Lab5	配列 **seriation**/多维标度法 **MDS** /聚类 **clustering** /分类 **classification**

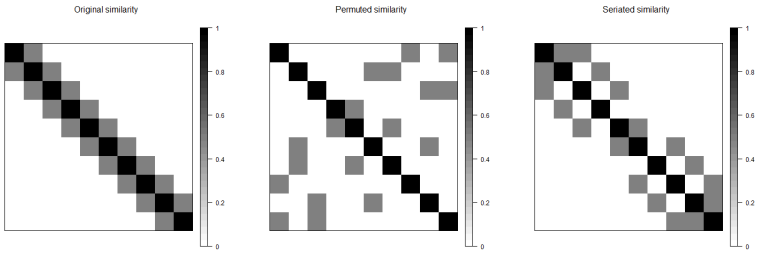
1 配列 (seriation)

程序包 seriation 中的函数 seriate() 可以对相似度矩阵或距离矩阵的变量重新排序（行和列同时重新排序），使得重排后相似度高的变量彼此靠近。

```
library(seriation)
order= seriate(S) #S: similarity matrix,或 dist matrix
pimage(S, order) #重排后画出d的热图
```

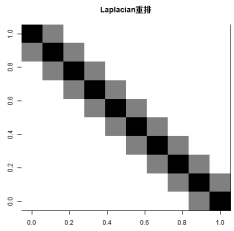
例 1. 我们生成一个只有主对角和次对角非 0 的相似度矩阵，打乱次序后用 seriate 函数重排次序。

```
set.seed(1)
n=10
S=diag(n)
diag(S[-1,][,-n])=1
S=(S+t(S))/2
                    [1,] 1.0 0.5 . . . . . . . .
                    [2,] 0.5 1.0 0.5 . . . . . . .
                    [3,] . 0.5 1.0 0.5 . . . . . .
                    [4,] . . 0.5 1.0 0.5 . . . . .
                    [5,] . . . 0.5 1.0 0.5 . . . .
                    [6,] . . . . 0.5 1.0 0.5 . . .
                    [7,] . . . . . 0.5 1.0 0.5 . .
                    [8,] . . . . . . 0.5 1.0 0.5 .
                    [9,] . . . . . . . 0.5 1.0 0.5
                    [10,] . . . . . . . . 0.5 1.0
pimage(S,col=gray(32:0 /32),main="Original similarity")
#S: Original similiarity matrix 下图左
i=sample(1:n) ; S.per=S[i,i] #随机置换打乱次序， 下图中
pimage(S.per,col=gray(32:0 /32),main="Permuted similarity")
order= seriate(S.per)
pimage(S.per, order,col=gray(32:0 /32),main="Seriated similarity")
# seriation重排后，下图右
```



seriate() 函数有众多选项，前述例子中我们使用了缺省选项，效果不好。下面我们使用拉普拉斯矩阵的最小非 0 特征根和一维 cMDS 坐标进行排序/配列（函数：ser）：

```
#####用Laplacian矩阵的最小非0特征根的特征向量排序， ser函数：
ser =function(S){ #S: similarity matrix
S=as.matrix(S)
n=nrow(S )
apply(S ,1,sum)->d #degree
diag(d)-S ->L #Laplacian
eigen(L)[[2]][,n-1]->v #L的最小非0特征根对应的特征向量
o=order(v) #v的分量的次序（order）
S.seriate=S[o,o] #根据次序重排S.per的行和列
image(S.seriate[,n:1] ,col=gray(32:0 /32))
return(S.seriate)
}
ser(S.per) #下图
```



综上，至少对本例来说，Laplacian 方法配列效果最好。

练习. **欧洲语言相似度的另一版本（语言次序打乱的版本）**

```
S=read.table("http://staff.ustc.edu.cn/~ynyang/vector/data/Euro-language1.txt")
S=as.matrix(S)
n=nrow(S)
image(S[,n:1] , col=gray(32:0 /32) )
ser(S)
```

试应用 seriate **函数** (in package seriation) **和** Laplacian **方法（上述** ser **函数）重排各个语言，即同时重排矩阵** S **的行和列使得相似度高的语言彼此尽量靠近。**

2 多维标度法 MDS：cMDS、isoMDS

经典的 MDS 的 R 函数为 cmdscale, 非度量型 MDS 函数为 isoMDS (library(MASS)).

```
cmdscale(d, k=2) # d: symmetric distance matrix or
# or distance object produced by dist or as.dist
isoMDS(d, k=2) # library(MASS)
```

两者都要求输入距离矩阵，数学上距离矩阵是一个 $n \times n$ 方阵，但 R 中的距离矩阵具有特殊的 dist 格式，比如 4 个物件 Item1-4 的两两距离在 R 中表示如下：

```
      Item1 Item2 Item3
Item2   2.4
Item3   2.2   2.7
Item4   3.4   3.6   2.2
```

产生 dist 格式的函数如下

```
d=dist(x,method=) # x: data.frame,
  #method= "euclidean","maximum",
    "manhattan","canberra","binary","minkowski"
d=as.dist(D) #D: 通常的距离矩阵，转换为dist格式
```

前者 (dist) 计算数据矩阵的行之间的距离（dist 格式），后者 (as.dist) 把通常的距离矩阵（方阵）转化成 R 的 dist 格式。

例 **2.** 欧洲 11 种语言的标度方法.

```
S=read.table("http://staff.ustc.edu.cn/~ynyang/vector
/data/Euro-language1.txt",head=T)
lang=rownames(S)
D=(10-S)^0.5 #相似度转化为距离
d=as.dist(D) #R 中距离矩阵是下三角阵的形式
#Classical MDS
mymds= cmdscale(d , k=2) #d: 距离格式，k: 维数
mymds= cmdscale(D, k=2) #D: 一般矩阵，k: 维数
plot(mymds,type="n",xlab="x",ylab="y")
text(mymds,lang,cex=0.8)
#non-metric MDS:
library(MASS)
mynmmds= isoMDS(d,k=2)
plot(mynmmds$points,type="n",xlab="x",ylab="y")
text(mynmmds$points,lang,cex=0.8)
```

例 2. car 数据集是 40 个消费者对 23 种汽车的 8 项指标打分的平均值，最高分为 6（最好），最低分为 1。计算 23 种汽车之间的 canberra 距离矩阵, 并应用 cMDS：

```
car=read.table("http://staff.ustc.edu.cn/~ynyang/vector/data/car.txt")
D=dist(t(car),method = "canberra")
#计算各列之间的距离
mymds= cmdscale(D , k=2) #D: 距离矩阵，k: 维数
```

（应用 dist 函数计算距离时选择 canberra），并做经典的 MDS (k=2)，画出散点图。

练习. **二战国家主要人物的相似性打分（分数为 1-9，1：最相似，9：最不相似）：**

```
leaders=read.table("http://staff.ustc.edu.cn/~ynyang/vector/data/leaders.txt")
```

试应用度量型和非度量型的 MDS 方法画出这些人物的二维坐标表示，并研究他们之间的相似性。

3 聚类 (clustering)

3.1 聚合（agglomeration）层次聚类分析

聚合层次聚类分析的 R 函数为

```
myclust = hclust(d, method="complete")
plot(myclust)
```

其中 d 为距离（相异度）矩阵 (由 dist(x) 计算得到, x 为数据矩阵), method 为连结（linkage）的方法，可选项为 “complete”, “single”, “average”, “centroid”, “median”, “ward.D”, “ward.D2” 等等。使用函数 cutree 可得到聚集成指定类别个数为 k 时的具体的聚类结果，即每个样本点所属的类的标号：

```
cutree(myclust,k=3)
# k: number of cluster
```

例 **3.** 欧洲城市温度数据. 只使用该数据的前 23 行即欧洲 23 个首都，前 12 列即 12 个月的平均温度（该数据集记为 cap23），试基于 cap23 数据进行完全连结方式的聚合聚类分析:

```
temperature=read.csv("http://staff.ustc.edu.cn/~ynyang/vector/data/temperature.csv",
row.names=1,sep=",")
cap23=temperature[1:23,1:12] #前23行、前12列为23个首都的月温度数据
cap=rownames(cap23)
cap=substr(cap,1,2) #城市名字太长，截取前两个字母
rownames(cap23)=cap
area.true=temperature[1:23,"Area"] # 首都所在的区域
area.true=substr(area.true,1,1)
d=dist(cap23)
mycluster =hclust(d, method="complete")
plot(mycluster )
clusters = cutree(mycluster,k=4) #聚集为k=4类，每个城市所属的类别的标号
> clusters
Am At Be Br Bu Co Du He Ki Kr Li Lo Ma Mi Mo Os Pa Pr Re Ro Sa So St
1  2  3  1  3  3  1  4  3  3  2  1  2  4  4  4  1  3  4  2  3  3  4
#Am,Br, Du,..属于第一类,等等。得到的类别是否与首都所在的区域(area.true)一致?
#注意，考察聚类结果与真实结果是否一致颇具挑战性，比如我们不知道聚类分析得到的
第一类(Am,Br,Du)对应于哪个真实类
tab=table(area.true, clusters)
```

3.2 K-均值聚类

K-means 聚类法是一种广为流行的聚类方法。假设 $\mathbf{x}_1,...,\mathbf{x}_n \in R^p$, 我们要把这 n 个数据点聚为 K 各类。k-means 方法求解 K 个类的中心 $\mathbf{m}_1,...,\mathbf{m}_K \in R^p$ (以 $\mathbf{m}_k$ 为中心的类记为 C_k), 并把每个数据点分配到距离其最近的中心所对应的类中，目标是使得类内平方和达到最小：

$$SS_W = \sum_{k=1}^{K}\sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{m}_k\|^2 1_{(\|\mathbf{x}_i-\mathbf{m}_k\| \le \|\mathbf{x}_i-\mathbf{m}_j\|, j=1,...,k)}$$

该优化问题的经典解法是如下递归迭代算法（Lloyd，1957）：

---

1. 初始化各类中心: 比如随机取数据矩阵的 K 行作为 $\mathbf{m}_k,...,\mathbf{m}_K$ 的初值；
2. 对 $i=1,2,...,n$, 若 $\|\mathbf{x}_i-\mathbf{m}_k\| \le \|\mathbf{x}_i-\mathbf{m}_j\|, j\neq k, j=1,...,K$, 则判定 $\mathbf{x}_i \in C_k$ (即将 $\mathbf{x}_i$ 划分到第 k 类)；
3. 求每一类内的平均值 $\bar{\mathbf{x}}_k=\sum_{i\in C_k}\mathbf{x}_i/|C_k|, k=1,...,K$, 并更新各类的中心：
$$\mathbf{m}_k=\bar{\mathbf{x}}_k, k=1,...,K.$$
4. 回到 2, 迭代，直到 $\mathbf{m}_k, k=1,...,K$ 稳定为止。

K-均值聚类的 R 函数为 kmeans:

```
myclust = kmeans(x, centers, nstart) #x: data;
```

其中

- x 为数据矩阵；
- centers 可以指定为类的个数 k, 也可以是 k 个类的中心初值。如果指定类的个数 k，则 k 个类的中心随机选取 x 的 k 行作为中心的初值；
- nstart 为 kmeans 运行次数。因为中心初值选定不同可能会导致不同的结果，所以通常反复运行 kmeans 多次 (nstart)，每次的中心初值都是随机选定的，比较这些结果的目标函数（组内平方和），最小的作为最终结果。

例 **3**（续）欧洲城市温度数据的 k-means 聚类：

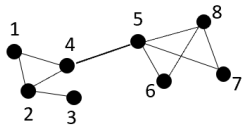
```
temp =read.csv("http://staff.ustc.edu.cn/~ynyang/vector/data/temperature.csv",row.names=1,sep=",")
myclust = kmeans(x=temp[,1:12], centers=4, nstart=50) #4类,重复50次
myclust$centers # 各类的中心
cl = myclust$cluster #各个样本点所属的类别 （1-4）
location=temp[ ,16:15]
plot(location, type="n" )
text(location,rownames(temp ), col=cl ) #四种颜色代表聚成的4类
area.true=temp[,"Area"]
table(area.true,cl) # 错误率是多少?
```

3.3 K-medoid 聚类

K-medoid 聚类分析可以使用 R 软件包 cluster 中的 pam 函数（pam: partition around medoids）。

```
myclust = pam(x, k) #x: data frame or distant matrix;
       #k:number of clusters
```

例 **4**. 下图有边 (edge) 相连的节点认为是相似的



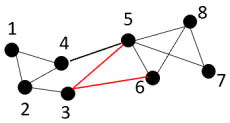
该图的邻接矩阵为

$$A=\begin{pmatrix} 0&1&0&1&0&0&0&0\\ 1&0&1&1&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 1&1&0&0&1&0&0&0\\ 0&0&0&1&0&1&1&1\\ 0&0&0&0&1&0&0&1\\ 0&0&0&0&1&0&0&1\\ 0&0&0&0&1&1&1&0 \end{pmatrix}$$

该矩阵的 (i,j) 元若为 1，表示 i 与 j 是相连的；若为 0，表示它们不连结。这里我们把相连的物体认为是相似的，相似度为 1；不相连的物体的相似度为 0. 我们把 $D=\mathbf{1}\mathbf{1}^\top-I_8-A$ 当作相异度/距离矩阵 (把 A 元素 0-1 互换，对角线换成 0)，基于 D 矩阵，应用 k-medoids 方法将 8 个节点聚集为两类:

```
install.packages("cluster")
library(cluster)
a=c(0,1,0,1,0,0,0,0,  1,0,1,1,0,0,0,0,  0,1,0,0,0,0,0,0,
1,1,0,0,1,0,0,0, 0,0,0,1,0,1,1,1,  0,0,0,0,1,0,0,1,
0,0,0,0,1,0,0,1, 0,0,0,0,1,1,1,0)
A=matrix(a,8,8)
rownames(A)=colnames(A)=1:8
d=1-A-diag(8)
mypam = pam(d, k=2)
```

练习. **例** 4 **中增加两条红色的边**:



试应用 k-medoid **方法将节点聚集为** 2 **类。**

3.4 谱聚类

假设 n 个物体的相似度矩阵为 A，$a_{ij}\geq 0$, 对角元 a_{ii} 不要求一定为 0。定义度数矩阵

$$D=diag(\mathbf{d}),\mathbf{d}=A\mathbb{1},$$

拉普拉斯 (Laplacian) 矩阵定义为

$$L=D-A$$

如果需要将 n 个物体聚集为 K 类，那么我们只需计算 L 的最小 K 个特征根对应的特征向量

$$V=(\mathbf{v}_1,....,\mathbf{v}_K)$$

并对该矩阵应用 Kmean 是方法（或其他聚类方法）进行聚类。我们可自行定义谱聚类函数 sc 如下

```
sc=function(A,k =2) #A:similarity matrix, k: number of clusters
{
        A=as.matrix(A)
        name=rownames(A)
        n=nrow(A)
        d=apply(A,1,sum) #degrees
        D=diag(d)
        L=D-A
        tmp= eigen(L)
        V=tmp$vectors #eigenvectors
        V=V[,(n-K+1):n] #V 的最后 K 列

        tmp=kmeans(V, centers=k)
        cluster=tmp$cluster
        names(cluster)=name
        return(cluster)
} #end
```

```
#对欧洲语言数据进行谱聚类:
S=read.table("http://staff.ustc.edu.cn/~ynyang/vector/data/Euro-language1.txt",head=T)
S=as.matrix(S)

sc(S, k=4)
Finnish Danish English German Italian Norwegian French
4 2 2 2 3 2 3
Hungarian Polish Dutch Spain
1 3 2 3
```

4 分类判别 (classification)

4.1 线性判别和二次判别

Fisher 线性判别分析和二次判别分析的 R 函数为 lda, qda (in package MASS)。

```
myclassifer = lda(class~variables,data=...)
myclassifer = qda(class~variables,data=...)
predict(classifer,newdata)
```

例 **5.** (阿拉斯加和加拿大三文鱼的判别分类，课本 Example 11.7) 三文鱼出生于淡水河流、中，出生 1-2 年后会游到海里生活，若干年再次回到淡水区域。为了判定一条三文鱼是来自美国阿拉斯加还是加拿大，研究人员收集了 50 条阿拉斯加、50 条加拿大三文鱼，测量了其在淡水中第一年的生长速度（Freshwater）和海水中第一年的生长速度（Marine）。生长速度以年轮直径代表。基于这两种鱼的两种生长速度，Fisher 判别法可以计算出线性判别准则函数，即上图的直线。对任意一条不知道其来与地区的三文鱼，我们可以利用该准则预测它是阿拉斯加的还是加拿大的。

```
> library(MASS)
> salmon = read.table("http://staff.ustc.edu.cn/~ynyang/ma18/databook/T11-2.DAT")
```

```
> dimnames(salmon)[[2]]=c("Country", "Gender","Freshwater","Marine")
# Country: 1: Alaska, 2: Canada   # gender: 1:female, 2:male
> salmon =salmon[,-2]  # 不用性别
> plot(salmon[,c("Freshwater","Marine")], col=salmon[,"Country"] ) # alaska: black, canada: red
> salmon.lda = lda(Country~Freshwater+Marine , data=salmon)
> (predict(salmon.lda)$ class->cl)  # 对 100 条鱼利用判别准则分类:
> table(real=salmon[,1], predicted=cl) # 与真实的类别进行比较:
predicted
real  1  2
1 44  6
2  1 49

# 6 个 1 错判为 2，1 个 2 错判为 1，总错判率 (6+1)/100=0.07，正确率 =0.93
# 注意：正确率 0.93 是对训练数据分类的结果，这并不一定说明我们得到的判别函数表现良好，
判别函数的好坏需要在测试数据上判定

# 假设捕获到一条三文鱼，不知其来源，测得淡水、海水年轮分别是 100，200，预测:
> data.frame(Freshwater=100,Marine=200) ->newd
> predict(salmon.lda, newdata=newd)
$`class`
[1] 2
# 预测为第二类，即 Canada

##qda: quadratic discriminant analysis, qda() 的调用方式与 lda() 相同
> salmon.qda = qda(Country~Freshwater+Marine, data=salmon )
> predict(salmon.qda)$class->cl
> table(real=salmon[,1], predicted=cl)
predicted
real  1  2
1 45  5
2  2 48
```

上面得到的 93% 的准确率可能偏高，这是因为我们求解判别准则和预测用的是同一批数据。评价判别准则的效果应该使用求解准则时没有用到的新数据。为此，我们随机地抽取部分数据用于训练求解判别函数，我们假装不知道剩余的数据的真实类别，而是利用得到的准则对剩余的数据（测试数据）进行预测. 然后，因为实际上我们知道测试数据的真实类别，我们可以利用真实类别对预测效果进行评价。

```
n=50     # size of training data set
N=1000    # number of resampling
acc.lda=acc.qda=NULL

for (i in 1:N){
 subset = sample(1:100,n) # 随机抽取 n 个样本组成训练集:
  training.data = salmon[subset,] # 训练数据集
  # 训练:
 tr.lda=lda(Country~Freshwater+Marine, data=training.data) # 求 lda
tr.qda=qda(Country~Freshwater+Marine, data=training.data )#lqa
# 测试数据集
 testing.data=salmon[ -subset, -1] # 测试数据集，但不利用第一列 Country 的信息
salmon[ -subset,1]->class.true  # 测试数据的真实类别
# 预测:
class.pred.lda = predict(tr.lda,newdata= testing.data)$class # 预测测试数据集的类别（Country）
class.pred.qda = predict(tr.qda,newdata= testing.data)$class
```

```
tab1 =  table(class.pred.lda, class.true)
acc1=(tab1[1,1]+tab1[2,2])/(100-n) #lda 的准确率
acc.lda =c(acc.lda, acc1)

tab2 = table(class.pred.qda, class.true)
acc2=(tab2[1,1]+tab2[2,2])/(100-n) #qda 的准确率
acc.qda =c(acc.qda,acc2)
 }
mean(acc.lda)   #lda 的准确率
mean(acc.qda)   #qda 的准确率
```

下面简单介绍 logistic 回归，支持向量机（SVM），回归决策树 (CART) 及相关的随机森林 (Random Forest)，神经网络 (ANN) 等主要分类方法。各种方法的调用方式基本相同，都是如下形式：

```
myclassifier = classfunction(class~feature1+...,data)
predict(myclassifier, newdata)
```

首先安装软件包 e1071 (支持向量机 svm), rpart (分类回归树 CART,tree), randomForest （随机森林）, nnet （神经网络）：

```
install.packages(c("e1071","rpart","randomForest","nnet"))
```

4.2 logistic 回归

logistic 回归分类效果与 LDA 类似，但更简单执行。在广义线性模型函数 glm 中指定 family=binomial 即为 logistic 回归：

```
mylog=glm(Country~Marine+Freshwater,data=training.data,family=binomial)
predict(mylog,newdata=testing.data)
```

4.3 支持向量机 (SVM: support vector machine)

与 LDA 类似，SVM 是线性判别方法，与 LDA 不同在于判别函数由边界点 (support vector 支持向量) 决定而不是由各类的中心决定，另外，SVM 使用核函数，可用于非线性分类。R 包 e1071 中的函数 svm(), 或 R 包 kernlab 中的 ksvm()，分别执行普通的 svm 分类和带有核技巧的 svm 分类。

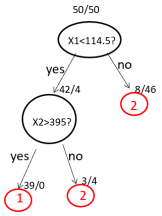
```
>library(e1071) #svm() : svm
>library(kernlab) #ksvm(): kernel svm
subset=c(1:30,51:50)
training.data=salmon[subset,]
testing.data=salmon[ -subset,-1]
class.true=salmon[ -subset,1]

>svm(factor(Country) ~ Marine +Freshwater, data=training.data)->mysvm
>predict(mysvm, newdata=testing.data )
```

4.4 回归分类树（CART: Classification and Regression Trees）/随机森林

回归分类树或回归决策树是一种将特征 (feature, 自变量) 逐个二分的分类 (以及回归) 方法, 其分类判别方法由一系列逻辑判断构成，呈树状结构，比如

x1=Freshwater, x2=Marine
若x1>114.5，判为第二类(Alaska)；
若x1<114.5，若x2<395判为第一类,否则判为第二类；



因为方法的简洁和直观，树方法 CART 在商业领域应用广泛，
随机森林 (random forest) 是通过反复抽样生成多个树（称为森林）分类方法，是 CART 的加强版本。

```
>library(rpart)
>mytree <- rpart(Country~Freshwater+Marine, data=training.data,
method="class" )
>predict(mytree,newdata=testing.data)
# 随机森林:
> myfit=randomForest(Country~Marine+Freshwater,data=training.data, ntree=1000) #1000 棵树
> predict(myfit, newdata=testing.data )
```

4.5 深度学习

安装深度学习的 R 包 keras 可能花费比较长的时间，代码如下：

```
install.packages("keras")
library(keras)
install_keras() # 安装有关的软件，比如 tensorflow
```

细节参看https://zhuanlan.zhihu.com/p/29261816 或 http://keras.rstudio.com

4.6 练习

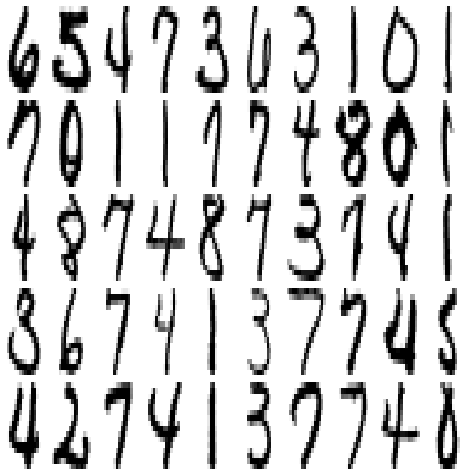
数据集 zip.train (http://staff.ustc.edu.cn/~ynyang/vector/data/zip.train) 收集了 7291 个信封上的邮政编码手写数字 (0-9), 每个数字是 16×16 像素的灰度值，按列排成长度为 256 的向量。数据集 zip.train 是 7291× 257 的矩阵，行代表每个数字，第一列是数字 (0-9), 第 2 列至 257 列是手写体图像的灰度值向量（已经经过了对其，归一化等处理）。首先读入数据，并画出前 50 行的数字:

```
> train=read.table("http://staff.ustc.edu.cn/~ynyang/ma17/data/zip.train" )

>  train[1:50,-1]->digit50 # plot 前 50 个数字:
>  digit50=as.matrix(digit50)

>  par(mfrow=c(5,10),mai=rep(0.06, 4)) #mai: 改变图框 4 个周边的空白大小

for (i in 1:50){
        digit50[i,]-> img
        matrix(img, 16,16  )->img
        img[1:16,16:1]->img
        image(1-img,col=gray((0:32)/32),axes=F)
}
```



1. 对 zip.train 数据应用任或所有何前述方法进行判别分析（称 zip.train 为训练数据集）。

2. 一个分类方法在训练数据集上表现良好（比如具有较高分类准确率或较低的错分率）并不一定表明它在新的数据上的预测分类效果良好，所以为了评判一种分类方法的预测精度，通常需要将训练得到的方法应用于新的数据集（测试数据集）上进行评估。为此在分类预测问题研究中，通常我们需要预留一部分数据作为测试集，在训练集上训练得到分类方法后预测测试集的各个样本的类别，并与它们实际的真实类别对比，计算预测错误率或准确率。本问题我们预留了一部分数据 zip.test (`http://staff.ustc.edu.cn/~ynyang/vector/data/zip.test`) 该数据集与 zip.train 格式相同但行数只有 2007 行（2007 个手写数字）。第一列为数字/类别，其它列为像素灰度值。读入该数据，并将 (a) 中得到的 LDA 判别分类用于预测 zip.test 的 2007 个数字，与第一列的真实数字比较即可得到预测的错误率或准确率。