8. Redução de dimensionalidade

8.1 Introdução

Em muitos casos, os conjuntos de dados a analisar são de dimensões elevadas e as variáveis possuem dependências entre si. Neste capítulo, apresentam-se métodos para reduzir a dimensionalidade dos dados. Estes métodos funcionam identificando conjuntos de variáveis não correlacionadas entre si que explicam a maior parte da variabilidade dos dados. Em termos algébricos, estamos interessados em matrizes de *rank* menor que permitam explicar os dados originais e reconstrui-los de forma o mais aproximada possível.

8.2 Análise de componentes principais

A técnica mais popular ao nível da redução de dimensionalidade de dados numéricos é a **análise de componentes principais** (ou PCA na denominação anglo-saxónica). A PCA consta de um procedimento algébrico que converte as variáveis originais (que são tipicamente correlacionadas) num conjunto de variáveis não correlacionadas (linearmente) que se designam por componentes principais (PCs) ou variáveis latentes. Assim, a PCA fornece um mapeamento de um espaço com N dimensões (em que N é o número de variáveis originais) para um espaço com M dimensões (onde M é tipicamente muito menor do que N).

As PCs são ordenadas pela quantidade decrescente de variabilidade (variância) que explicam. Cada PC é gerada de forma a explicar o máximo de variabilidade da parte ainda não explicada, tendo que ser ortogonal às PCs anteriores. É importante notar que a PCA é sensível à escala dos dados, pelo que se recomenda a sua normalização prévia.

A PCA consiste numa decomposição dos dados originais (uma matriz X) em duas matrizes: $T.P^T$. A matriz T tem o nome de *scores*, indicando as coordenadas dos exemplos iniciais (linhas de X) no novo sistema de coordenadas dado pelas PCs. As PCs determinadas são combinações lineares das variáveis originais, sendo os coeficientes destas no espaço original são dados pelas colunas da matriz P, sendo designados por *loadings*. Se considerarmos apenas as primeiras *k* componentes principais, isto implica considerarmos apenas as primeiras *k* colunas das matrizes T e P, obtendo-se uma

aproximação dos dados originais que será tanto mais precisa quanto maior é o valor de *k*.

Há várias formas de realizar a PCA em R, em diversos *packages* distintos. Uma das funções mais usadas, num dos packages base do R, é a função **princomp**. Esta tem como argumento obrigatório a matriz de dados ou data frame com os dados originais (necessariamente dados numéricos). Os argumentos opcionais permitem, por exemplo, filtrar linhas ou tratar de formas distintas os valores omissos (argumento *na.action*). O resultado é uma *list*, que inclui vários campos com os diversos resultados, incluindo os *loadings* e os *scores*.

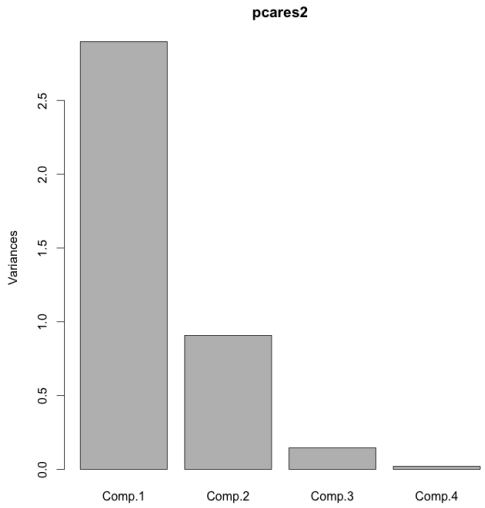
Uma função alternativa é a função **prcomp** que difere da anterior no método de cálculo da PCA, nos argumentos e na estrutura dos resultados. Por exemplo, esta função permite indicar explicitamente a normalização dos dados com o argumento "*scale*". Os exemplos seguintes ilustrarão as principais diferenças na estrutura dos resultados das duas funções.

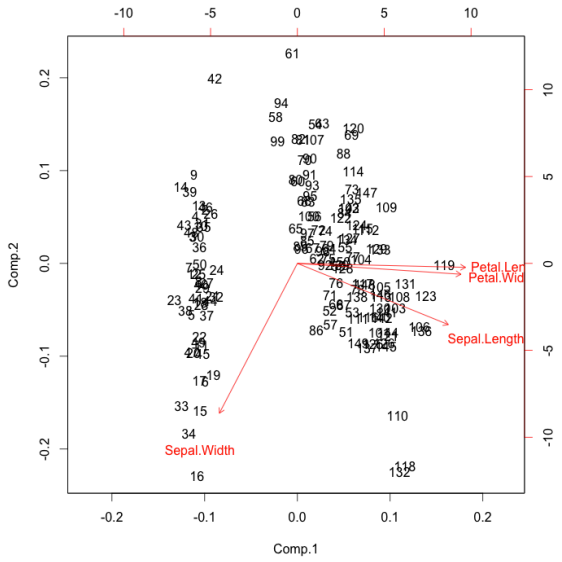
O exemplo seguinte usa o conjunto de dados *iris*, já usado anteriormente para demonstrar o uso das funções anteriores. Como o conjunto de dados possui 5 atributos, 4 dos quais numéricos, serão apenas estas colunas a serem consideradas (as 4 primeiras). Serão usadas ambas as funções anteriores:

```
> pcares = prcomp(iris[,-5], scale = T)
> pcares2 = princomp(scale(iris[,-5]))
> summary(pcares)
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.7084 0.9560 0.38309 0.14393
Proportion of Variance 0.7296 0.2285 0.03669 0.00518
Cumulative Proportion  0.7296 0.9581 0.99482 1.00000
> summary(pcares2)
Importance of components:
                          Comp.1    Comp.2     Comp.3      Comp.4
Standard deviation     1.7026571 0.9528572 0.38180950 0.143445939
Proportion of Variance 0.7296245 0.2285076 0.03668922 0.005178709
Cumulative Proportion  0.7296245 0.9581321 0.99482129 1.000000000
> pcares$rotation
                   PC1         PC2        PC3        PC4
Sepal.Length  0.5210659 -0.37741762  0.7195664  0.2612863
Sepal.Width  -0.2693474 -0.92329566 -0.2443818 -0.1235096
Petal.Length  0.5804131 -0.02449161 -0.1421264 -0.8014492
Petal.Width   0.5648565 -0.06694199 -0.6342727  0.5235971
> pcares2$loadings
Loadings:
```

```
             Comp.1 Comp.2    Comp.3      Comp.4
Sepal.Length 0.521   -0.377   -0.720  0.261
Sepal.Width  -0.269  -0.923    0.244      -0.124
Petal.Length  0.580             0.142      -0.801
Petal.Width   0.565             0.634  0.524
> plot(pcares2)
> biplot(pcares)
```

Os resultados demonstram que os métodos usadas por ambas as funções obtêm resultados similares mas ligeiramente distintos e que a estrutura das *lists* de resultados são também diferentes (e.g. a diferença de nomes de *loadings* para *rotation*). Os resultados dos gráficos (últimas duas linhas) são mostrados em seguida.





8.3 Decomposição em valores singulares

A decomposição em valores singulares (SVD em notação anglo-saxónica) é um método algébrico de fatorização de matrizes que pode ser usado em análise de dados para reduzir a dimensionalidade dos dados. De fato, a PCA é um caso particular da SVD, sendo a SVD um dos métodos aconselhados para calcular a PCA (usado pelo R na função **prcomp**).

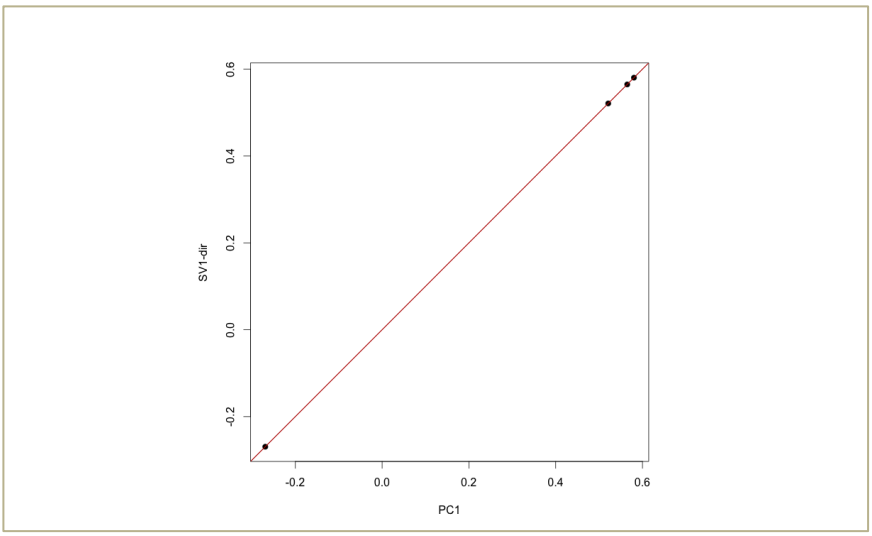
A SVD consta da fatorização de uma matriz M (de dimensões *n* x *m*) em $M = UDV^T$ onde: *U* é uma matriz *n x n*, *V* é uma matriz *m x m*, *A* tem dimensões *n x m*; além disso $U.U^T$ e $V.V^T$ são iguais à matriz identidade de dimensões *n* e *m*, respetivamente. As colunas de U são os vetores singulares esquerdos e as de V os vetores singulares direitos. A matriz D é uma matriz diagonal com os valores singulares de M.

Em R, a operação de SVD pode ser executada com a função **svd**. O principal argumento para esta função é a matriz (ou data frame) com dados numéricos, tal como acontece com as funções que realizam a PCA. O resultado é uma list, com três campos: *d* – matriz diagonal D; *u* – matriz U; *v* – matriz V.

Note-se que as colunas de v são equivalentes aos *loadings* resultantes da PCA (se os dados para esta forem normalizados). No exemplo seguinte ilustra-se esta característica com o conjunto de dados usado na secção anterior:

```
> svdres= svd(scale(iris[,-5]))
> svdres$v
           [,1]        [,2]       [,3]       [,4]
[1,]  0.5210659 -0.37741762  0.7195664  0.2612863
[2,] -0.2693474 -0.92329566 -0.2443818 -0.1235096
[3,]  0.5804131 -0.02449161 -0.1421264 -0.8014492
[4,]  0.5648565 -0.06694199 -0.6342727  0.5235971
>   plot(pcares$rotation[,1],   svdres$v[,1],   pch=19,   xlab="PC1",
ylab="SV1-dir")
> abline(0,1, col="red")
```

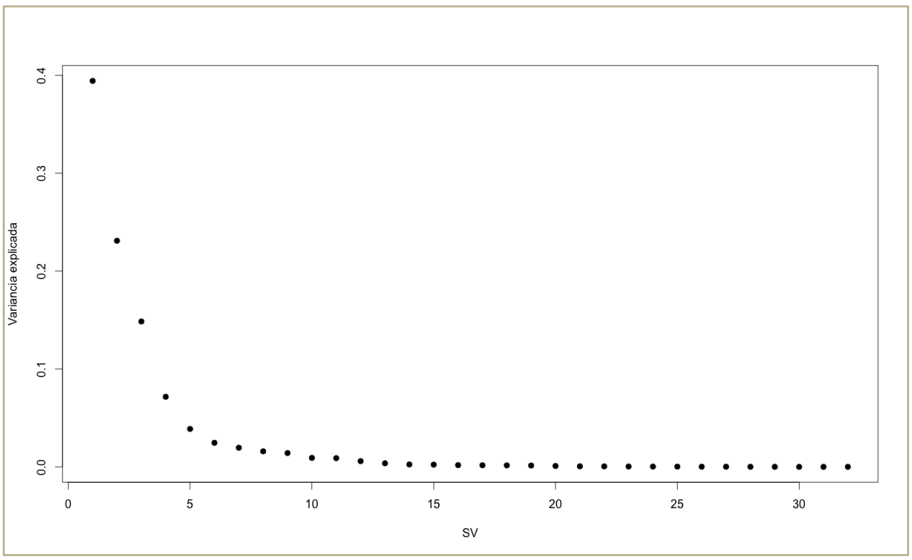
O gráfico gerado pelas duas últimas linhas do exemplo anterior será o seguinte:



No exemplo seguinte demonstra-se a aproximação de uma matriz de dados usando um número diferentes de valores singulares. A primeira linha faz o download do ficheiro (podendo ser substituída por um download direto usando o seu browser) e a segunda linha carrega o ficheiro de dados. A linha seguinte executa o processo de SVD, enquanto a última permite visualizar a componente da variância explicada por cada valor singular.

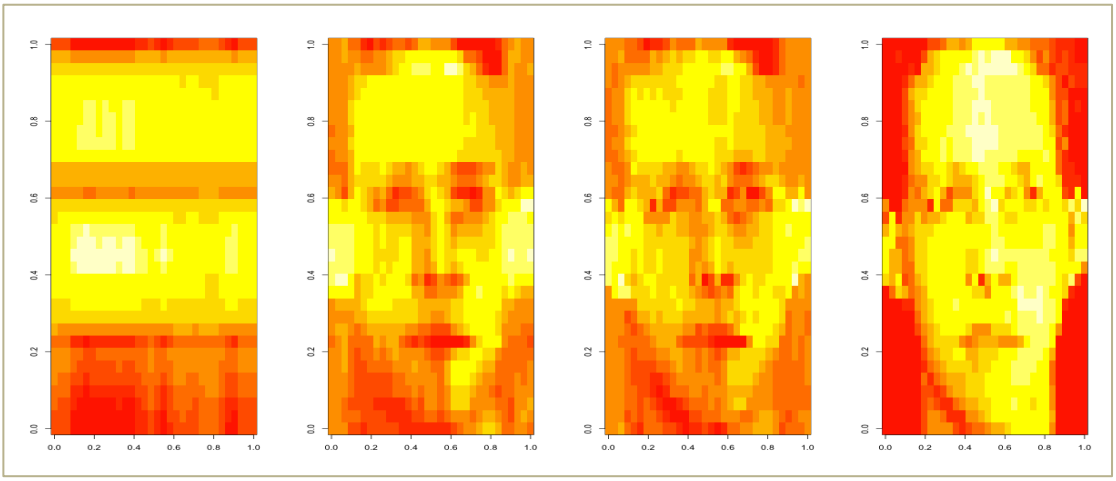
```
>download.file("https://spark-public.s3.amazonaws.com/dataanalysis/
face.rda", destfile="face.rda")
> load("face.rda")
> svd1 <- svd(scale(faceData))
>        plot(svd1$d^2/sum(svd1$d^2),pch=19,xlab="SV",ylab="Variancia
explicada")
```

O gráfico seguinte mostra o resultado da última linha:



Nos comandos seguintes mostra-se como podemos criar aproximações aos dados originais usando 1, 5 e 10 valores singulares, respetivamente. Estas aproximações são mostradas em forma de gráfico em seguida, comparando-se com os dados originais (último gráfico).

```
> aprox1 <- svd1$u[,1] %*% t(svd1$v[,1]) * svd1$d[1]
> aprox5 <- svd1$u[,1:5] %*% diag(svd1$d[1:5])%*% t(svd1$v[,1:5])
>     aprox10     <-     svd1$u[,1:10]     %*%     diag(svd1$d[1:10])%*%
t(svd1$v[,1:10])
> par(mfrow=c(1,4))
> image(t(aprox1)[,nrow(aprox1):1])
> image(t(aprox5)[,nrow(aprox5):1])
> image(t(aprox10)[,nrow(aprox10):1])
> image(t(faceData)[,nrow(faceData):1])
```



Exercícios resolvidos

1. Neste exercício será utilizado o data frame "wines" do package kohonen. Este dataset consiste na análise química a 177 amostras de vinho cultivados na mesma região, provenientes de 3 castas.

a) Importe os dados do dataset wines do package kohonen e explore o conteúdo das variáveis wine, vintages e wine.classes. Verifique o seu tipo e sumarie os seus dados.

b) Utilizando a função *prcomp*, efectue a análise dos principais componentes dos dados (normalizados).

c) Apresente em forma de gráfico o resultado da função anterior e analise quantos componentes serão necessários considerar para explicar 90 % da variabilidade dos dados.

d) Proceda ao mesmo tipo de análise utilizando a função *princomp*. Verifique as diferenças ao nível da estrutura de resultados e ao nível dos valores numéricos calculados (e.g. para os campos rotation / loadings)

e) Utilizando SVD calcule as matrizes de scores e loadings para o dataset anterior. (Obs: SVD consta da fatorização de uma matriz M (de dimensões *n* x *m*) em $M = UDV^T$. A matrix de loadings corresponde à matriz V e a matriz de scores à matriz U*D.

f) Calcule a importância de cada um dos componentes sabendo que é dada pela fracção da variância explicada, isto é, FV = wines.svd$d^2 / sum(wines.svd$d^2), sendo wines.svd o resultado da aplicação da função svd aos dados.

g) Apresente três gráficos que representem a variância (**wines.svd$d^2 / (nrow(wines) - 1)**), variância relativa (**FV**) e a percentagem acumulada da variância total usando a função **cumsum**.

h) Sabendo que a matriz de *scores* mostra a posição de cada amostra relativamente a cada PC, produza um gráfico que apresente a distribuição dos exemplos considerando os 2 principais componentes PC1 e PC2. Coloque símbolos e cores diferentes para cada classe.

i) Usando a função **pairs**, apresente a posição das amostras considerando os 4 principais componentes (use os atributos **pch** e **col** para atribuir símbolos e cores diferentes para cada classe).

```
# a) carregar dados
install.packages("kohonen")
library(kohonen)
data(wines, package = "kohonen")
```

```
class(wines)
typeof(wines)
colnames(wines)
wines
summary(wines)
class(vintages)
vintages
levels(vintages)
table(vintages)
class(wine.classes)
wine.classes
# b, c)
wines.pca = prcomp(wines, scale = TRUE)
plot(wines.pca, main = "PCA", xlab = "Componentes principais")
summary(wines.pca)   # verificar o campo Cumulative Proportion - necessarias 8 PCs
# d)
wines.pca2 = princomp(scale(wines))
plot(wines.pca2, main = "PCA")
summary(wines.pca2)
# e)
wines.sc = scale(wines)
wines.svd = svd(wines.sc)
wines.scores = wines.svd$u %*% diag(wines.svd$d)
wines.loadings = wines.svd$v
# f)
wines.vars <- wines.svd$d^2 / (nrow(wines) - 1)
wines.relvars <- wines.vars / sum(wines.vars)
variances <- 100 * round(wines.relvars, digits = 3)
variances[1:5]
wines.relvars2 = wines.svd$d^2 / sum(wines.svd$d^2)
# g)
barplot(wines.vars, main = "Variancias",
        names.arg = paste("PC", 1:13))
barplot(wines.relvars, main = "Variancias relativas",
        names.arg = paste("PC", 1:13))
barplot(cumsum(100 * wines.relvars),
        main = "Variancias acumuladas (%)",
        names.arg = paste("PC", 1:13), ylim = c(0, 100))
# h)
plot(wines.scores[,1:2], type = "p",
     xlab = paste("PC 1 (", variances[1], "%)", sep = ""),
     ylab = paste("PC 2 (", variances[2], "%)", sep = ""),
     pch = wine.classes, col = wine.classes)
abline(h = 0, v = 0, col = "gray")
legend(-4, -2, legend = levels(vintages), col = 1:3, pch = 1:3)
```

```
# i)
pairs(wines.scores[,1:4],pch = wine.classes, col = wine.classes)
```

9. Clustering

9.1 Introdução

O objetivo do clustering é o de realizar o agrupamento de entidades / exemplos de um conjunto de dados (as linhas de uma matriz ou data frame) com base na similaridade entre estes. Uma tarefa similar passa pelo agrupamento das colunas (variáveis).

Embora não se pretenda neste texto abordar em detalhe as variantes, as formulações e os algoritmos para realizar esta tarefa note-se que as abordagens para este problemas variam sobretudo nas métricas usadas com vista à definição de similaridade, nos algoritmos usados para realizar o agrupamento, na forma que assume a saída destes mesmos algoritmos e, finalmente, na forma de visualizar e interpretar os resultados.

Neste texto, abordaremos duas abordagens distintas, o clustering hierárquico e o clustering k-means, focando na forma como estas se podem implementar com a linguagem R.

9.2 Clustering hierárquico

O clustering hierárquico tem como principal característica o tipo de resultados que gera que está intimamente ligado com o processo usado na sua construção. Neste caso, o resultado final do processo é uma árvore binária que representa possíveis divisões dos dados em clusters. Assim, na raiz todos os dados estão agrupados num único cluster, e ao descer na árvore os clusters vão-se dividindo de forma binária, ou seja, em cada nó da árvore são criados dois clusters pela divisão de um único.

O método mais usado para construir estas árvores é designado por aglomerativo, pois inicia-se com um cluster para cada exemplo, ou seja parte das folhas da árvore em direção à raiz, no processo de construção. Em cada iteração, vão-se juntando dois clusters e criando um único (criando nós na árvore) até se atingir o ponto em que todos os clusters estão unidos num único (raiz da árvore).

O processo é baseado numa matriz de distâncias onde estão guardadas as distâncias entre todos os pares de objetos; esta matriz é construída aplicando uma

métrica de similaridade sobre as linhas da matriz inicial de dados. Esta pode ser, por exemplo, baseada em métricas clássicas de distância como a distância euclidiana ou a distância de Manhattan ou, em alternativa, métricas baseadas na correlação (e.g. Spearman ou Pearson).

Para obter uma matriz de distâncias em R, pode usar-se a função **dist**. Esta recebe como argumento uma matriz ou um data frame com valores numéricos, retornando a matriz de distâncias. Por omissão, o método usado para o cálculo das distâncias é a distância euclidiana, mas podem ser escolhidas outras opções como a distância de Manhattan ou a distãncia de Minkowski, através da definição do argumento *method*.

O exemplo seguinte mostra como se pode calcular a matriz de distâncias para um exemplo com pontos gerados de forma aleatória formando três clusters naturais. Note que o resultado é uma matriz triangular dado que as matrizes de distâncias são simétricas.

```
> x = rnorm(12,mean=rep(1:3,each=4),sd=0.2)
> y = rnorm(12,mean=rep(c(1,2,1),each=4),sd=0.2)
> dataFrame <- data.frame(x=x,y=y)
> dist(dataFrame)
1      2      3      4      5      6      7      8      9     10     11
2  0.3412
3  0.5749 0.2410
4  0.2638 0.5258 0.7186
5  1.6942 1.3582 1.1195 1.8067
6  1.6581 1.3196 1.0833 1.7808 0.0815
7  1.4982 1.1662 0.9256 1.6013 0.2111 0.2166
8  1.9915 1.6909 1.4564 2.0284 0.6170 0.6979 0.6506
9  2.1363 1.8317 1.6783 2.3567 1.1834 1.1150 1.2858 1.7646
10 2.0642 1.7699 1.6310 2.2923 1.2384 1.1655 1.3206 1.8351 0.1409
11 2.1470 1.8518 1.7107 2.3746 1.2815 1.2107 1.3736 1.8699 0.1162
0.0831
12 2.0566 1.7466 1.5865 2.2723 1.0770 1.0077 1.1774 1.6622 0.1084
0.1912 0.2080
```

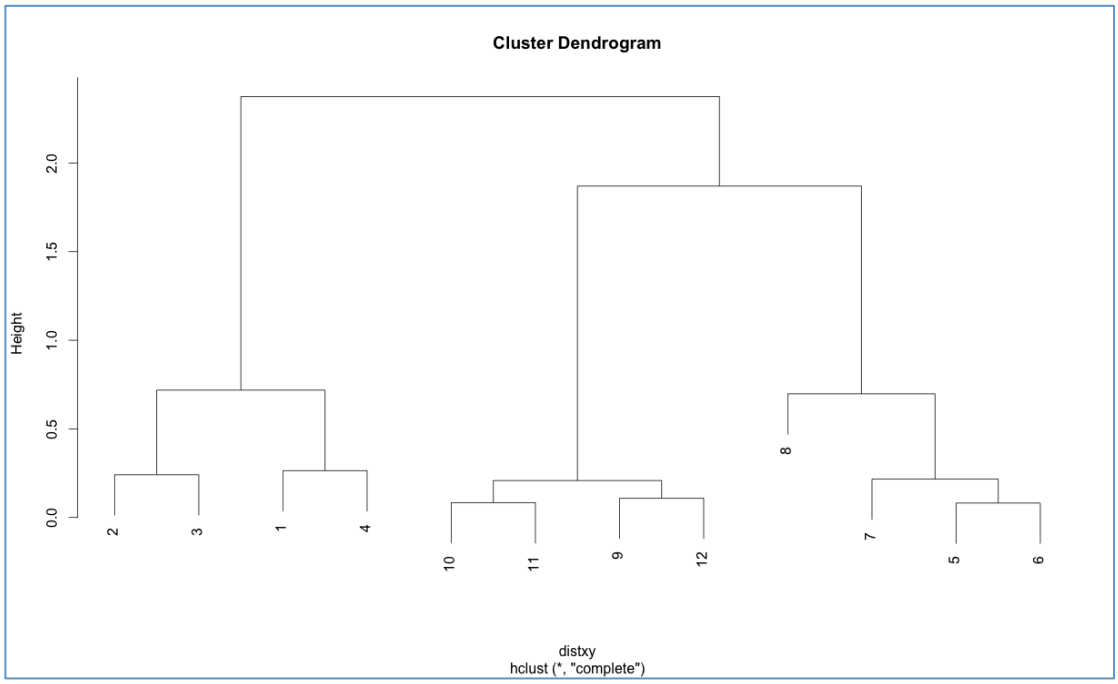
Para executar o processo de clustering hierárquico usa-se a função **hclust**. Esta recebe como argumento a matriz com as distâncias. Como argumento opcional (*method*), a função permite escolher a forma como se calculam distâncias entre clusters com mais do que um ponto nos passos intermédios do algoritmo. Os valores possíveis para esta opção incluem "*complete*" (valor por omissão, indica que a distância entre 2 clusters é a maior distância entre qualquer par de elementos dos 2 clusters), "*single*"

(indica que a distância entre 2 clusters é a menor distância entre qualquer par de elementos dos 2 clusters) e "*average*" (indica que a distância entre 2 clusters é a média das distância entre todos os pares de elementos dos 2 clusters).

Veja-se um exemplo de aplicação da função **hclust** com os dados do exemplo anterior:

```
> distxy = dist(dataFrame, method = "euclidean")
> hc = hclust(distxy)
> plot(hc)
```

O resultado seria o seguinte:



Para ilustrar estes métodos com um exemplo usando um conjunto de dados de maior dimensão, tome-se como ponto de partida os dados do conjunto de dados iris ja usados em capítulos anteriores. No exemplo seguinte, os dados são inicialmente standardizados para que a métrica de distância pese de forma uniforme as várias variáveis, podendo visualizar-se o efeito com um gráfico do tipo boxplot. Note-se que o atributo *Species* (posição 5) não é utilizado neste processo, sendo usados apenas os atributos numéricos.

```
> iris.sc = scale(iris[,1:4])
```

```
 > boxplot(iris[,1:4])
 > boxplot(iris.sc)
```

Em seguida, procede-se ao cálculo da matriz de distâncias usando distâncias euclideanas e realiza-se o processo de clustering hierárquico usando a funçãoo hclust.

```
> dist.iris = dist(iris.sc, method = "euclidean")
> hc.complete = hclust(dist.iris, method = "complete")
> plot(hc.complete, cex = 0.4)
```

No gráfico gerado pelo código anterior, note-se que o número de pontos torna o resultado difícil de interpretar. Em muitos casos, faz sentido comparar a divisão dos dados em clusters com conhecimento próprio do problema. Neste caso, iremos usar o campo iris$Species como divisão natural dos dados (note-se que este campo não foi usado para realizar o clustering). O exemplo seguinte usa este campo para colocar as folhas da árvore e, assim, verificar os clusters gerados com os clusters naturais, usando dois valores distintos para o argumento *method* na função hclust.

```
> dist.iris = dist(iris.sc, method = "euclidean")
> hc.complete = hclust(dist.iris, method = "complete")
> my.plot.hc(hc.complete,lab.col=as.integer(iris$Species)+1, cex=0.4
> hc.average = hclust(dist.iris, method = "average")
> my.plot.hc(hc.average,lab.col=as.integer(iris$Species)+1,cex=0.4)
```

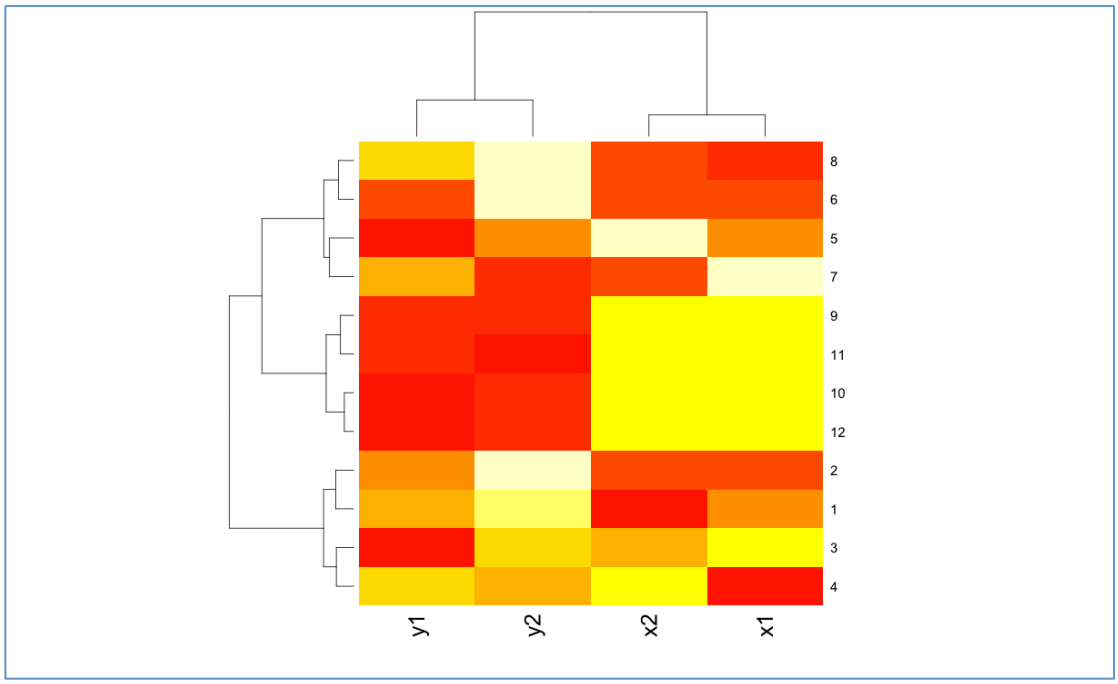
A função *my.plot.hc* usada para definir o gráfico é dada em seguida:

```
my.plot.hc = function(hclust, lab = 1:length(hclust$order),
                      lab.col=rep(1, length(hclust$order)), hang = 0.1, ...)
{
  y = rep(hclust$height, 2)
  x = as.numeric(hclust$merge)
  y = y[which(x<0)]
  x = x[which(x<0)]
  x = abs(x)
  y = y[order(x)]
  x = x[order(x)]
  plot(hclust, labels = F, hang = hang, ...)
  text(x = x, y = y[hclust$order]- (max(hclust$height) * hang),
       labels = lab[hclust$order], col = lab.col[hclust$order], srt = 90,
       adj = c(1,0.5), xpd = NA, ...)
}
```

Uma representação gráfica dos dados relacionada com o clustering hierárquico são os *heatmaps*, que podem ser construídos em R com a função **heatmap**. Esta permite representar os dados como uma imagem onde cada valor da matriz de dados é representado com uma célula cuja cor varia consoante o valor respetivo, num gradiente de cores que pode ser configurado. Os *heatmaps* tipicamente incluem as árvores criadas por clustering hierárquico quer ao nível das linhas, quer ao nível das colunas de dados. Note o exemplo seguinte:

```
> x1 <- rnorm(12,mean=rep(1:3,each=4),sd=0.2)
> x2 <- rnorm(12,mean=rep(1:3,each=4),sd=0.2)
> y1 <- rnorm(12,mean=rep(c(1,2,1),each=4),sd=0.2)
> y2 <- rnorm(12,mean=rep(c(1,2,1),each=4),sd=0.2)
> df2 = data.frame(x1, x2, y1, y2)
> heatmap(as.matrix(df2))
```

Cuja representação gráfica é a seguinte:



9.3 Clustering *k-means*

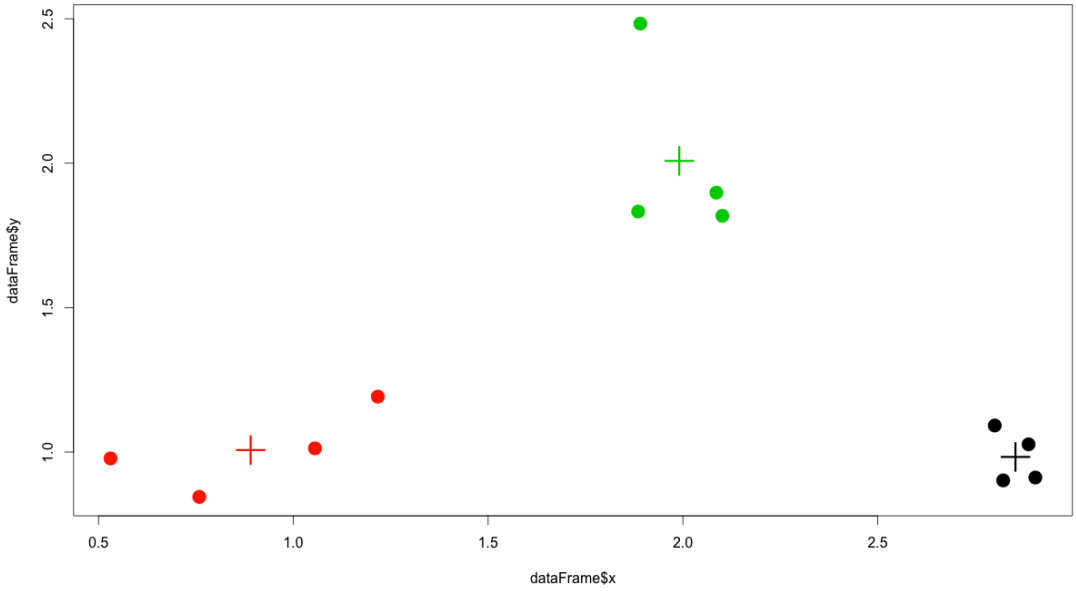
O problema de clustering *k-means* constitui uma das possíveis formulações do clustering, onde o objetivo é o de minimizar a média do quadrado das distâncias de cada

ponto ao centro do cluster a que pertence. Nesta formulação, o número de clusters é dado como parâmetro de entrada (designado por *k*). Dado que este problema apresenta uma significativa complexidade (dentro da classe dos problemas NP-completos), métodos heurísticos são tipicamente usados na resolução do problema.

A função **kmeans** permite resolver um problema de clustering k-means dado um conjunto de dados e o valor de *k* (parâmetro *centers*). Veja um exemplo de seguida, usando os dados criados na secção anterior:

```
> resKmeans <- kmeans(dataFrame, centers=3)
> resKmeans$cluster
 [1] 2 2 2 2 3 3 3 3 1 1 1 1
>   plot(dataFrame$x,dataFrame$y,   col=resKmeans$cluster,   pch=19,
cex=2)
> points(resKmeans$centers, col=1:3, pch=3, cex=3, lwd=3)
```

O resultado da última linha será o gráfico seguinte:



Note-se que, em casos reais, dado que o método de otimização usado é heurístico há a necessidade de se correr o algoritmo por diversas vezes, de forma a aumentar a qualidade da solução final encontrada. Este número de repetições pode ser controlado pelo parâmetro *nstart*.

De forma idêntica ao método anterior, ir-se-á demonstrar este método usando o conjunto de dados iris.

```
> kmeans.iris = kmeans(iris[,1:4], centers = 3, nstart = 10000)
> table(kmeans.iris$cluster, iris$Species)
     setosa versicolor virginica
   1      0          2        36
   2     50          0         0
   3      0         48        14
```

Note-se que o segundo comando executado permite comparar o resultado da função kmeans com o agrupamento natural dos dados pelo campo *Species*. Esta comparação permite aferir da conformidade dos clusters gerados e dos clusters naturais.

Exercícios resolvidos

1. Neste grupo de exercícios vamos considerar um subgrupo das amostras presentes do *dataset wines* do package kohonen, já trabalhado no capítulo anterior.

a) Como primeiro passo, retire 30 amostras de forma aleatória.

b) Calcula as matrizes de distâncias, entre as várias amostras, considerando vários tipos de distância: **euclidiana**, **manhattan** e **máxima**

c) Utilizando clustering hierárquico agrupe as amostras considerando as distâncias anteriormente calculadas. Teste a função com vários métodos ("**single**", "**complete**" e "**average**"). Visualize os resultados de cada opção. Tente verificar até que ponto o clustering realizado está de acordo com as classes presentes na variável *vintages*.

d) Considere a função **kmeans** do R para agrupar os dados do dataset wines em 3 grupos. Verifique quantas amostras foram colocadas num grupo diferente do real (assumindo a classificação dada pela variável vintages).

```
# a)
library(kohonen)
data(wines, package = "kohonen")
indexes = sample(nrow(wines), 30)
newData = wines.sc[indexes,]
# b)
newData.Euclidean = dist(newData)
newData.Maximum = dist(newData, method ="maximum")
newData.Manhattan = dist(newData, method ="manhattan")
# c)
```

```
newData.hcsingle <- hclust(newData.Euclidean, method = "single")
plot(newData.hcsingle, labels = vintages[indexes])

newData.hccomplete <- hclust(newData.Euclidean, method = "complete")
plot(newData.hccomplete, labels = vintages[indexes])

newData.hcsingle <- hclust(newData.Manhattan, method = "single")
plot(newData.hcsingle, labels = vintages[indexes])

newData.hccomplete <- hclust(newData.Manhattan, method = "complete")
plot(newData.hccomplete, labels = vintages[indexes])

newData.hccomplete <- hclust(newData.Manhattan, method = "average")
plot(newData.hccomplete, labels = vintages[indexes])
 # d)
wines.km = kmeans(wines.sc, 3)
table(vintages, wines.km$cluster)
```