Detección de anomalías: Autoencoders y PCA

Joaquín Amat Rodrigo j.amatrodrigo@gmail.com

Febrero, 2020

Tabla de contenidos

Versión PDF: Github

Más sobre ciencia de datos: joaquinamatrodrigo.github.io o cienciadedatos.net

Introducción

La detección de anomalías (*outliers*) con PCA y *Autoencoders* es una estrategia no supervisada para identificar anomalías cuando los datos no están etiquetados, es decir, no se conoce la clasificación real (anomalía - no anomalía) de las observaciones.

Si bien esta estrategia hace uso de PCA o *Autoencoders*, no utiliza directamente su resultado como forma de detectar anomalías, sino que emplea el error de reconstrucción producido al revertir la reducción de dimensionalidad. El error de reconstrucción como estrategia para detectar anomalías se basa en la siguiente idea: los métodos de reducción de dimensionalidad permiten proyectar las observaciones en un espacio de menor dimensión que el espacio original, a la vez que tratan de conservar la mayor información posible. La forma en que consiguen minimizar la perdida global de información es buscando un nuevo espacio en el que la mayoría de observaciones puedan ser bien representadas.

Los métodos de PCA y *Autoencoders* crean una función que mapea la posición que ocupa cada observación en el espacio original con el que ocupa en el nuevo espacio generado. Este mapeo funciona en ambas direcciones, por lo que también se puede ir desde el nuevo espacio al espacio original. Solo aquellas observaciones que hayan sido bien proyectadas podrán volver a la posición que ocupaban en el espacio original con una precisión elevada.

Dado que la búsqueda de ese nuevo espacio ha sido guiada por la mayoría de las observaciones, serán las observaciones más próximas al promedio las que mejor puedan ser proyectadas y en consecuencia mejor reconstruidas. Las observaciones anómalas, por el contrario, serán mal proyectadas y su reconstrucción será peor. Es este error de reconstrucción (elevado al cuadrado) el que puede emplearse para identificar anomalías.

Packages

Los siguientes *packages* se emplean a lo largo del documento.

```
library(R.matlab)   # Lectura de archivos .mat
library(factoextra) # Visualización de PCA
library(h2o)        # Entrenar autoencoders
library(tidyverse)  # Preparación de datos y gráficos
```

Data sets

Los datos empleados en este documento se han obtenido de Outlier Detection DataSets (ODDS), un repositorio con sets de datos comúnmente empleados para comparar la capacidad que tienen diferentes algoritmos a la hora de identificar *outliers*. Shebuti Rayana (2016). ODDS Library [http://odds.cs.stonybrook.edu]. Stony Brook, NY: Stony Brook University, Department of Computer Science.

- **Cardiotocogrpahy dataset** link:
  - Número de observaciones: 1831
  - Número de variables: 21
  - Número de outliers: 176 (9.6%)
  - y: 1 = outliers, 0 = inliers
  - Observaciones: todas las variables están centradas y escaladas (media 0, sd 1).
  - Referencia: *C. C. Aggarwal and S. Sathe, "Theoretical foundations and algorithms for outlier ensembles." ACM SIGKDD Explorations Newsletter, vol.17, no. 1, pp.24–47, 2015. Saket Sathe and Charu C. Aggarwal. LODES: Local Density meets Spectral Outlier Detection. SIAM Conference on Data Mining, 2016.*

- **Speech dataset** link:
  - Número de observaciones: 3686
  - Número de variables: 400
  - Número de outliers: 61 (1.65%)
  - y: 1 = outliers, 0 = inliers
  - Referencia: *Learing Outlier Ensembles: The Best of Both Worlds – Supervised and Unsupervised. Barbora Micenkova, Brian McWilliams, and Ira Assent, KDD ODD2 Workshop, 2014.*

- **Shuttle dataset** link:
  - Número de observaciones: 49097
  - Número de variables: 9
  - Número de outliers: 3511 (7%)
  - y: 1 = outliers, 0 = inliers
  - Referencia: *Abe, Naoki, Bianca Zadrozny, and John Langford. "Outlier detection by active learning." Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2006.*
  -

Todos estos data sets están etiquetados, se conoce si las observaciones son o no anomalías (variable *y*). Aunque los métodos que se describen en el documento son no supervisados, es decir, no hacen uso de la variable respuesta, conocer la verdadera clasificación permite evaluar su capacidad para identificar correctamente las anomalías.

Los datos están disponibles en formato matlab (*.mat*). Para leer su contenido se emplea la función readMat() del paquete R.matlab v3.6.2.

```
cardio_mat  <- readMat("./datos/cardio.mat")
df_cardio   <- as.data.frame(cardio_mat$X)
df_cardio$y <- as.character(cardio_mat$y)

speech_mat  <- readMat("./datos/speech.mat")
df_speech   <- as.data.frame(speech_mat$X)
df_speech$y <- as.character(speech_mat$y)

shuttle_mat  <- readMat("./datos/shuttle.mat")
df_shuttle   <- as.data.frame(shuttle_mat$X)
df_shuttle$y <- as.character(shuttle_mat$y)

datos <- df_cardio
```

PCA

Introducción

Principal Component Analysis (PCA) es un método estadístico que permite simplificar la complejidad de espacios muestrales con muchas dimensiones a la vez que conserva su información. Supóngase que existe una muestra con n individuos cada uno con p variables (X_1, X_2, ..., X_p), es decir, el espacio muestral tiene p dimensiones. PCA permite encontrar un número de factores subyacentes, ($z < p$) que explican aproximadamente lo mismo que las p variables originales. Donde antes se necesitaban p variables para caracterizar a cada individuo, ahora bastan z. Estas nuevas variables se llaman *componentes principales* y tienen la característica de no estar linealmente correlacionadas entre ellas.

Cada componente principal (Z_i) se obtiene por combinación lineal de las variables originales. Se pueden entender como nuevas variables obtenidas al combinar de una determinada forma las variables originales. La primera componente principal de un grupo de variables (X_1, X_2, ..., X_p) es la combinación lineal normalizada de dichas variables que tiene mayor varianza. Una vez calculada la primera componente (Z_1), se calcula la segunda (Z_2) repitiendo el mismo proceso, pero añadiendo la condición de que la combinación lineal no pude estar correlacionada con la primera componente. El proceso se repite de forma iterativa hasta calcular todas las posibles componentes (min(n-1, p)) o hasta que se decida detener el proceso.

La principal limitación que tiene el PCA como método de reducción de dimensionalidad es que solo contempla combinaciones lineales de las variables originales, lo que significa que no es capaz de capturar otro tipo de relaciones.

Para más información sobre PCA consultar Análisis de Componentes Principales (Principal Component Analysis, PCA) y t-SNE

Cálculo de PCA

Con la función prcomp() del paquete stats se pueden calcular las componentes principales de un dataframe o matriz. El objeto prcomp almacena la información de los *eigenvectors*, *eigenvalues* y la matriz de rotación. Con el método predict() se pueden proyectar nuevas observaciones.

```
# Cálculo de PCA
pca <- prcomp(
         x = datos %>% select(-y),
         center = TRUE,
         scale. = TRUE
       )
```

Diagnóstico

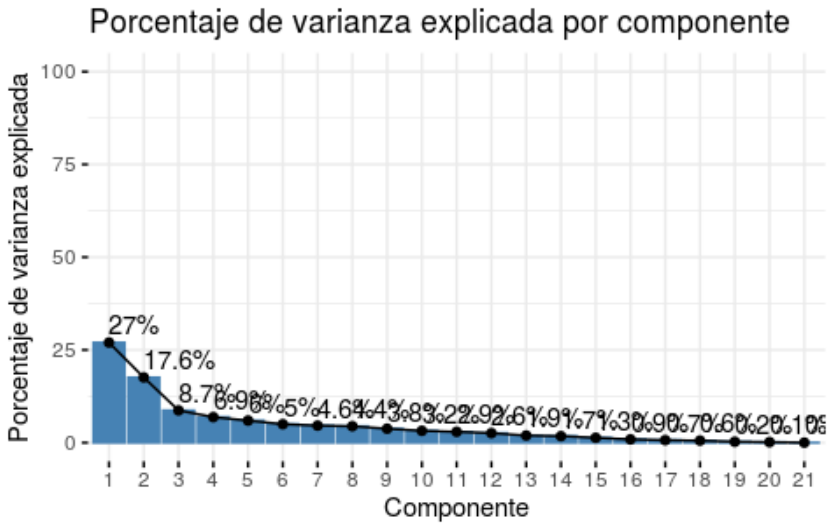
Tras obtener el PCA, se evalúa su capacidad para reducir la dimensionalidad de los datos explorando los *eigenvectors*, *eigenvalues* y la varianza capturada en el proceso.

Información sobre los eigenvalues/varianza

```
get_eig(X = pca)
```

```
##          eigenvalue variance.percent cumulative.variance.percent
## Dim.1  5.666458e+00     2.698313e+01                     26.98313
## Dim.2  3.691532e+00     1.757873e+01                     44.56186
## Dim.3  1.824771e+00     8.689387e+00                     53.25125
## Dim.4  1.445127e+00     6.881558e+00                     60.13280
## Dim.5  1.250148e+00     5.953085e+00                     66.08589
## Dim.6  1.045601e+00     4.979053e+00                     71.06494
## Dim.7  9.637038e-01     4.589066e+00                     75.65401
## Dim.8  9.300013e-01     4.428578e+00                     80.08259
## Dim.9  7.895594e-01     3.759807e+00                     83.84239
## Dim.10 6.709390e-01     3.194947e+00                     87.03734
## Dim.11 6.132349e-01     2.920166e+00                     89.95751
## Dim.12 5.396723e-01     2.569868e+00                     92.52737
## Dim.13 4.035869e-01     1.921842e+00                     94.44922
## Dim.14 3.669830e-01     1.747538e+00                     96.19675
## Dim.15 2.754805e-01     1.311812e+00                     97.50857
## Dim.16 1.861647e-01     8.864985e-01                     98.39506
## Dim.17 1.410317e-01     6.715796e-01                     99.06664
## Dim.18 1.158728e-01     5.517753e-01                     99.61842
## Dim.19 5.117794e-02     2.437045e-01                     99.86212
## Dim.20 2.895402e-02     1.378763e-01                    100.00000
## Dim.21 2.617874e-30     1.246607e-29                    100.00000
```

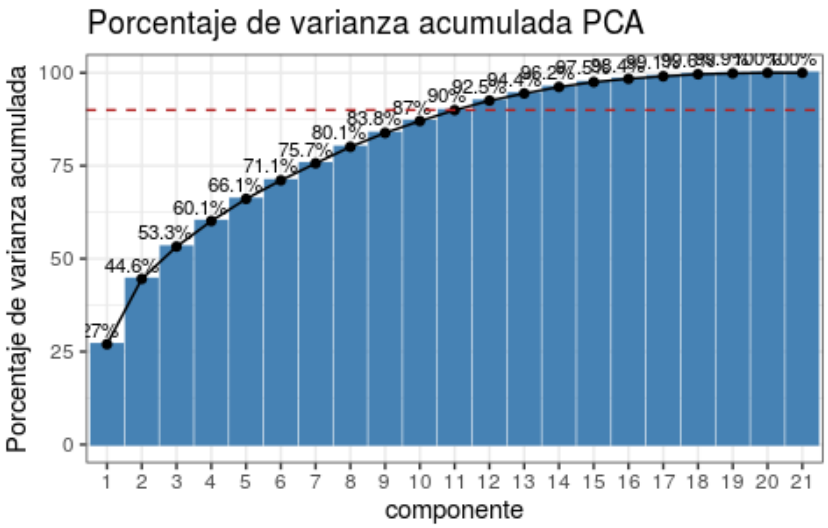
```
fviz_eig(
    X         = pca,
    choice    = "variance",
    addlabels = TRUE,
    ncp       = 21,
    ylim      = c(0, 100),
    main      = "Porcentaje de varianza explicada por componente",
    xlab      = "Componente",
    ylab      = "Porcentaje de varianza explicada",
    labelsize = 12
)
```



Porcentaje de varianza acumulada

Con las primeras 11 componentes se consigue explicar aproximadamente el 90% de la varianza observada en los datos.

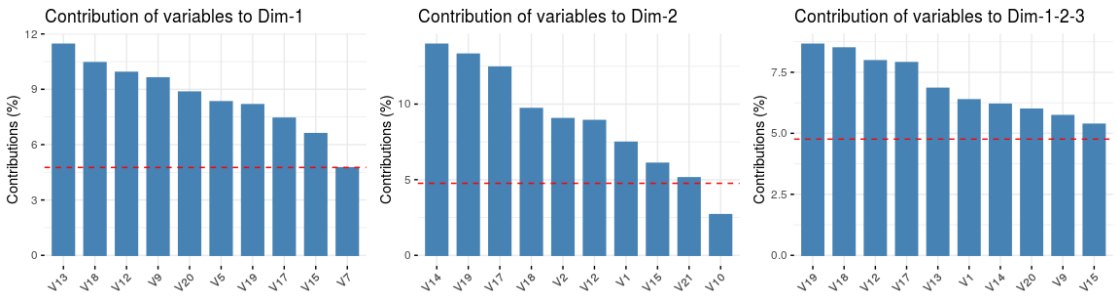
```
ggplot(data = get_eig(X = pca),
       aes(x = as.factor(1:nrow(get_eig(X = pca))),
           y = cumulative.variance.percent,
           group = 1)
       ) +
    geom_col(fill = "steelblue", color = "steelblue") +
    geom_line(color = "black", linetype = "solid") +
    geom_point(shape = 19, color = "black") +
    geom_text(aes(label = paste0(round(cumulative.variance.percent, 1), "%")),
              size = 3, vjust = -0.5, hjust = 0.7) +
    geom_hline(yintercept = 90, color = "firebrick", linetype = "dashed") +
    labs(title = "Porcentaje de varianza acumulada PCA",
         x = "componente",
         y = "Porcentaje de varianza acumulada") +
    theme_bw()
```



Contribución de cada variable a las componentes

La función fviz_contrib() del paquete factoextra permite mostrar de forma gráfica la contribución que tiene cada variable original a una determinada componente o el promedio de contribución a un grupo de componentes.

```
# Contribución a la primera componente.
p1 <- fviz_contrib(X = pca, choice = "var", axes = 1, top = 10)
# Contribución a la segunda componente.
p2 <- fviz_contrib(X = pca, choice = "var", axes = 2, top = 10)
# Contribución conjunta a las 3 primeras componentes.
p3 <- fviz_contrib(X = pca, choice = "var", axes = 1:3, top = 10)

ggpubr::ggarrange(p1, p2, p3, nrow = 1)
```



Reconstrucción

Una vez obtenido el PCA (matriz de *eigenvectors*, proyecciones y medias), la reconstrucción de las observaciones iniciales se puede obtener empelando la siguiente ecuación:

$$\text{reconstrucción} = \text{PC scores} \cdot \text{Eigenvectors}^{\top}$$

Es importante tener en cuenta que, si los datos han sido centrados y escalados (cosa que en términos generales debe hacerse al aplicar PCA) esta transformación debe revertirse.

La siguiente función obtiene la reconstrucción de las observaciones con las que se ha entrenado un objeto `prcomp`, empleando únicamente determinadas componentes.

```
reconstruct_prcomp <- function(pca, comp = NULL){

  # Esta función reconstruye las mismas observaciones con las que se ha creado el
  # PCA empleando únicamente determinadas componentes principales.

  # Parameters
  # ----------
  # pca: "prcomp"
  #   objeto prcomp con los resultados del PCA.
  #
  # comp: "numeric"
  #   componentes principales empleadas en la reconstrucción.
  #
  # Return
  # ----------
  # "matrix" con la reconstrucción de cada observación. Las dimensiones de la
  # matriz son las mismas que las de la matriz o dataframe con el que se estrenó
  # el objeto pca.

  # Si no se especifica comp, se emplean todas las componentes.
  if (is.null(comp)) {
      comp <- seq_along(pca$sdev)
  }
  # Reconstrucción
  recon <- as.matrix(pca$x[, comp]) %*% t(as.matrix(pca$rotation[, comp]))
  # Si se ha aplicado centrado o escalado se revierte la trasformación.
  if (pca$scale[1] != FALSE) {
    recon <- scale(recon , center = FALSE, scale = 1/pca$scale)
  }
  if (pca$center[1] != FALSE) {
    recon <- scale(recon , center = -1*pca$center, scale = FALSE)
  }
 return(recon)
}
```

Se comprueba que, utilizando todas las componentes, la reconstrucción es total. Los valores reconstruidos son iguales a los datos originales (las pequeñas diferencias se deben a imprecisión numérica de las operaciones).

```
# Se aplica la reconstrucción
reconstruccion <- reconstruct_prcomp(pca = pca, comp = NULL)

# Se comparan las 2 primeras variables, originales y reconstruidas, de la primera
# observación del set de datos.
print(as.numeric(reconstruccion[1, 1:2]), digits = 22)
```

```
## [1] 0.0049123146617690363613 38 0.6931907749193891854489 2
```

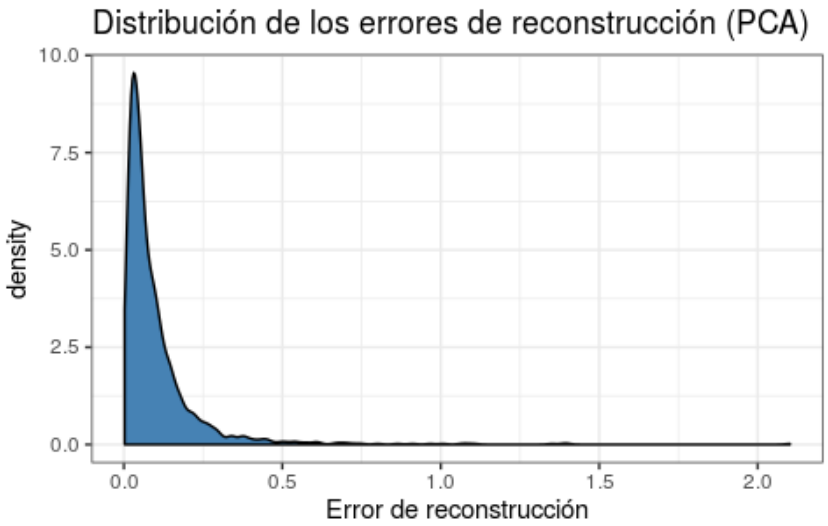
```
print(as.numeric(df_cardio[1, 1:2]), digits = 22)
```

```
## [1] 0.0049123146617683841053 11 0.6931907749193862988690 28
```

Error de reconstrucción

El error cuadrático medio de reconstrucción de una observación se calcula como el promedio de las diferencias al cuadrado entre el valor original de sus variables y el valor reconstruido, es decir, el promedio de los errores de reconstrucción de todas sus variables elevados al cuadrado.

```
# Reconstrucción empleando las 11 primeras componentes (90% de la varianza
explicada)
reconstruccion <- reconstruct_prcomp(pca = pca, comp = 1:11)

# Cálculo del error cuadrático medio de reconstrucción
error_reconstruccion <- reconstruccion - select(datos, -y)
error_reconstruccion <- error_reconstruccion^2
error_reconstruccion <- apply(X = error_reconstruccion, MARGIN = 1, FUN = mean)

# Distribución del error de reconstrucción
tibble(error_reconstruccion) %>%
  ggplot(aes(x = error_reconstruccion)) +
  geom_density(fill = "steelblue") +
  labs(title = "Distribución de los errores de reconstrucción (PCA)",
       x = "Error de reconstrucción") +
  theme_bw()
```

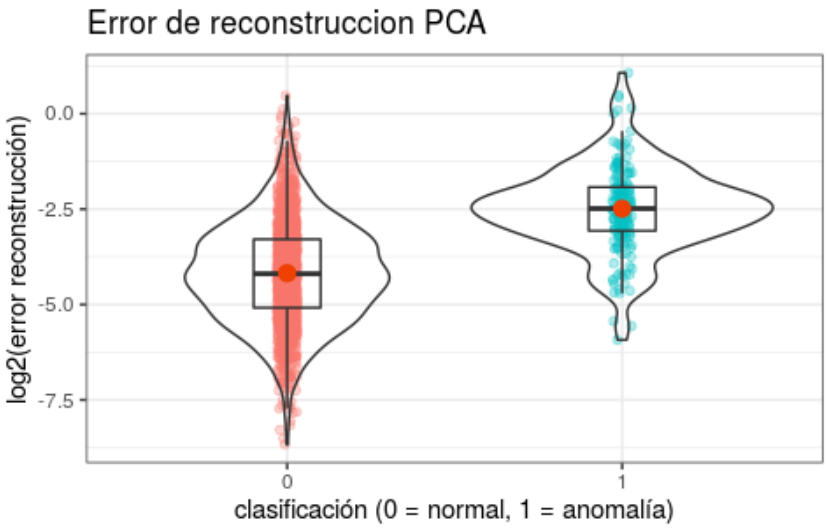


Detección de anomalías

Una vez que el error de reconstrucción ha sido calculado, se puede emplear como criterio para identificar anomalías. Asumiendo que la reducción de dimensionalidad se ha realizado de forma que la mayoría de los datos (los normales) queden bien representados, aquellas observaciones con mayor error de reconstrucción deberían ser las más atípicas.

En la práctica, si se está empleando esta estrategia de detección es porque no se dispone de datos etiquetados, es decir, no se conoce qué observaciones son realmente anomalías. Sin embargo, como en este ejemplo se dispone de la clasificación real, se puede verificar si realmente los datos anómalos tienen errores de reconstrucción más elevados.

```
# Se añade el error de reconstrucción al dataframe original.
datos$error_reconstruccion <- error_reconstruccion

ggplot(data = datos,
       aes(x = y, y = log2(error_reconstruccion))) +
geom_jitter(aes(color = y), width = 0.03, alpha = 0.3) +
geom_violin(alpha = 0) +
geom_boxplot(width = 0.2, outlier.shape = NA, alpha = 0) +
stat_summary(fun.y = "mean", colour = "orangered2", size = 3, geom = "point") +
labs(title = "Error de reconstruccion PCA",
     x = "clasificación (0 = normal, 1 = anomalía)",
     y = "log2(error reconstrucción)") +
theme_bw() +
theme(legend.position = "none")
```



La distribución de los errores de reconstrucción en el grupo de las anomalías (*1*) es claramente superior. Sin embargo, al existir solapamiento, si se clasifican las *n* observaciones con mayor error de reconstrucción como anomalías, se incurriría en errores de falsos positivos.

Acorde a la documentación, el set de datos *Cardiotocogrpahy* contiene 176 anomalías. Véase la matriz de confusión resultante si se clasifican como anomalías las 176 observaciones con mayor error de reconstrucción.

```
resultados <- datos %>%
              select(y, error_reconstruccion) %>%
              arrange(desc(error_reconstruccion)) %>%
              mutate(clasificacion = if_else(row_number() <= 176, "1", "0"))

mat_confusion <- MLmetrics::ConfusionMatrix(
                    y_pred = resultados$clasificacion,
                    y_true = resultados$y
                 )
mat_confusion
```

```
##        y_pred
## y_true    0    1
##      0 1545  110
##      1  110   66
```

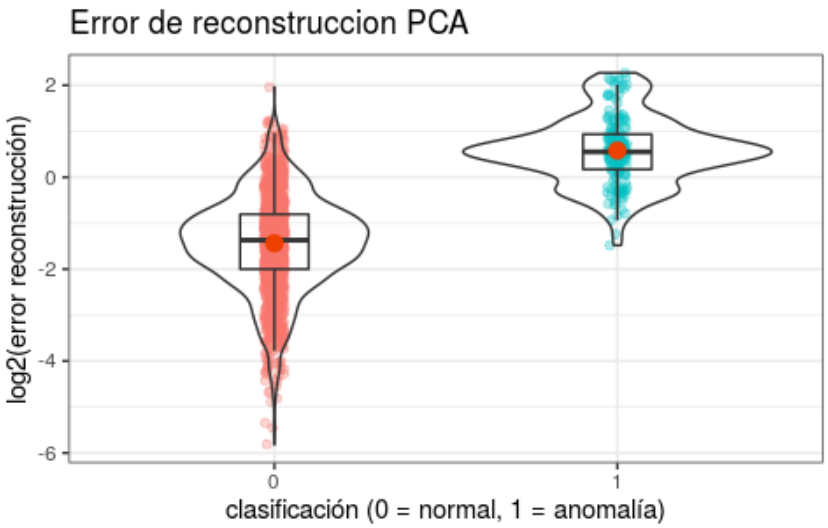
De las 176 observaciones identificadas como anomalías, solo el 38% (66/176) lo son. El porcentaje de falsos positivos (62%) es muy elevado.

Últimas componentes

En el apartado anterior, se han empleado las 11 primeras componentes de un total de 21 ya que con ellas se puede explicar aproximadamente el 90% de la varianza observada. Que esta selección de componentes sea adecuada para la detección de anomalías depende en gran medida de en qué direcciones se desvíen las anomalías. Con frecuencia, los datos anómalos no tienen valores atípicos en las direcciones dominantes pero sí en las poco dominantes, por lo que su comportamiento atípico quedaría recogido en las últimas componentes.

Se repite la detección de anomalías pero esta vez descartando las 4 primeras componentes principales en la reconstrucción.

```
# Reconstrucción empleando todas excepto las 4 primeras componentes
reconstruccion <- reconstruct_prcomp(pca = pca, comp = 5:21)

# Cálculo del error cuadrático medio de recostrucción
error_reconstruccion <- reconstruccion - select(datos, -y)
error_reconstruccion <- error_reconstruccion^2
error_reconstruccion <- apply(X = error_reconstruccion, MARGIN = 1, FUN = mean)

# Se añade el error de reconstrucción al dataframe original.
datos$error_reconstruccion <- error_reconstruccion

ggplot(data = datos,
       aes(x = y, y = log2(error_reconstruccion))) +
geom_jitter(aes(color = y), width = 0.03, alpha = 0.3) +
geom_violin(alpha = 0) +
geom_boxplot(width = 0.2, outlier.shape = NA, alpha = 0) +
stat_summary(fun.y = "mean", colour = "orangered2", size = 3, geom = "point") +
labs(title = "Error de reconstruccion PCA",
     x = "clasificación (0 = normal, 1 = anomalía)",
     y = "log2(error reconstrucción)") +
theme_bw() +
theme(legend.position = "none")
```




```
resultados <- datos %>%
              select(y, error_reconstruccion) %>%
              arrange(desc(error_reconstruccion)) %>%
              mutate(clasificacion = if_else(row_number() <= 176, "1", "0"))

mat_confusion <- MLmetrics::ConfusionMatrix(
                    y_pred = resultados$clasificacion,
                    y_true = resultados$y
                 )
mat_confusion
```

```
##        y_pred
## y_true    0    1
##      0 1600   55
##      1   55  121
```

Descartando las primeras 4 componentes, 121 de las 176 (69%) observaciones identificadas como anomalías lo son realmente. Se ha conseguido reducir el porcentaje de falsos positivos al 31%.

Reentrenamiento iterativo

El PCA anterior se ha obtenido empleando todas las observaciones, incluyendo potenciales anomalías. Dado que el objetivo es generar un espacio de proyección únicamente con los datos "normales", se puede mejorar el resultado reentrenando el PCA pero excluyendo las *n* observaciones con mayor error de reconstrucción (potenciales anomalías).

Se repite la detección de anomalías pero esta vez descartando las observaciones con un error de reconstrucción superior al percentil 70.

```
# Se eliminan las observaciones con un error de reconstrucción superior al
# percentil 70%
cuantil       <- quantile(x = datos$error_reconstruccion, probs = 0.7)
datos_trimmed <- datos %>% filter(error_reconstruccion < cuantil)

# Se elimina la columna error_reconstruccion de ambos dataframes para que no
# participen en el cálculo del PCA
datos$error_reconstruccion <- NULL
datos_trimmed$error_reconstruccion <- NULL

# Tras aplicar el filtrado, la columna V6 tiene varianza cero. Es necesario
# eliminarla para poder aplicar el PCA
datos$V6 <- NULL
datos_trimmed$V6 <- NULL

# PCA únicamente con los datos trimmed.
pca <- prcomp(
         x      = datos_trimmed %>% select(-y),
         center = TRUE,
         scale. = TRUE
      )
```

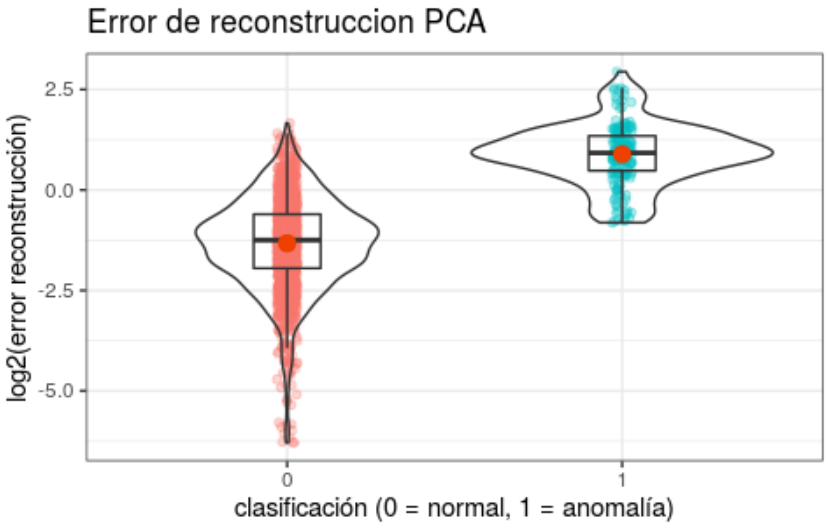
Con este nuevo modelo PCA se recalcula el error de reconstrucción de todas las observaciones. La función error_reconstruccion_prcomp() emplea un PCA ya entrenado para proyectar nuevas observaciones, reconstruirlas y calcular el error.

```
error_reconstruccion_prcomp <- function(pca, new_data, comp=NULL){

  # Esta función calcula el error de reconstrucción de un PCA al proyectar y
  # reconstruir las observaciones empleando únicamente determinadas  componentes
  # principales.
  #
  # Parameters
```

```r
  # ----------
  # pca: "prcomp"
  #   objeto prcomp con los resultados del PCA
  #
  # new_data: "matriz" o "data.frame"
  #  nuevas observaciones
  #
  # comp: "numeric"
  #   componentes principales empleadas en la reconstrucción.
  #
  # Return
  # ----------
  # "numeric" vector con el error de reconstrucción de cada observación.

  # Si no se especifica comp, se emplean todas las componentes
  if (is.null(comp)) {
      comp <- seq_along(pca$sdev)
  }

  # Se seleccionan únicamente las componentes en comp
  pca$rotation <- pca$rotation[, comp]

  proyeciones <- predict(object = pca,
                         newdata = new_data
                  )

  # Recostrucción
  reconstruccion <- as.matrix(proyeciones) %*% t(as.matrix(pca$rotation))

  # Si se ha aplicado centrado o escalado se revierte la trasformación
  if (pca$scale[1] != FALSE) {
    reconstruccion <- scale(reconstruccion , center = FALSE, scale = 1/pca$scale)
  }
  if (pca$center[1] != FALSE) {
    reconstruccion <- scale(reconstruccion , center = -1*pca$center, scale = FALSE)
  }

  # Cálculo del error de reconstrucción
  error_reconstruccion <- reconstruccion - new_data
  error_reconstruccion <- error_reconstruccion^2
  error_reconstruccion <- apply(X = error_reconstruccion, MARGIN = 1, FUN = mean)

  return(error_reconstruccion)
}
```

```
# Se calcula el error de reconstrucción.
error_reconstruccion <- error_reconstruccion_prcomp(
                          pca      = pca,
                          new_data = datos %>% select(-y),
                          comp     = 5:20
                        )
```

```
# Se añade el error de reconstrucción al dataframe original.
datos$error_reconstruccion <- error_reconstruccion

ggplot(data = datos,
       aes(x = y, y = log2(error_reconstruccion))) +
geom_jitter(aes(color = y), width = 0.03, alpha = 0.3) +
geom_violin(alpha = 0) +
geom_boxplot(width = 0.2, outlier.shape = NA, alpha = 0) +
stat_summary(fun.y = "mean", colour = "orangered2", size = 3, geom = "point") +
labs(title = "Error de reconstruccion PCA",
     x = "clasificación (0 = normal, 1 = anomalía)",
     y = "log2(error reconstrucción)") +
theme_bw() +
theme(legend.position = "none")
```




```
resultados <- datos %>%
              select(y, error_reconstruccion) %>%
              arrange(desc(error_reconstruccion)) %>%
              mutate(clasificacion = if_else(row_number() <= 176, "1", "0"))

mat_confusion <- MLmetrics::ConfusionMatrix(
                    y_pred = resultados$clasificacion,
                    y_true = resultados$y
                  )
mat_confusion
```

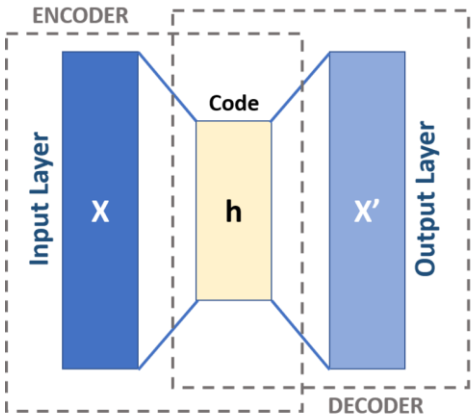
```
##        y_pred
## y_true    0    1
##      0 1605   50
##      1   50  126
```

Tras el reentrenamiento, 126 de las 176 (72%) observaciones identificadas como anomalías lo son realmente. Se ha reducido ligeramente el porcentaje de falsos positivos al 28%.

Autoencoder

Introducción

Los *autoencoders* son un tipo de redes neuronales en las que la entra y salida del modelo es la misma, es decir, redes entrenadas para predecir un resultado igual a los datos de entrada. Para conseguir este tipo de comportamiento, la arquitectura de los *autoencoders* es simétrica, con una región llamada *encoder* y otra *decoder*. ¿Cómo sirve esto para reducir la dimensionalidad? Los *autoencoders* siguen una arquitectura de cuello de botella, la región *encoder* está formada por una o varias capas, cada una con menos neuronas que su capa precedente, obligando así a que la información de entrada se vaya comprimiendo. En la región *decoder* esta compresión se revierte siguiendo la misma estructura pero esta vez de menos a más neuronas.



Arquitectura de un autoencoder básico. Fuente: Wikipedia

Para conseguir que la salida reconstruida sea lo más parecida posible a la entrada, el modelo debe aprender a capturar toda la información posible en la zona intermedia. Una vez entrenado, la salida de la capa central del *autoencoder* (la capa con menos neuronas) es una representación de los datos de entrada pero con una dimensionalidad igual el número de neuronas de esta capa.

La principal ventaja de los *autoencoders* es que no tienen ninguna restricción en cuanto al tipo de relaciones que pueden aprender, por lo tanto, a diferencia del PCA, la reducción de dimensionalidad puede incluir relaciones no lineales. La desventaja es su alto riesgo de sobreentrenamiento (*overfitting*), por lo que se recomienda emplear muy pocas épocas y siempre evaluar la evolución del error con un conjunto de validación.

En el caso de utilizar funciones de activación lineales, las variables generadas en el cuello de botella (la capa con menos neuronas), son muy similares a las componentes principales de un PCA pero sin que necesariamente tengan que ser ortogonales entre ellas.

Entrenamiento autoencoder

El paquete `h2o` permite entrenar, entre otros modelos de machine learning, redes neuronales con arquitectura de *autoencoder*.

```
# Creación de un cluster local.
h2o.init(ip = "localhost",
         # -1 indica que se empleen todos los cores disponibles.
         nthreads = -1,
         # Máxima memoria disponible para el cluster.
         max_mem_size = "4g")
```

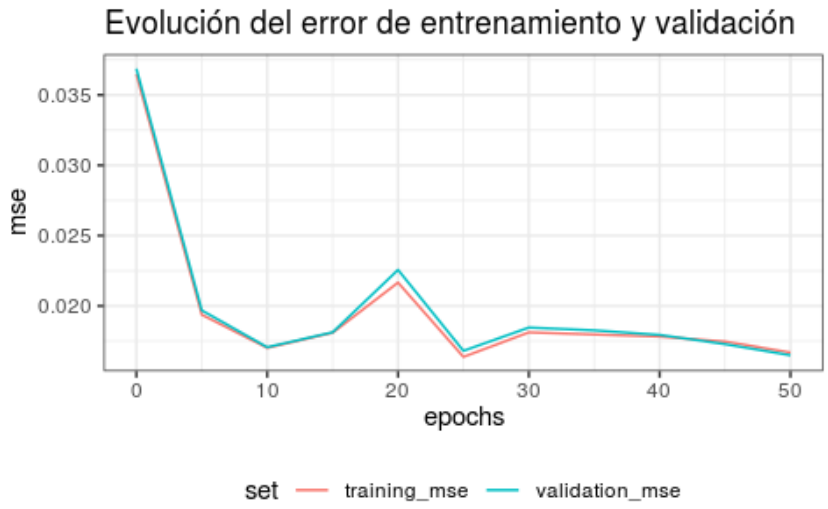
```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         8 hours 26 minutes
##     H2O cluster timezone:       Europe/Madrid
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.28.0.1
##     H2O cluster version age:    2 months and 7 days
##     H2O cluster name:           H2O_started_from_R_ximo_rvk435
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.46 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, XGBoost, Algos, AutoML, Core V3,
TargetEncoder, Core V4
##     R Version:                  R version 3.6.2 (2019-12-12)
```

```
# Se eliminan los datos del cluster por si ya había sido iniciado.
h2o.removeAll()
h2o.no_progress()

# Se transfieren los datos al cluster de h2o.
datos <- df_cardio
datos_h2o <- as.h2o(
                x = datos,
                destination_frame = "datos_h2o"
             )

# División de las observaciones en conjunto de entrenamiento y validación.
datos_h2o_split <- h2o.splitFrame(data = datos_h2o, ratios = 0.8, seed = 123)
datos_h2o_train <- datos_h2o_split[[1]]
datos_h2o_validacion <- datos_h2o_split[[2]]

# Se define la variables empleadas por el autoencoder
predictores  <- setdiff(h2o.colnames(datos_h2o), "y")

# Entrenamiento del modelo autoencoder con 2 neuronas en la capa oculta.
autoencoder <- h2o.deeplearning(
                x = predictores,
                training_frame   = datos_h2o_train,
                validation_frame = datos_h2o_validacion,
                activation     = "Tanh",
                autoencoder    = TRUE,
                hidden         = c(2),
                epochs         = 50,
                ignore_const_cols    = FALSE,
                score_each_iteration = TRUE
               )
```

Diagnóstico

Para identificar el número de épocas adecuado se emplea la evolución del error de entrenamiento y validación.

```
scoring <- as.data.frame(autoencoder@model$scoring_history)
scoring %>%
  select(epochs, training_mse, validation_mse) %>%
  pivot_longer(cols = c(training_mse, validation_mse),
               names_to = "set",
               values_to = "mse") %>%
  ggplot() +
  geom_line(aes(x = epochs, y = mse, color = set)) +
  labs(title = "Evolución del error de entrenamiento y validación") +
  theme_bw() +
  theme(legend.position = "bottom")
```



A partir de las 5 épocas, la reducción en el *mse* es mínima. Una vez identificado el número óptimo de épocas, se reentrena el modelo, esta vez con todos los datos.

```
# Entrenamiento del modelo autoencoder
autoencoder <- h2o.deeplearning(
                x = predictores,
                training_frame   = datos_h2o,
                activation     = "Tanh",
                autoencoder    = TRUE,
                hidden         = c(2),
                epochs         = 5,
                ignore_const_cols    = FALSE,
                score_each_iteration = TRUE,
                seed = 123
              )
```

Error de reconstrucción

La función h2o.anomaly() permite obtener el error de reconstrucción empleando un modelo H2OAutoEncoderModel. Para ello, realiza automáticamente la codificación, decodificación y la comparación de los valores reconstruidos con los valores originales.

```
error_reconstruccion <- h2o.anomaly(
                          object = autoencoder,
                          data   = datos_h2o[, predictores],
                          per_feature = FALSE
                        )
error_reconstruccion <- as.data.frame(error_reconstruccion)
```

```
error_reconstruccion <- as.data.frame(error_reconstruccion)
error_reconstruccion <- error_reconstruccion[, 1]

# Distribución del error de reconstrucción.
tibble(error_reconstruccion) %>%
  ggplot(aes(x = error_reconstruccion)) +
  geom_density(fill = "steelblue") +
  labs(title = "Distribución de los errores de reconstrucción (Autoencoder)",
       x = "Error de reconstrucción") +
  theme_bw()
```

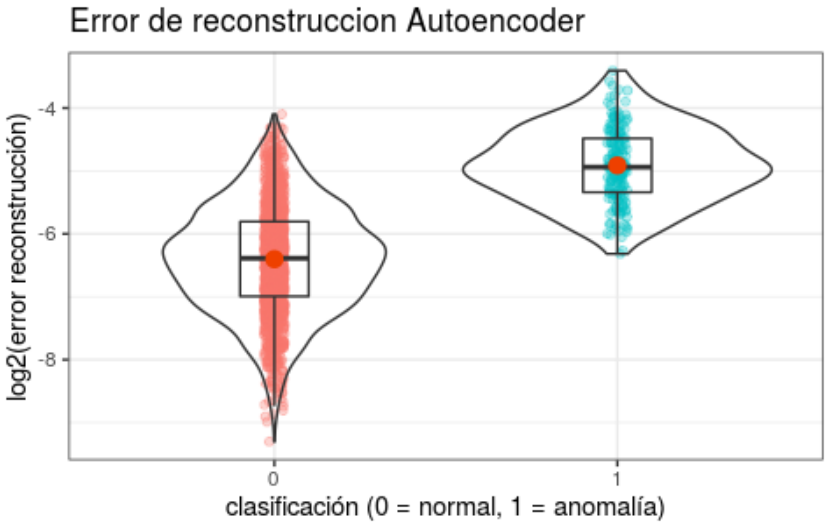
Detección de anomalías

Una vez que el error de reconstrucción ha sido calculado, se puede emplear como criterio para identificar anomalías. Asumiendo que la reducción de dimensionalidad se ha realizado de forma que la mayoría de los datos (los normales) queden bien representados, aquellas observaciones con mayor error de reconstrucción deberían ser las más atípicas.

En la práctica, si se está empleando esta estrategia de detección es porque no se dispone de datos etiquetados, es decir, no se conoce qué observaciones son realmente anomalías. Sin embargo, como en este ejemplo se dispone de la clasificación real, se puede verificar si realmente los datos anómalos tienen errores de reconstrucción más elevados.

```
# Se añade el error de reconstrucción al dataframe original.
datos$error_reconstruccion <- error_reconstruccion

ggplot(data = datos,
       aes(x = y, y = log2(error_reconstruccion))) +
geom_jitter(aes(color = y), width = 0.03, alpha = 0.3) +
```

```
geom_violin(alpha = 0) +
geom_boxplot(width = 0.2, outlier.shape = NA, alpha = 0) +
stat_summary(fun.y = "mean", colour = "orangered2", size = 3, geom = "point") +
labs(title = "Error de reconstruccion Autoencoder",
     x = "clasificación (0 = normal, 1 = anomalía)",
     y = "log2(error reconstrucción)") +
theme_bw() +
theme(legend.position = "none")
```



Acorde a la documentación, el set de datos *Cardiotocogrpahy* contiene 176 anomalías. Véase la matriz de confusión resultante si se clasifican como anomalías las 176 observaciones con mayor error de reconstrucción.

```
resultados <- datos %>%
              select(y, error_reconstruccion) %>%
              arrange(desc(error_reconstruccion)) %>%
              mutate(clasificacion = if_else(row_number() <= 176, "1", "0"))

mat_confusion <- MLmetrics::ConfusionMatrix(
                    y_pred = resultados$clasificacion,
                    y_true = resultados$y
                 )
mat_confusion
```

```
##        y_pred
## y_true    0    1
##      0 1575   80
##      1   80   96
```

De las 176 observaciones identificadas como anomalías, solo el 0.55% (96/176) lo son realmente. El porcentaje de falsos positivos es del 0.45%.

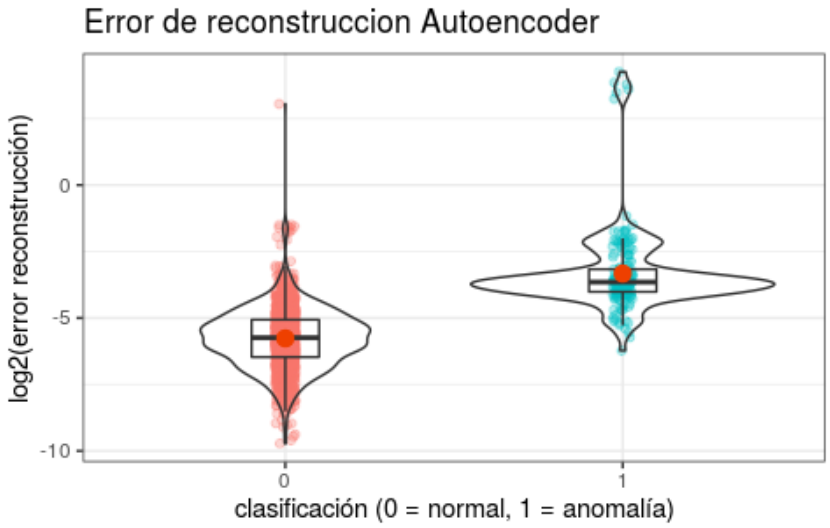
Reentrenamiento iterativo

El *autoencoder* anterior se ha entrenado empleando todas las observaciones, incluyendo las potenciales anomalías. Dado que el objetivo es generar un espacio de proyección para datos "normales", se puede mejorar el resultado reentrenando el modelo pero esta vez excluyendo las *n* observaciones con mayor error de reconstrucción (potenciales anomalías).

```
# Se eliminan las observaciones con un error de reconstrucción superior al
# percentil 70
cuantil       <- quantile(x = error_reconstruccion, probs = 0.7)
datos_trimmed <- datos %>% filter(error_reconstruccion < cuantil)

datos_trimmed_h2o <- as.h2o(
                        x = datos_trimmed,
                        destination_frame = "datos_trimmed_h2o"
                     )

autoencoder <- h2o.deeplearning(
                x = predictores,
                training_frame = datos_trimmed_h2o,
                activation     = "Tanh",
                autoencoder    = TRUE,
                hidden         = 2,
                epochs         = 5,
                score_each_iteration = TRUE,
                ignore_const_cols    = FALSE
               )

error_reconstruccion <- h2o.anomaly(
                          object = autoencoder,
                          data   = datos_h2o[, predictores],
                          per_feature = FALSE
                        )
error_reconstruccion <- as.data.frame(error_reconstruccion)
error_reconstruccion <- as.data.frame(error_reconstruccion)
error_reconstruccion <- error_reconstruccion[, 1]
```

```
# Se añade el error de reconstrucción al dataframe original.
datos$error_reconstruccion <- error_reconstruccion

ggplot(data = datos,
       aes(x = y, y = log2(error_reconstruccion))) +
geom_jitter(aes(color = y), width = 0.03, alpha = 0.3) +
geom_violin(alpha = 0) +
geom_boxplot(width = 0.2, outlier.shape = NA, alpha = 0) +
stat_summary(fun.y = "mean", colour = "orangered2", size = 3, geom = "point") +
labs(title = "Error de reconstruccion Autoencoder",
     x = "clasificación (0 = normal, 1 = anomalía)",
```

```
     y = "log2(error reconstrucción)") +
theme_bw() +
theme(legend.position = "none")
```




```
resultados <- datos %>%
              select(y, error_reconstruccion) %>%
              arrange(desc(error_reconstruccion)) %>%
              mutate(clasificacion = if_else(row_number() <= 176, "1", "0"))

mat_confusion <- MLmetrics::ConfusionMatrix(
                    y_pred = resultados$clasificacion,
                    y_true = resultados$y
                  )
mat_confusion
```

```
##        y_pred
## y_true    0    1
##      0 1599   56
##      1   56  120
```

Tras el reentrenamiento, 120 de las 176 (0.68%) observaciones identificadas como anomalías lo son realmente. Se ha conseguido reducir el porcentaje de falsos positivos al 0.32%.

Bibliografía

Outlier Analysis Aggarwal, Charu C.

Deep Learning with H2O by Arno Candel & Erin LeDell with assistance from Viraj Parmar & Anisha Arora Edited by: Angela Bartz