!MasterIA_GuionPracticas_Outliers_D2_ClusterBasedOutliers.R

laurarodrigueznavas

2020-07-16

```
# Máster -> Detección de anomalías
# Laura Rodriguez Navas

source("!Outliers_A3_Funciones_a_cargar_en_cada_sesion.R")

#############################################################################
# MULTIVARIATE STATISTICAL OUTLIERS. CLUSTERING OUTLIERS
#############################################################################

# Los outliers son respecto a un conjunto de variables.


#####################################################################
# Lectura de valores y Preprocesamiento
#####################################################################

# Trabajamos sobre las columnas numéricas de iris [1:4]
# Este conjunto de datos está disponible en R
# Tanto LOF como clustering usan distancias entre registros, por lo que habrá
# que trabajar sobre los datos previamente normalizados

# Construimos los siguiente conjuntos:

# mis.datos.numericos -> con las columnas 1:4 de iris
# mis.datos.numericos.normalizados -> con los valores normalizados
# a Los rownames de mis.datos.numericos.normalizados les asignamos los rownames de mis.datos.numericos

# Establecemos la variable numero.de.outliers a 5 y numero.de.clusters a 3


mis.datos.numericos   = iris[,1:4]
# mis.datos.numericos   = mis.datos.originales[,sapply(mis.datos.originales, is.numeric)]
mis.datos.numericos.normalizados           = scale(mis.datos.numericos)
rownames(mis.datos.numericos.normalizados) = rownames(mis.datos.numericos)

numero.de.outliers   = 5
numero.de.clusters   = 3

set.seed(2)  # Para establecer la semilla para la primera iteración de kmeans


#############################################################################
```

```
# Cómputo de los outliers según la distancia euclídea de cada dato
# al centroide de su cluster
# El centroide podrá ser cualquiera (podrá provenir de un k-means
# o ser un medoide, por ejemplo)
#############################################################################


#############################################################################
# k-Means

# Construimos el modelo kmeans (modelo.kmeans) con los datos normalizados.
# Para ello, usamos la función de R llamada "kmeans"

# A partir del resultado de kmeans, accedemos a:

# a) $cluster para obtener
#   los índices de asignación de cada dato al cluster correspondiente
#   El resultado lo guardamos en la variable indices.clustering.iris
#   Por ejemplo, si el dato con índice 69 está asignado al tercer cluster,
#   en el vector indices.clustering.iris habrá un 3 en la componente número 69

# b) $centers para obtener los datos de los centroides.
#   Los datos están normalizados por lo que los centroides también lo están.
#   El resultado lo guardamos en la variable centroides.normalizados.iris


# indices.clustering.iris
# 1   2   3   4   ... 69  70  71 ...
# 1   1   1   1   ... 3   3   2  ...

# centroides.normalizados.iris
#    Sepal.Length Sepal.Width Petal.Length Petal.Width
# 1  -1.01119138  0.85041372   -1.3006301  -1.2507035
# 2   1.13217737  0.08812645    0.9928284   1.0141287
# 3  -0.05005221 -0.88042696    0.3465767   0.2805873


# COMPLETAR

modelo.kmeans = kmeans(mis.datos.numericos.normalizados, numero.de.clusters)
indices.clustering.iris = modelo.kmeans$cluster
centroides.normalizados.iris = modelo.kmeans$centers

indices.clustering.iris
```

```
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
##   2   2   2   2   2   2   2   2   2   2   3   3   3   1   1   1   3   1   1   1
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
##   1   1   1   1   1   3   1   1   1   1   3   1   1   1   1   3   3   3   1   1
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

```
##   1   1   1   1   1   3   3   1   1   1   1   1   1   1   1   1   1   1   1   1
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
##   3   1   3   3   3   3   1   3   3   3   3   3   3   1   1   3   3   3   3   1
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
##   3   1   3   1   3   3   1   3   3   3   3   3   3   1   1   3   3   3   1   3
## 141 142 143 144 145 146 147 148 149 150
##   3   3   1   3   3   3   1   3   3   1
```

```
centroides.normalizados.iris
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1  -0.05005221 -0.88042696    0.3465767   0.2805873
## 2  -1.01119138  0.85041372   -1.3006301  -1.2507035
## 3   1.13217737  0.08812645    0.9928284   1.0141287
```

```
# Calculamos la distancia euclídea de cada dato a su centroide (con los valores normalizados)
# Para ello, usad la siguiente función:

distancias_a_centroides = function (datos.normalizados,
                                    indices.asignacion.clustering,
                                    datos.centroides.normalizados){

  sqrt(rowSums(   (datos.normalizados - datos.centroides.normalizados[indices.asignacion.clustering,])^
}

# dist.centroides.iris
# 1          2          3             ......
# 0.21224719 0.99271979 0.64980753    ......

# Ordenamos dichas distancias a través de la función order y obtenemos
# los índices correspondientes. Nos quedamos con los primeros
# (tantos como diga la variable numero.de.outliers)

# top.outliers.iris
# [1]  42  16 132 118  61



# COMPLETAR

dist.centroides.iris = distancias_a_centroides(mis.datos.numericos.normalizados,
                    indices.clustering.iris, centroides.normalizados.iris)
dist.centroides.iris
```

```
##          1          2          3          4          5          6          7
## 0.21224719 0.99271979 0.64980753 0.90043954 0.40081182 1.20750269 0.50077168
##          8          9         10         11         12         13         14
## 0.09101975 1.41699943 0.78729327 0.78735613 0.27525909 1.03152318 1.33036767
##         15         16         17         18         19         20         21
## 1.63318426 2.39097792 1.20345649 0.21546645 1.20582692 0.86416678 0.50233278
##         22         23         24         25         26         27         28
## 0.66603310 0.68428828 0.47785541 0.36224071 0.98693280 0.22607352 0.29373456
##         29         30         31         32         33         34         35
## 0.25276420 0.64802729 0.79870753 0.52134601 1.57132248 1.87025929 0.76601866
##         36         37         38         39         40         41         42
## 0.54713039 0.62868698 0.45829036 1.22957840 0.14532143 0.20194091 2.66163878
```

```
##         43         44         45         46         47         48         49
## 0.90623502 0.49913793 0.91852203 1.01605653 0.86663433 0.72034591 0.72082821
##         50         51         52         53         54         55         56
## 0.30194189 0.94969901 0.99020420 0.72419115 0.96617285 0.90820971 0.35602646
##         57         58         59         60         61         62         63
## 1.00314223 1.49711767 1.11259349 0.77661127 1.96536543 0.77271645 1.25521261
##         64         65         66         67         68         69         70
## 0.65934255 0.73586930 0.98451145 0.79917371 0.56648821 1.19587266 0.67654881
##         71         72         73         74         75         76         77
## 1.18403557 0.52883896 0.78969753 0.57179658 0.90287314 1.02741732 1.08939645
##         78         79         80         81         82         83         84
## 0.51876463 0.58777761 0.76176891 0.89438614 0.98220802 0.39082749 0.54042536
##         85         86         87         88         89         90         91
## 0.90265419 1.39386507 0.77749506 1.05774967 0.81590025 0.59787905 0.48984029
##         92         93         94         95         96         97         98
## 0.84127588 0.38811327 1.52759268 0.30728694 0.81453376 0.56239619 0.72429625
##         99        100        101        102        103        104        105
## 1.28375778 0.38036372 1.05492381 0.76330053 0.52296801 0.76978600 0.52854495
##        106        107        108        109        110        111        112
## 1.19820311 1.22141419 0.91967832 1.39845666 1.47828134 0.47715060 1.02915085
##        113        114        115        116        117        118        119
## 0.27726063 0.94542082 1.38967803 0.68393590 0.46307922 2.09425567 1.82481066
##        120        121        122        123        124        125        126
## 1.17397090 0.52364782 0.90906516 1.46551332 0.84452547 0.51835646 0.66437594
##        127        128        129        130        131        132        133
## 0.79507685 0.94608205 0.83817520 0.75495281 1.06767181 2.16620031 0.87389558
##        134        135        136        137        138        139        140
## 0.79374139 0.80252234 1.25562227 1.06856050 0.51262657 0.96693203 0.22973182
##        141        142        143        144        145        146        147
## 0.57181922 0.50960181 0.76330053 0.53961093 0.84844265 0.52300623 1.02749641
##        148        149        150
## 0.44223037 1.08075040 1.00173079
```

```r
dist.centroides.iris.ordered = order(dist.centroides.iris, decreasing = TRUE)
top.outliers.iris = dist.centroides.iris.ordered[1:numero.de.outliers]

top.outliers.iris
```

```
## [1]  42  16 132 118  61
```