

Detección de anomalías

Laura del Pino Díaz

4/2/2017

Índice

Personas en el gimnasio del campus	2
Detección de anomalías	3
Paquetes necesarios	3
Funciones básicas	4
Detección de anomalías univariante	8

Personas en el gimnasio del campus

La base de datos que vamos a usar es *Crowdeness at campus gym* que traducido al español es la población que hay dentro de un gimnasio universitario.

Los atributos que tenemos son los siguientes:

- timestamp (entero): el número de segundos desde el comienzo del día.
- day_of_week (entero): día de la semana del 0 al 6.
- is_weekend (entero): si el día pertenece a un fin de semana.
- is_holiday (entero): si el día es un día de vacaciones.
- apparent_temperature(real): temperatura aparente en el momento en el que se realiza la observación, en grados Fahrenheit.
- temperature (real): temperatura medida.
- is_start_of_semester (entero): indica si es el comienzo del semestre.

Como salida se tiene el siguiente atributo: * Number_of_people (entero)

El objetivo sobre esta base de datos es detectar si existen anomalías en el número de personas que están en el gimnasio.

```
datos = read.csv("./data.csv", header = TRUE)
```

Base de datos obtenida de Kaggle

Detección de anomalías

Paquetes necesarios

Son varios los paquetes de detección de anomalías que podemos encontrar disponibles para R, pero los que vamos a utilizar se listan a continuación.

- Librerías de gráficos

```
library(ggplot2)
library(devtools)
library(reshape)
library(ggbiplot)
library(rgl)
library(GGally)
```

- Librerías de detección de anomalías para 1-variantes

```
library(outliers)
library(EnvStats)
```

- Librerías de detección de anomalías para multi-variantes

```
library(mvoutlier)
library(CerioliOutlierDetection)
library(robustbase)
library(mvnormtest)
library(MASS)
```

- Librerías de detección de anomalías con métodos no supervisados

```
library(DMwR)
library(cluster)
```

Funciones básicas

Además de las funciones contenidas en los paquetes anteriormente mencionados necesitaremos las siguientes funciones:

- `MiPlot_Univariate_Outliers`: muestra en un gráfico aquellos puntos que son anomalías. Para ello necesita los datos, un vector lógico que indique quienes son los outlier y el título del gráfico.

```
MiPlot_Univariate_Outliers = function(datos, indices_de_Outliers,
titulo) {
  numero.de.datos = nrow(as.matrix(datos))
  vectorTFoutliers = rep(FALSE, numero.de.datos)
  vectorTFoutliers[indices_de_Outliers] = TRUE
  vector.colores.outlier = rep("black", numero.de.datos)
  vector.colores.outlier[vectorTFoutliers] = "red"

  # cat('\nNúmero de datos: ') cat(numero.de.datos)
  # cat('\n¿Quiénes es outlier?: ') cat(vectorTFoutliers)
  # cat('\n')

  x11()
  plot(datos, col = vector.colores.outlier, main = titulo)
}
```

- `vector_es_outlier_IQR`: devuelve un vector que determina si el valor en la posición i-ésima es una anomalía o no basándose en la distancia intercuartil.

```
vector_es_outlier_IQR = function(datos, indice.de.columna, coef = 1.5) {
  columna.datos = datos[, indice.de.columna]
  cuartil.primerio = quantile(columna.datos)[2] #quantile[1] es el mínimo y quantile[5] el máximo.
  cuartil.tercero = quantile(columna.datos)[4]
  iqr = cuartil.tercero - cuartil.primerio
  extremo.superior.outlier = (iqr * coef) + cuartil.tercero
  extremo.inferior.outlier = cuartil.primerio - (iqr * coef)
  es.outlier = columna.datos > extremo.superior.outlier | columna.datos <
    extremo.inferior.outlier
  return(es.outlier)
}
```

- `Nombres_de_Filas`: devuelve el nombre de las filas

```
Nombres_de_Filas = function(datos, vector_TF_datos_a_incluir) {
  numero.de.filas = nrow(datos)

  if (is.null(row.names(datos)))
    row.names(datos) = rep(1:numero.de.filas)

  nombres.de.filas = rep("", numero.de.filas)
  nombres.de.filas[vector_TF_datos_a_incluir == TRUE] = row.names(datos)[vector_TF_datos_a_incluir ==
    TRUE]
  nombres.de.filas
}
```

- `vector_claves_outliers_IQR`: devuelve un vector con los índices de los outliers.

```
vector_claves_outliers_IQR = function(datos, indice, coef = 1.5) {
  columna.datos = datos[, indice]
  vector.de.outliers = vector_es_outlier_IQR(datos, indice,
    coef)
  which(vector.de.outliers == TRUE)
}
```

- `MiBoxPlot_IQR_Univariate_Outliers`: genera un gráfico de cajas de todas aquellas variables que se le indiquen en *indice.de.columna* . Si se le especifica el valor 3 de coeficiente se determinan aquellas anomalías consideradas extremas.

```
MiBoxPlot_IQR_Univariate_Outliers = function(datos, indice.de.columna,
  coef = 1.5) {

  datos = as.data.frame(datos)
  vector.TF.outliers.IQR = vector_es_outlier_IQR(datos, indice.de.columna,
    coef)
  nombres.de.filas = Nombres_de_Filas(datos, vector.TF.outliers.IQR)
  nombre.de.columna = colnames(datos, indice.de.columna)

  ggboxplot = ggplot(data = datos, aes(x = factor(""), y = datos[,
    indice.de.columna]), environment = environment()) + xlab(nombre.de.columna) +
    ylab("") + geom_boxplot(outlier.colour = "red") + geom_text(aes(label = nombres.de.filas)) #,

  x11()
  ggboxplot
}
```

- `MiBoxPlot_juntos`: muestran en un mismo gráfico todos los boxplot

```
MiBoxPlot_juntos = function(datos, vector_TF_datos_a_incluir = c()) {

  nombres.de.filas = Nombres_de_Filas(datos, vector_TF_datos_a_incluir)

  datos = scale(datos)
  datos.melted = melt(datos)
  colnames(datos.melted)[2] = "Variables"
  colnames(datos.melted)[3] = "zscore"
  factor.melted = colnames(datos.melted)[1]
  columna.factor = as.factor(datos.melted[, factor.melted])
  levels(columna.factor)[!levels(columna.factor) %in% nombres.de.filas] = ""

  ggplot(data = datos.melted, aes(x = Variables, y = zscore),
    environment = environment()) + geom_boxplot(outlier.colour = "red") +
    geom_text(aes(label = columna.factor), size = 3)
}
```

- `MiBoxPlot_juntos_con_etiquetas`: muestra en un solo gráfico todas los boxplot con sus etiquetas.

```
MiBoxPlot_juntos_con_etiquetas = function(datos, coef = 1.5) {
  matriz.datos.TF.outliers = sapply(1:ncol(datos), function(x) vector_es_outlier_IQR(datos,
    x, coef)) # Aplicamos outlier IQR a cada columna
  vector.datos.TF.outliers = apply(matriz.datos.TF.outliers,
    1, sum)
  vector.datos.TF.outliers[vector.datos.TF.outliers > 1] = 1 # Si un registro es outlier en alguna c

  MiBoxPlot_juntos(datos, vector.datos.TF.outliers)
}
```

- `MiPlot_resultados_TestGrubbs`: aplica el test de Grubbs.

```
MiPlot_resultados_TestGrubbs = function(datos) {
  alpha = 0.05

  test.de.Grubbs = grubbs.test(datos, two.sided = TRUE)
  cat("p.value: ")
  cat(test.de.Grubbs$p.value)
  cat("\n")

  if (test.de.Grubbs$p.value < alpha) {
    indice.de.outlier.Grubbs = order(abs(datos - mean(datos)),
      decreasing = T)[1]
    indice.de.outlier.Grubbs
    cat("?ndice de outlier: ")
    cat(indice.de.outlier.Grubbs)
    cat("\n")
    valor.de.outlier.Grubbs = datos[indice.de.outlier.Grubbs]
    cat("Valor del outlier: ")
    cat(valor.de.outlier.Grubbs)
    MiPlot_Univariate_Outliers(datos, "Test de Grubbs", indice.de.outlier.Grubbs)
  } else cat("No hay outliers")
}
```

- `MiPlot_resultados_TestRosner`: realiza el test de Rosner sobre la base de datos.

```
MiPlot_resultados_TestRosner = function(datos) {
  test.de.rosner = rosnerTest(datos, k = 4)
  is.outlier.rosner = test.de.rosner$all.stats$Outlier
  k.mayores.desviaciones.de.la.media = test.de.rosner$all.stats$Obs.Num
  indices.de.outliers.rosner = k.mayores.desviaciones.de.la.media[is.outlier.rosner]
  valores.de.outliers.rosner = datos[indices.de.outliers.rosner]
  cat("\nTest de Rosner")
  cat("\n?ndices de las k-mayores desviaciones de la media: ")
  cat(k.mayores.desviaciones.de.la.media)
  cat("\nDe las k mayores desviaciones, ?Qui?n es outlier? ")
  cat(is.outlier.rosner)
  cat("\nLos ?ndices de los outliers son: ")
  cat(indices.de.outliers.rosner)
  cat("\nLos valores de los outliers son: ")
  cat(valores.de.outliers.rosner)
  MiPlot_Univariate_Outliers(datos, indices.de.outliers.rosner,
    "Test de Rosner")
}
```

- MiBiplot y MiBiPlot_Multivariate_Outliers : reduce las dimensiones de la base de datos y muestra el gráfico biplot.

```
MiBiplot = function(datos) {
  PCA.model = princomp(scale(datos))
  biplot = ggbiplot(PCA.model, obs.scale = 1, var.scale = 1,
    varname.size = 5, alpha = 1/2)
  x11()
  print(biplot)
}

MiBiPlot_Multivariate_Outliers = function(datos, vectorTFoutliers,
  titulo) {
  identificadores_de_datos = rownames(datos)
  identificadores_de_datos[!vectorTFoutliers] = ""
  cat(identificadores_de_datos)

  PCA.model = princomp(scale(datos))
  outlier.shapes = c(".", "x") #c(21,8)
  biplot = ggbiplot(PCA.model, obs.scale = 1, var.scale = 1,
    varname.size = 5, groups = vectorTFoutliers, alpha = 1/2) #alpha = 1/10,
  biplot = biplot + labs(color = "Outliers")
  biplot = biplot + scale_color_manual(values = c("black",
    "red"))
  biplot = biplot + geom_text(label = identificadores_de_datos,
    stat = "identity", size = 3, hjust = 0, vjust = 0)
  biplot = biplot + ggtitle(titulo)

  x11()
  print(biplot)
}
```

- MiBiPlot_Clustering_Outliers: realiza una clusterización y después genera un gráfico biplot.

```
MiBiPlot_Clustering_Outliers = function(datos, titulo) {
  PCA.model = princomp(scale(datos))
  outlier.shapes = c("o", "x") #c(21,8)

  identificadores_de_datos = rownames(datos)
  identificadores_de_datos[!BIPLOT.isOutlier] = ""
  # cat(identificadores_de_datos)

  BIPLOT.asignaciones.clusters = factor(BIPLOT.asignaciones.clusters)

  biplot = ggbiplot(PCA.model, obs.scale = 1, var.scale = 1,
    varname.size = 3, alpha = 0) + geom_point(aes(shape = BIPLOT.isOutlier,
    colour = factor(BIPLOT.asignaciones.clusters))) + scale_color_manual(values = BIPLOT.cluster.co)
  scale_shape_manual(values = outlier.shapes) + ggtitle(titulo) +
  geom_text(label = identificadores_de_datos, stat = "identity",
    size = 3, hjust = 0, vjust = 0)

  x11()
  print(biplot)
}
```

Detección de anomalías univariante

Como primera aproximación a la detección de anomalías, vamos a estudiar cada una de las variables para saber si contienen valores anómalos. Para ello utilizaremos el test de la distancia intercuartil (IQR).

La distancia intercuartil la calculamos como la resta del primer cuartil al primero.

```
vectorIQR = apply(datos, 2, function(x) {
  iqr = quantile(x, 0.75) - quantile(x, 0.25)
})
vectorQuantiles = apply(datos, 2, function(x) {
  quantile(x, 0.25)
})
vectorQuantiles = rbind(vectorQuantiles, apply(datos, 2, function(x) {
  quantile(x, 0.75)
}))
vectorIQR = rbind(vectorIQR, vectorQuantiles)
```

En este momento en el vectorIQR tenemos en la primera línea la distancia intercuartil y en las dos siguientes filas el primer y el tercer cuartil. Con esta información podemos determinar si son outliers y si son extremos outliers.

```
outlierMatrix = matrix(data = FALSE, nrow = (dim(datos)[1]),
  ncol = (dim(datos)[2]))
for (i in 1:(dim(datos)[1])) {
  for (j in 1:(dim(datos)[2])) {
    outlierMatrix[i, j] = datos[i, j] > vectorIQR[1, j] *
      1.5 + vectorIQR[3, j] | datos[i, j] < vectorIQR[2,
        j] - vectorIQR[1, j] * 1.5
  }
}
```

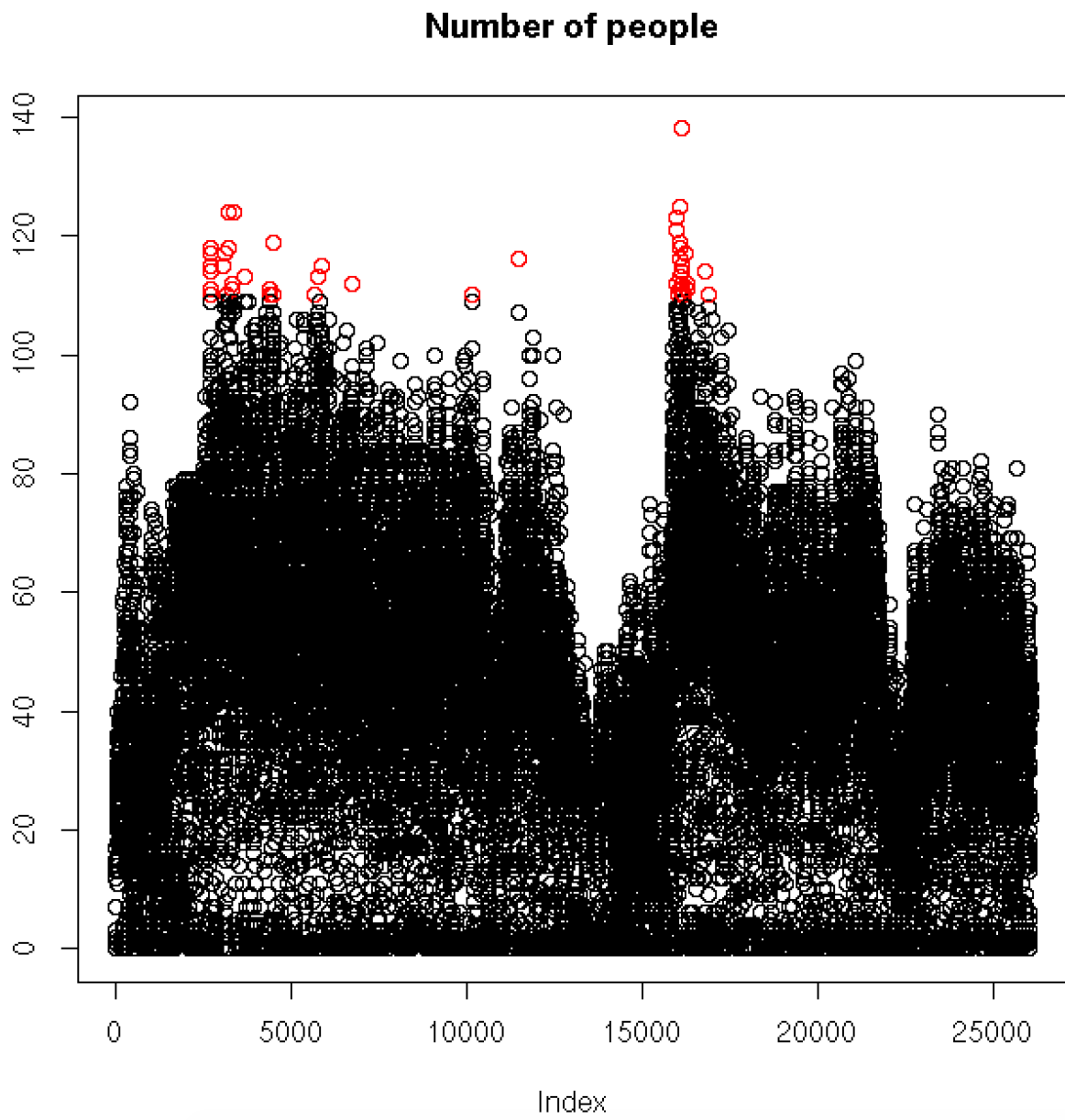
En este momento tenemos en *outlierMatrix* todas aquellas posiciones que son outliers normales, es decir que están a más de 1,5*distancia intercuartil de los cuartiles primero y tercero.

Pero lo cierto es que este método no es aplicable a todas las variables, un ejemplo de ello es la variable *timestamp*, *dayOfWeek*, *isWeekend*, *isHoliday* o *isStartOfSemester* que solo tiene valor descriptivo. Por ello tenemos que eliminar todos aquellos datos que nos sobran y que hemos calculado por limpieza de código.

```
outlierMatrix[, c(2, 3, 4, 5, 8)] = NA
```

Los outlier calculados se muestran a continuación:

```
MiPlot_Univariate_Outliers(datos[, 1], which(outlierMatrix[,
  1] == TRUE), "Number of people")
MiPlot_Univariate_Outliers(datos[, 6], which(outlierMatrix[,
  6] == TRUE), "Apparent Temperature")
MiPlot_Univariate_Outliers(datos[, 7], which(outlierMatrix[,
  7] == TRUE), "Temperature")
```

Apparent Temperature

