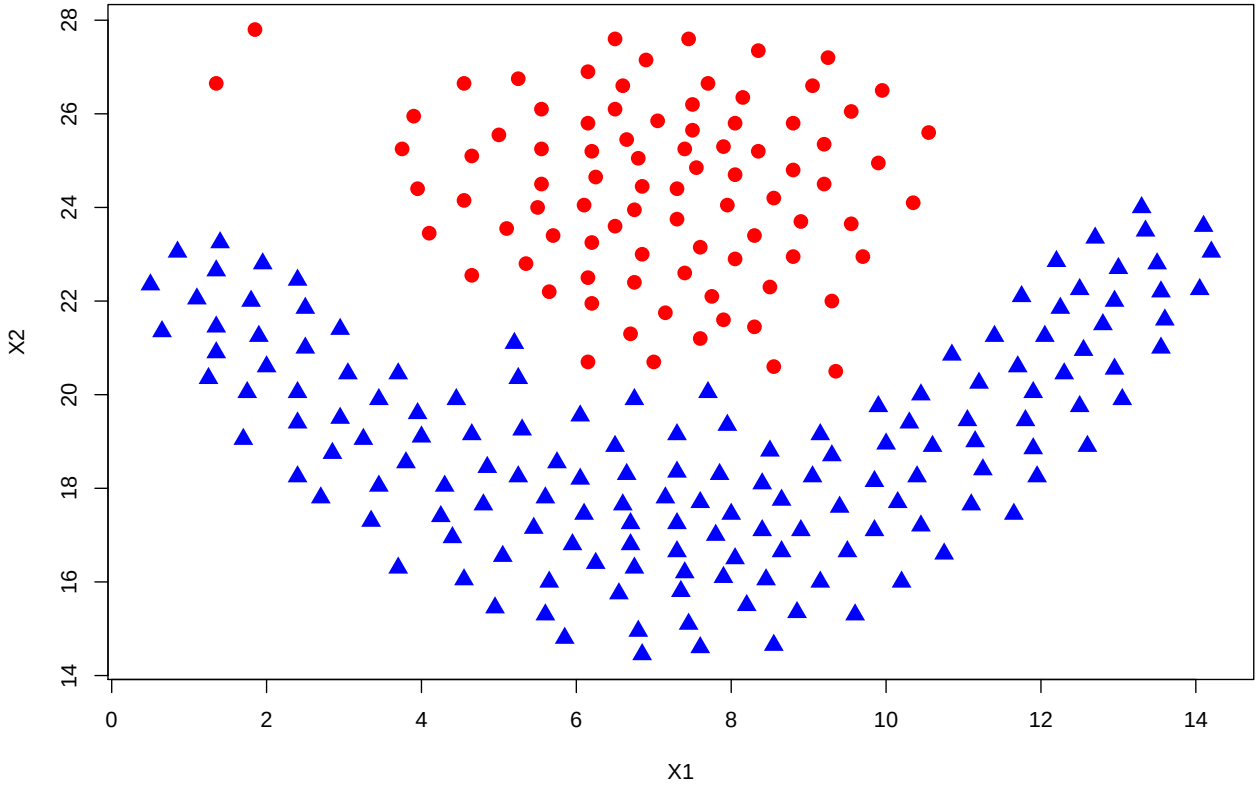

Figure 4: The flame data.

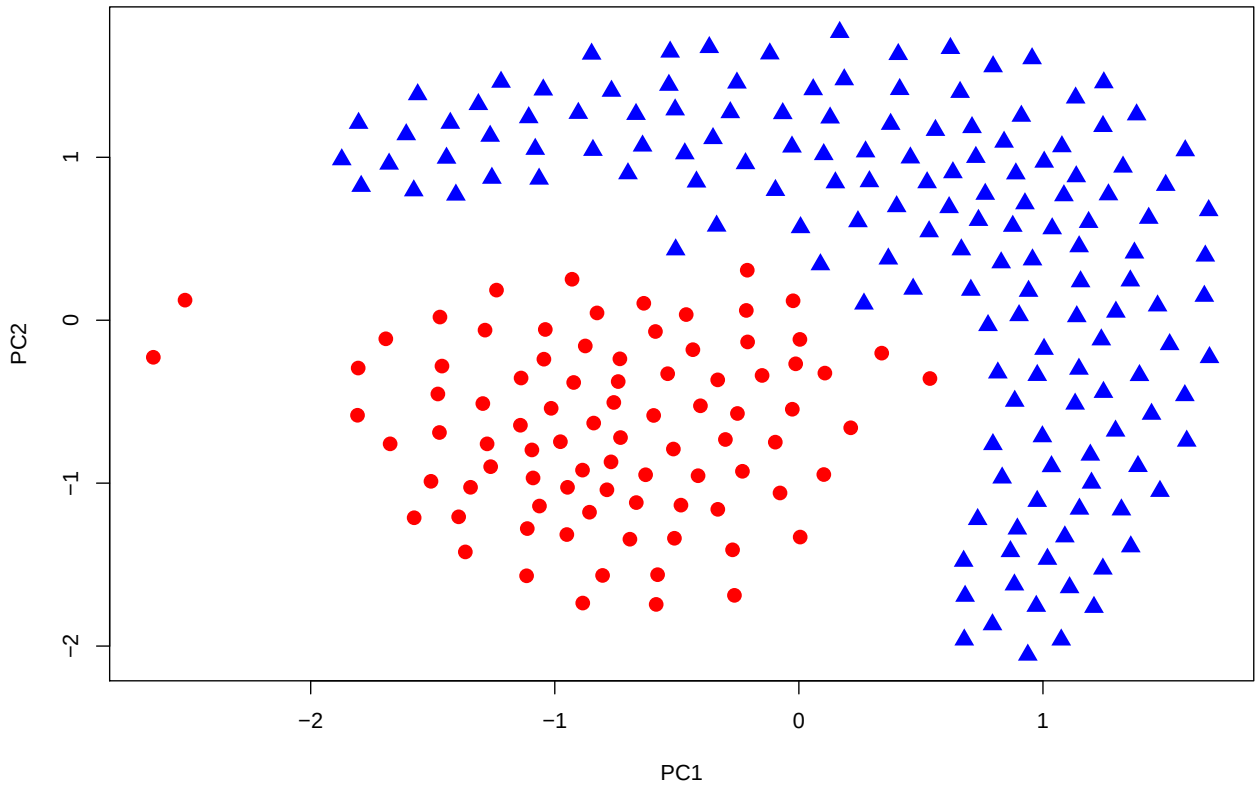

Figure 5: The PC plot of the flame data.

We can imagine that if we use PCA, the two classes cannot be linearly separated.

```
pr.out=prcomp(flame[,1:2], scale=TRUE)
plot(pr.out$x[flame$V3==1,1:2],col="red",pch=16,cex=1.5,
     xlab="PC1",ylab="PC2",xlim=c(-2.65,1.69),ylim=c(-2.06,1.77))
points(pr.out$x[flame$V3==2,1:2],cex=1.5,col="blue",pch=17)
```

Now apply kernel PCA on the data to see how it can capture the nonlinear structure. We can use the `kpca` function in the `kernlab` package. The RBF kernel is specified by `rbfdot` and the associated parameter `sigma` is the inverse kernel width, i.e. the inverse of c in the slides.

```
##### kernel PCA
#install.packages("kernlab")
library(kernlab)
par(mfrow = c(1, 4))
##### sigma=0.01
kpc=kpca(~.,data=flame[,-3],kernel="rbfdot",
         kpar=list(sigma=0.01),features=2)
plot(rotated(kpc)[flame$V3==1,1:2],col="red",pch=16,cex=1.5,
     xlab="KPC1",ylab="KPC2",xlim=c(-8.80,10.06),ylim=c(-7.0,10.70),
     main="sigma=0.01")
points(rotated(kpc)[flame$V3==2,1:2],cex=1.5,col="blue",pch=17)
##### sigma=0.1
kpc=kpca(~.,data=flame[,-3],kernel="rbfdot",
         kpar=list(sigma=0.1),features=2)
plot(rotated(kpc)[flame$V3==1,1:2],col="red",pch=16,cex=1.5,
     xlab="KPC1",ylab="KPC2",xlim=c(-8.80,10.06),ylim=c(-7.0,10.70),
```

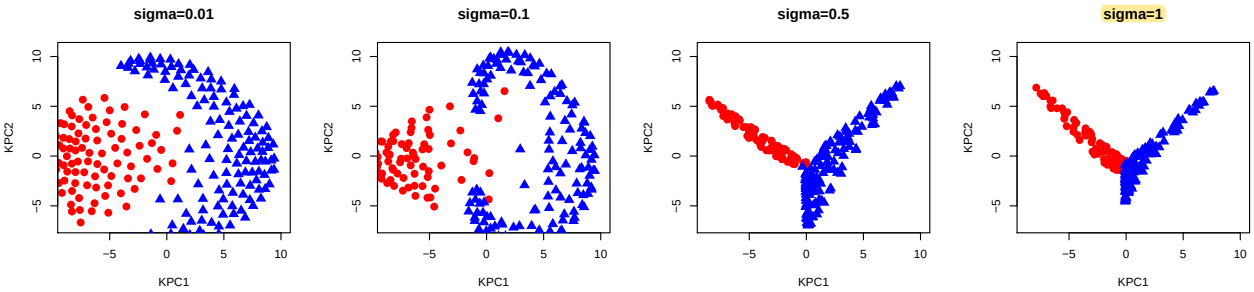

Figure 6: The kernel PC plot of the flame data.

```
     main="sigma=0.1")
points(rotated(kpc)[flame$V3==2,1:2],cex=1.5,col="blue",pch=17)
##### sigma=0.5
kpc=kpca(~.,data=flame[,-3],kernel="rbfdot",
         kpar=list(sigma=0.5),features=2)
plot(rotated(kpc)[flame$V3==1,1:2],col="red",pch=16,cex=1.5,
     xlab="KPC1",ylab="KPC2",xlim=c(-8.80,10.06),ylim=c(-7.0,10.70),
     main="sigma=0.5")
points(rotated(kpc)[flame$V3==2,1:2],cex=1.5,col="blue",pch=17)
##### sigma=1
kpc=kpca(~.,data=flame[,-3],kernel="rbfdot",
         kpar=list(sigma=1),features=2)
plot(rotated(kpc)[flame$V3==1,1:2],col="red",pch=16,cex=1.5,
     xlab="KPC1",ylab="KPC2",xlim=c(-8.80,10.06),ylim=c(-7.0,10.70),
     main="sigma=1")
points(rotated(kpc)[flame$V3==2,1:2],cex=1.5,col="blue",pch=17)
```

The performance of kernel PCA is determined by the kernel we choose and the associated parameters. Here we use the RBF kernel with different `sigma` and obtain the following kernel PC plots. When we increase the value of `sigma` to 0.5, the two classes can be roughly linearly separated on the first PC.

3 Isometric feature mapping (ISOMAP)

ISOMAP can be applied by using the `ISOMAP` in the `RDRToolbox` package. To install the `RDRToolbox` package:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install(version = "3.10")       #CHECK
BiocManager::install("RDRToolbox", version = "3.10")   #update to version 3.10
```

We take the Swiss roll data as an example.

```
library(RDRToolbox)
```

```
## This version of Shiny is designed to work with 'htmlwidgets' >= 1.5.
##     Please upgrade via install.packages('htmlwidgets').
```

```
library(rgl)
#install.packages("KODAMA")
library(KODAMA)
```

```
##
## Attaching package: 'KODAMA'
```

```
## The following object is masked from 'package:kernlab':
##
##     scaling
```

```
x=swissroll()
labels=c(rep(1,250),rep(2,250),rep(3,250),rep(4,250))
cols=c("black","red","blue","green")
# open3d()
# plot3d(x, col=cols[as.numeric(as.factor(labels))],box=FALSE,size=3)
# rgl.postscript("swissroll.pdf", fmt="pdf")
```

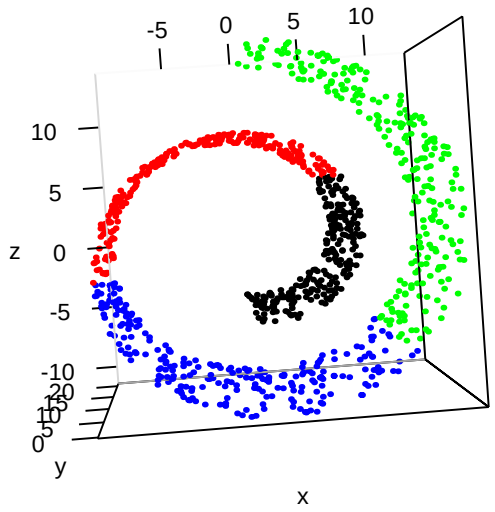

Figure 7: The swiss roll data.

Now we apply ISOMAP on this data.

```
par(mfrow = c(2, 2))
###k=3
swissIsomap = Isomap(data=x, dims=2, k=3)
```

```
## Computing distance matrix ... done
## Building graph with shortest paths (using 3 nearest neighbours) ... done
## Computing low dimensional embedding ... done
## number of samples: 1000
## reduction from 3 to 2 dimensions
## number of connected components in graph: 3
```

```
plotDR(data=swissIsomap$dim2, labels=(labels),axesLabels=c("", ""),
       legend=TRUE)
```

```
title(main="k=3")
###k=5
swissIsomap = Isomap(data=x, dims=2, k=5)
```

```
## Computing distance matrix ... done
## Building graph with shortest paths (using 5 nearest neighbours) ... done
## Computing low dimensional embedding ... done
## number of samples: 1000
## reduction from 3 to 2 dimensions
## number of connected components in graph: 1
```

```
plotDR(data=swissIsomap$dim2, labels=(labels),axesLabels=c("", ""),
       legend=TRUE)
title(main="k=5")
###k=10
swissIsomap = Isomap(data=x, dims=2, k=10)
```

```
## Computing distance matrix ... done
## Building graph with shortest paths (using 10 nearest neighbours) ... done
## Computing low dimensional embedding ... done
## number of samples: 1000
## reduction from 3 to 2 dimensions
## number of connected components in graph: 1
```

```
plotDR(data=swissIsomap$dim2, labels=(labels),axesLabels=c("", ""),
       legend=TRUE)
title(main="k=10")
###k=20
swissIsomap = Isomap(data=x, dims=2, k=20)
```

```
## Computing distance matrix ... done
## Building graph with shortest paths (using 20 nearest neighbours) ... done
## Computing low dimensional embedding ... done
## number of samples: 1000
## reduction from 3 to 2 dimensions
## number of connected components in graph: 1
```

```
plotDR(data=swissIsomap$dim2, labels=(labels),axesLabels=c("", ""),
       legend=TRUE)
title(main="k=20")
```

We can see that neither small k nor large k has good performance. Usually we choose k between 8 and 12.

We can also get a residual variance plot to see the intrinsic dimension of the data. In this example, the intrinsic dimension is actually 2.

```
swissResidual = Isomap(data=x, dims=1:3, k=10, plotResiduals=TRUE)
```

```
## Computing distance matrix ... done
## Building graph with shortest paths (using 10 nearest neighbours) ... done
## Computing low dimensional embedding ... done
## number of samples: 1000
## reduction from 3 to 123 dimensions
## number of connected components in graph: 1
```

Exercise: The `NCI60` cancer cell line microarray data in the `ISLR` consist of 6,830 gene expression measurements on 64 cancer cell lines. Try to apply the dimension reduction methods to visualise the data. You may want to try ICA later, because it takes long time to process high-dimensional data.

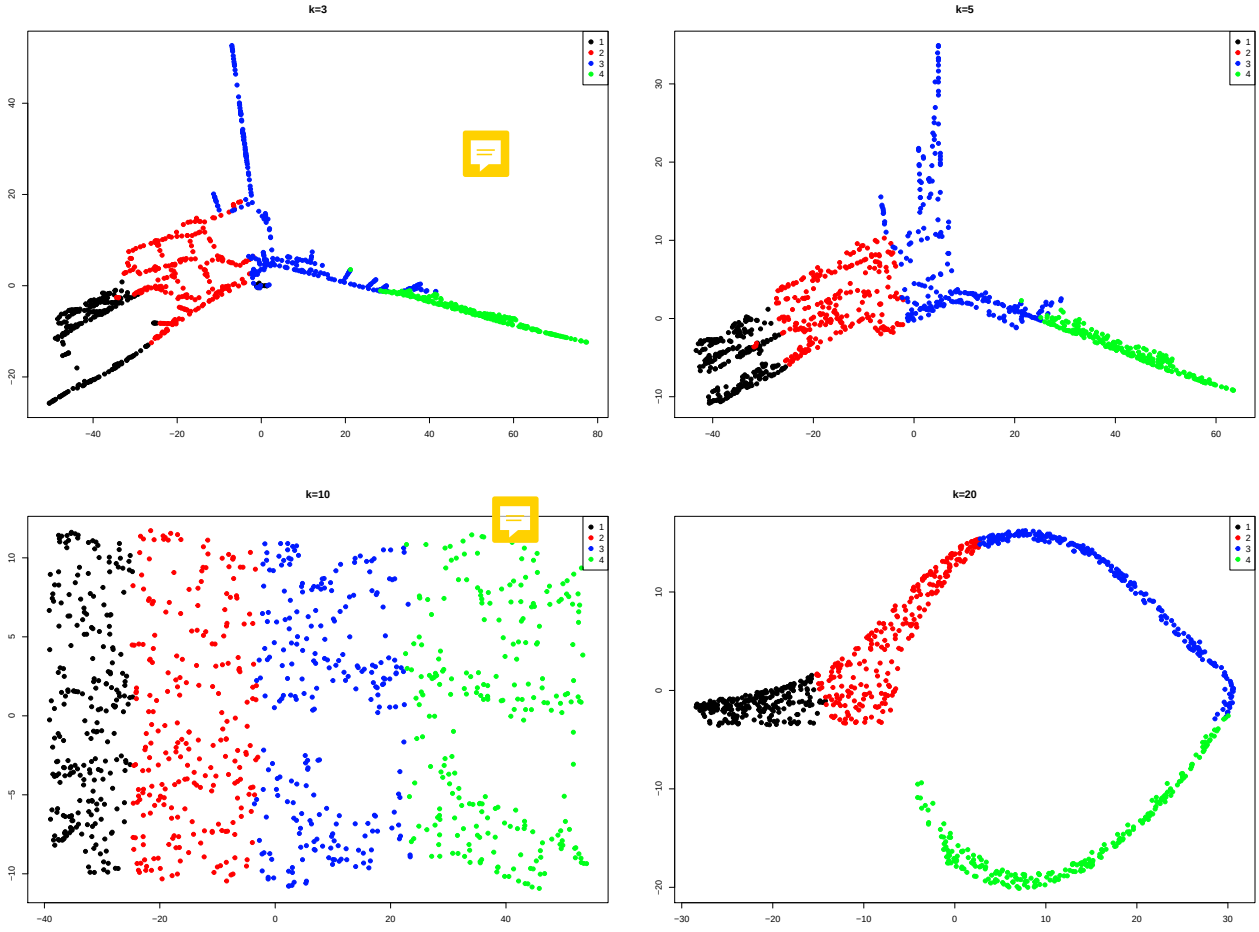

Figure 8: The ISOMAP projection of the Swiss roll data with different k.

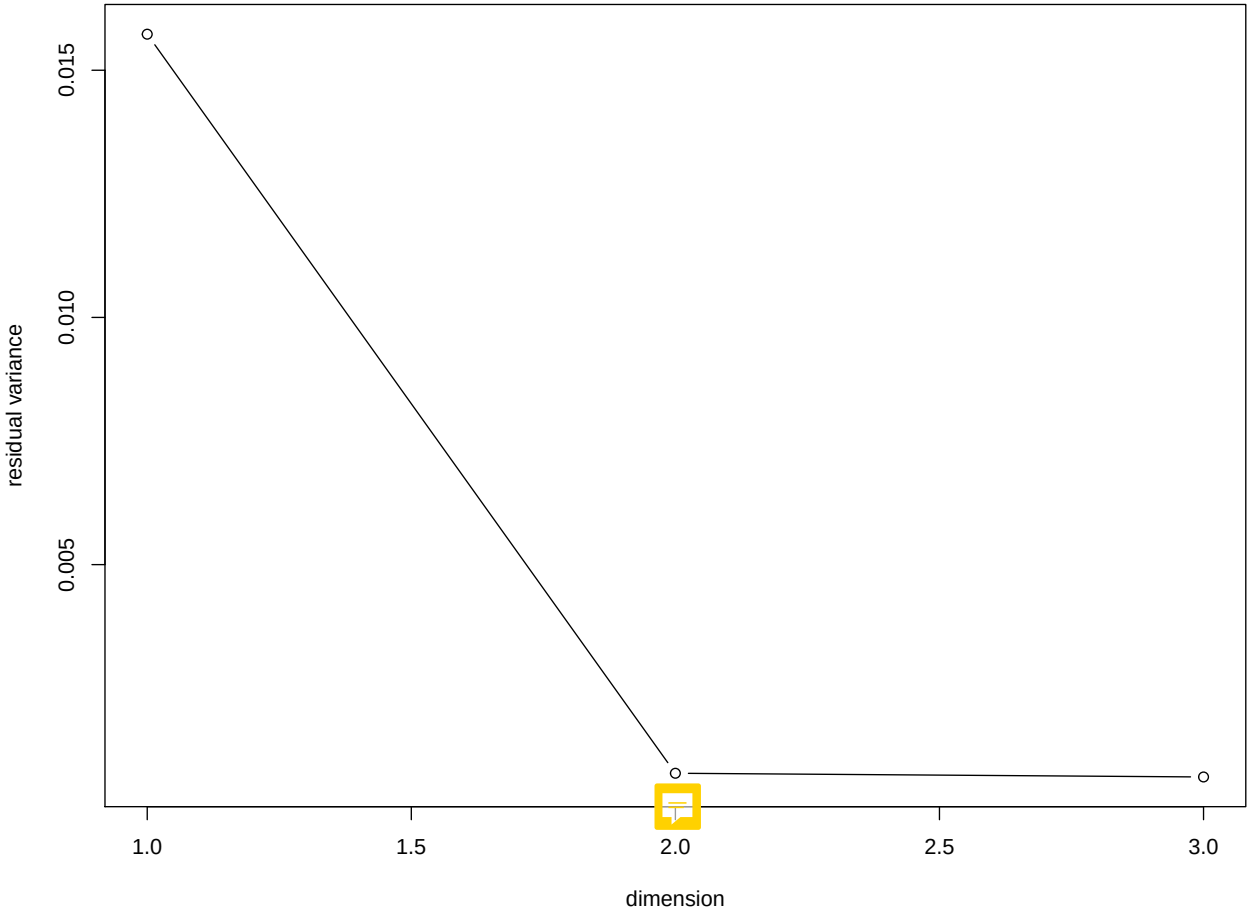

Figure 9: The swiss roll data.

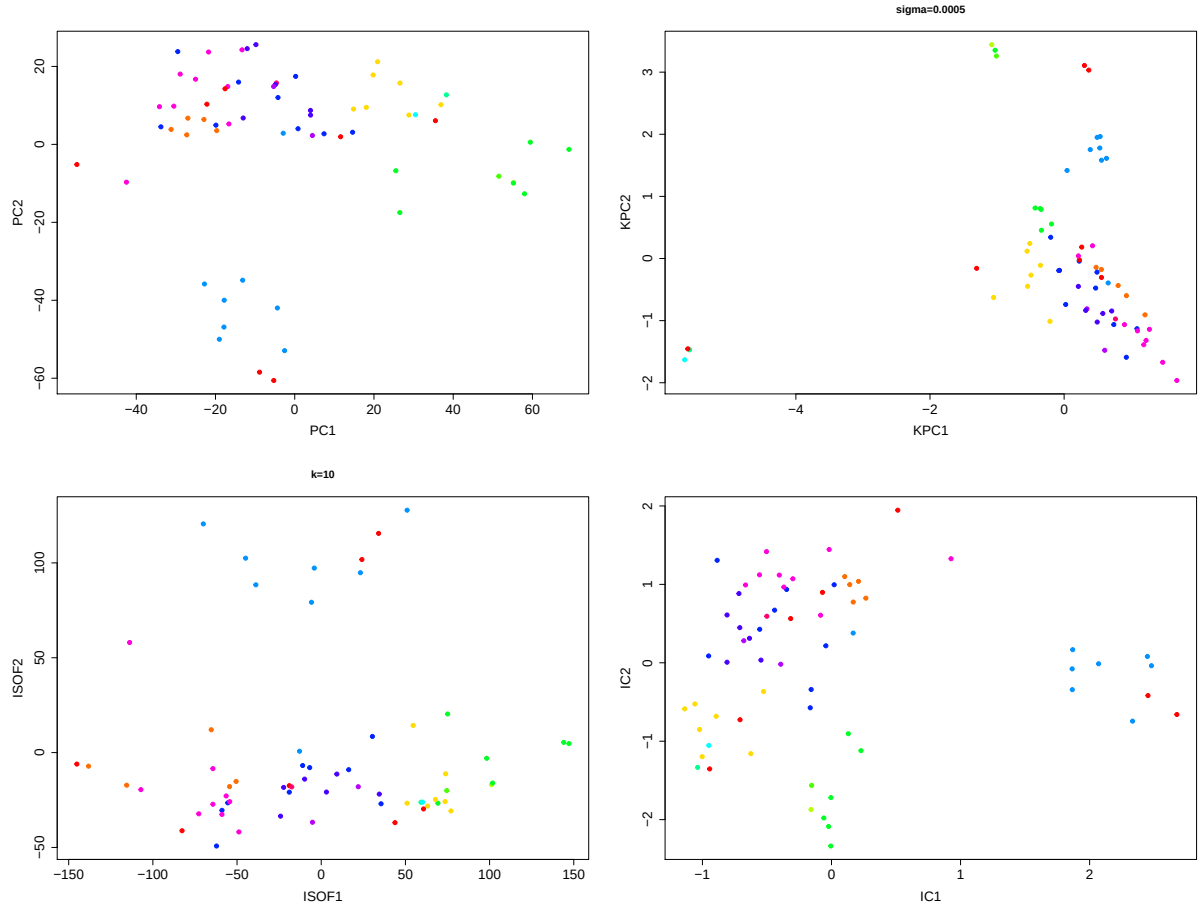

Figure 10: The gene expression data visualised by different methods in two dimensions.

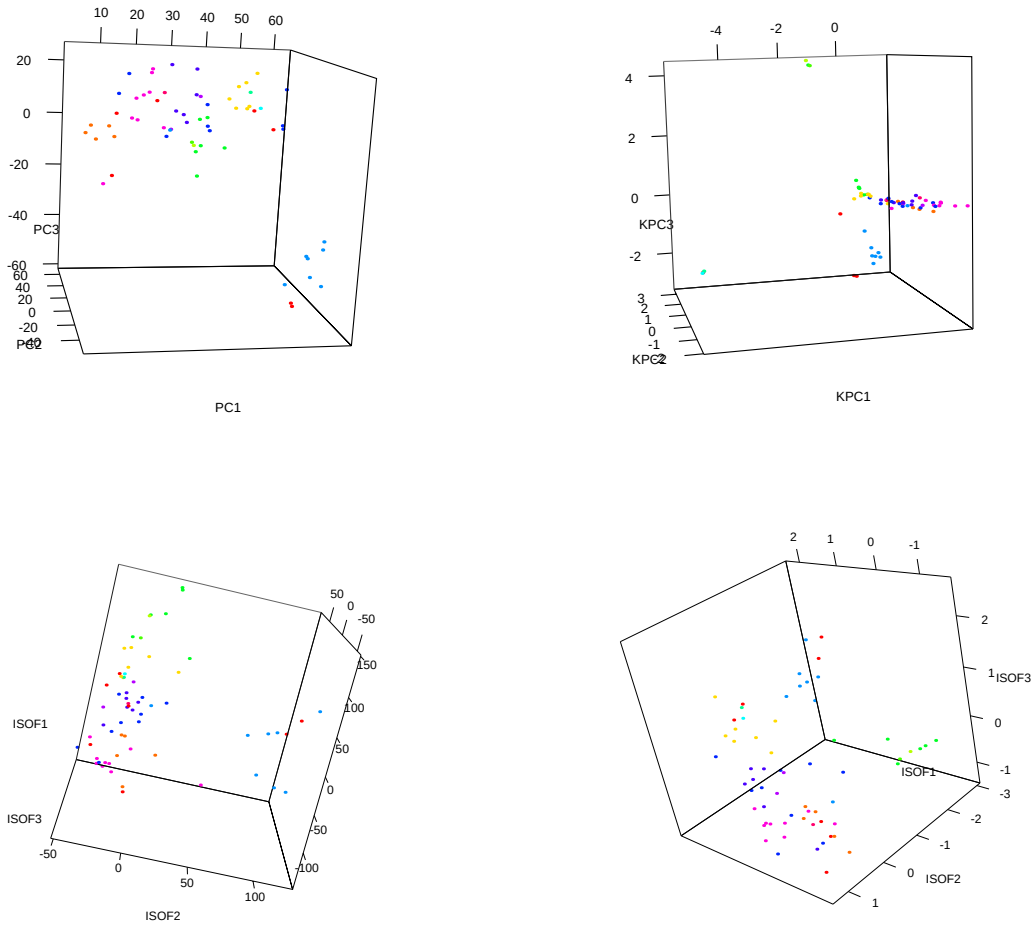

Figure 11: The gene expression data visualised by different methods in three dimensions.