Sesion 3. Análisis de Anomalías

Jose Fernando Zea y Fernando López-Torrijos

13 de abril de 2021

Introducción

Machine Learning o Aprendizaje máquina es un conjunto de metodologías de predicción y/o prescripción que se utilizan, por ejemplo, para:

- Recomendar películas, música, libros, noticias, imágenes ó páginas web.
- Segmentar clientes basados en atributos comunes y en comportamientos de compra.
- Predecir la probabilidad de que un empleado abandone una organización,
- Predecir la fuga de un cliente (churn analysis).
- Cuantificar la probabilidad de falla de un dispositivo o máquina.
- Predecir si un cliente va a estar en mora en los próximos meses.
- Predecir la propensión a comprar de un cliente.
- Detectar fraude en una transacción en línea.

Aprendizaje supervisado

Hay modelos predictivos en donde una variable se predice en términos de otras. Por ejemplo, el valor de una vivienda se puede predecir en términos de diferentes atributos de la vivienda (Ver figura 1. En estos modelos se dispone de variables predictivas (X), que producen una variable respuesta.

- Las **variables predictivas** se conocen también como variables independientes, explicativas o regresoras. En un lenguaje muy propio del Machine Learning, también se denominan atributos (features).

- La **variable objetivo** se conoce también como variable dependiente, explicada, respuesta o regresada.

Los modelos anteriormente descritos se conocen como modelos de aprendizaje supervisado. La palabra *supervisado* refleja el hecho de que la variable objetivo auxilia o supervisa el aprendizaje dado que se conoce cuál debe ser el resultado. Otra manera de nombrar el objetivo es decir que se conoce la *etiqueta* del resultado. Dados los datos, el algoritmo de aprendizaje optimiza una función para encontrar una combinación de las variables predictivas que esté lo más cercana al valor etiquetaero de la variable objetivo.

La mayoría de los modelos supervisados se pueden clasificar en dos clases de modelos:

Modelo de regresión

Cuando el resultado a predecir es una variable continua.

Por ejemplo, en el pronostico del precio de una vivienda la variable a predecir es continua y, dada las combinaciones de los predictores, el valor que se generará estará en el dominio de los reales positivos.

Modelo de clasificación

Cuando la variable a predecir es categórica el modelo se denomina *de clasificación*, algunos ejemplos de este tipo de pronósticos son:

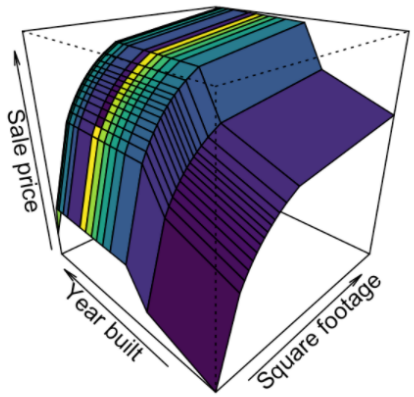


Figure 1: Superficie que relaciona año, metraje y precio de venta de casas

- Clasificar si una persona tomará un crédito que ofrece un banco (SI/NO).
- Clasificar si en una fotografía dermatológica el lunar es sospechoso de ser tumoral (SI/NO).

Aprendizaje no supervisado

Se denomina aprendizaje no supervisado a los modelos donde no hay una variable objetivo que ayude a optimizarlo. Se busca la definición de conglomerados (clusters). También se usa como técnica de reducción de dimensionalidad.

Las técnicas que se utilizan para la identificación de casos anómalos suelen corresponder a este tipo de aprendizaje.

Métodos basados en la proximidad

La idea de los métodos basados en la proximidad es modelar los valores atípicos como puntos que están aislados de los datos restantes. Este modelado se puede realizar de tres formas: análisis de conglomerados, análisis basado en densidad o análisis del vecino más cercano (KNN). En análisis de conglomerados y otros métodos basados en la densidad, las regiones densas en los datos se identifican directamente y los valores atípicos se definen como aquellos puntos que no se encuentran en dichas regiones. La principal diferencia entre el análisis de conglomerados y los métodos basados en la densidad es que los primeros segmentan puntos, mientras que los otros segmentan el espacio.

Son métodos que proporcionan un alto nivel de interpretabilidad en cuanto que las regiones de datos dispersas se pueden presentar en términos de combinaciones de los atributos originales. Por ejemplo, los conjuntos de restricciones sobre los atributos originales se pueden presentar como criterios específicos para que los puntos de datos particulares se interpreten como valores atípicos.

Vecinos más cercanos (KNN)

Es una técnica de clasificación no supervisada cuya concepto es muy sencillo. Clasifica a un nuevo miembro a una *clase* o categoría que ya existe. Lo determina según que sus atributos se parezcan a los casos de una categoría u otra. Se entiende vecindad como similitud debido a que se mide la *distancia* entre los atributos de cada par de casos. De ahí el nombre. Si es *vecino* de dos o más categorías diferentes, se asigna al grupo que cuente con más miembros similares.

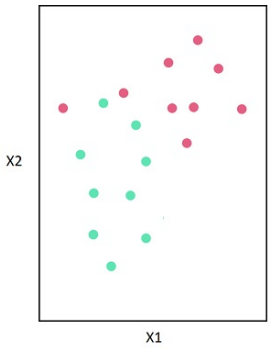


Figure 2: Esquema 1 del KNN

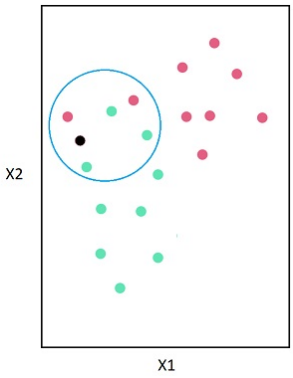


Figure 3: Esquema 2 del KNN

Suponga los grupos que se presentan en la Figura 2 y 3. Se quiere asignar un nuevo punto, en negro. KNN buscará los k puntos más cercanos a éste para encontrar la clase dominante del grupo. Para garantizar la existencia de una clase dominante, k debe ser un número impar.

En el ejemplo de la Figura 3 se utilizaron las cinco observaciones más cercanas ($K = 5$), para determinar a qué grupo asignar al nuevo *individuo*.

Hay muchas maneras de *medir distancias*. Cada manera se adapta de mejor o peor manera al tipo de atributos con que cuentan los individuos o casos. En cualquier caso, se recomienda *escalar* los atributos antes de aplicarles el algoritmo. *Escalar* una variable significa transformarla de tal modo que se haga comparable a otras, a pesar de que puedan tener escalas muy diferentes. Por ejemplo, el área de un país y su PIB tienen unidades de medida muy diferentes (miles de km^2 y miles de dólares). Ambas se pueden tomar como atributos para medir la *distancia* de un país con respecto a los demás. Si el valor máximo de una medida es mucho más grande que el de la otra en varios órdenes de magnitud, entonces dicha variable *dominará*. El escalamiento evita que ocurra.

Una desventaja del procedimiento es que si hay datos faltantes para uno o más individuos, esos individuos los descartará para la comparación. Obliga a asignarle un valor a dichos valores faltantes. Técnicamente se denomina *imputarle* valores a los datos faltantes.

Local Outlier Factor

El algoritmo LOF usa una idea similar al KNN, pero orientado a detectar datos anómalos. Calcula una puntuación (denominada factor de valor atípico local) que refleja el grado de anomalía de las observaciones. Mide la desviación de la *densidad local* de un punto con respecto a sus vecinos. La idea es detectar las muestras que tienen una densidad sustancialmente menor que sus vecinas.

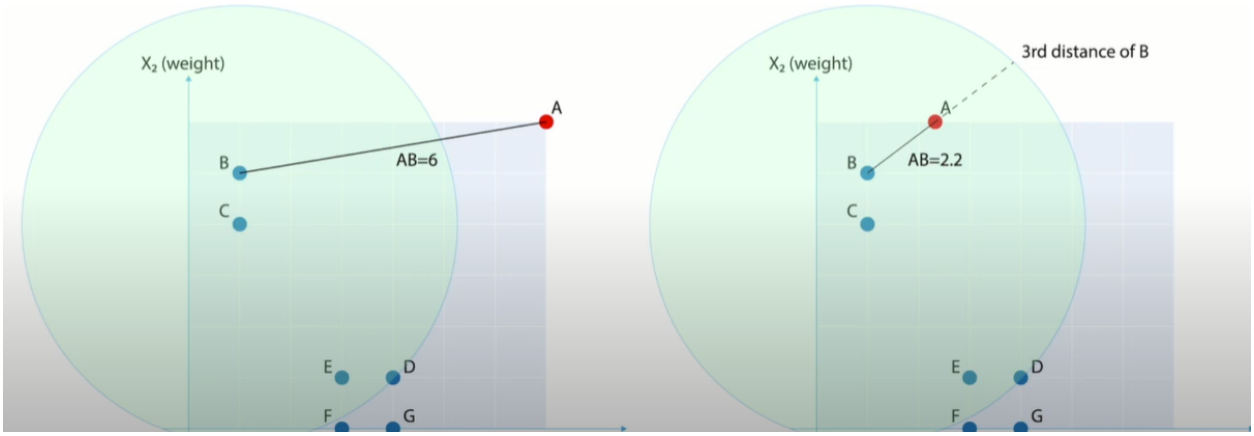
En la práctica, la densidad local se obtiene de los k vecinos más cercanos. La puntuación LOF de una observación es igual a la relación entre la densidad local promedio de sus k vecinos más cercanos y su propia densidad local. Se espera que un *caso* normal tenga una densidad local similar a la de sus vecinos, mientras que los *casos* anómalos se espera que tengan una densidad local mucho menor.

La fortaleza del algoritmo LOF es que se enfoca en qué tan aislada está la muestra respecto a su vecindario circundante.

Explicación Local Outlier Factor

1. La distancia alcanzable (RD) entre el punto A y B se calcula como el máximo entre la tercera mayor distancia a B y la distancia AB. Si el punto cae dentro del radio de la tercera mayor distancia al punto B lse considera como distancia alcanzable la tercera mayor distancia a B, en otro caso se considera la distancia al punto B.

$$RD_{AB} = max(k\text{-}\acute{e}sima\ dist(B), dist(A, B))$$



2. La distancia promedio alcanzable al punto A (RD_A) es el promedio de las distancias alcanzables de los tres vecinos más cercanos a A:

$$\overline{RD}_A = \frac{RD_{AB} + RD_{AC} + RD_{AD}}{3}$$

Si esta medida es muy grande significa que los vecinos de A están muy alejados, en otras palabras hay una baja densidad alrededor de A, o en otras palabras es baja la densidad local alcanzable:

$$LRD_A = \frac{1}{\overline{RD}_A}$$

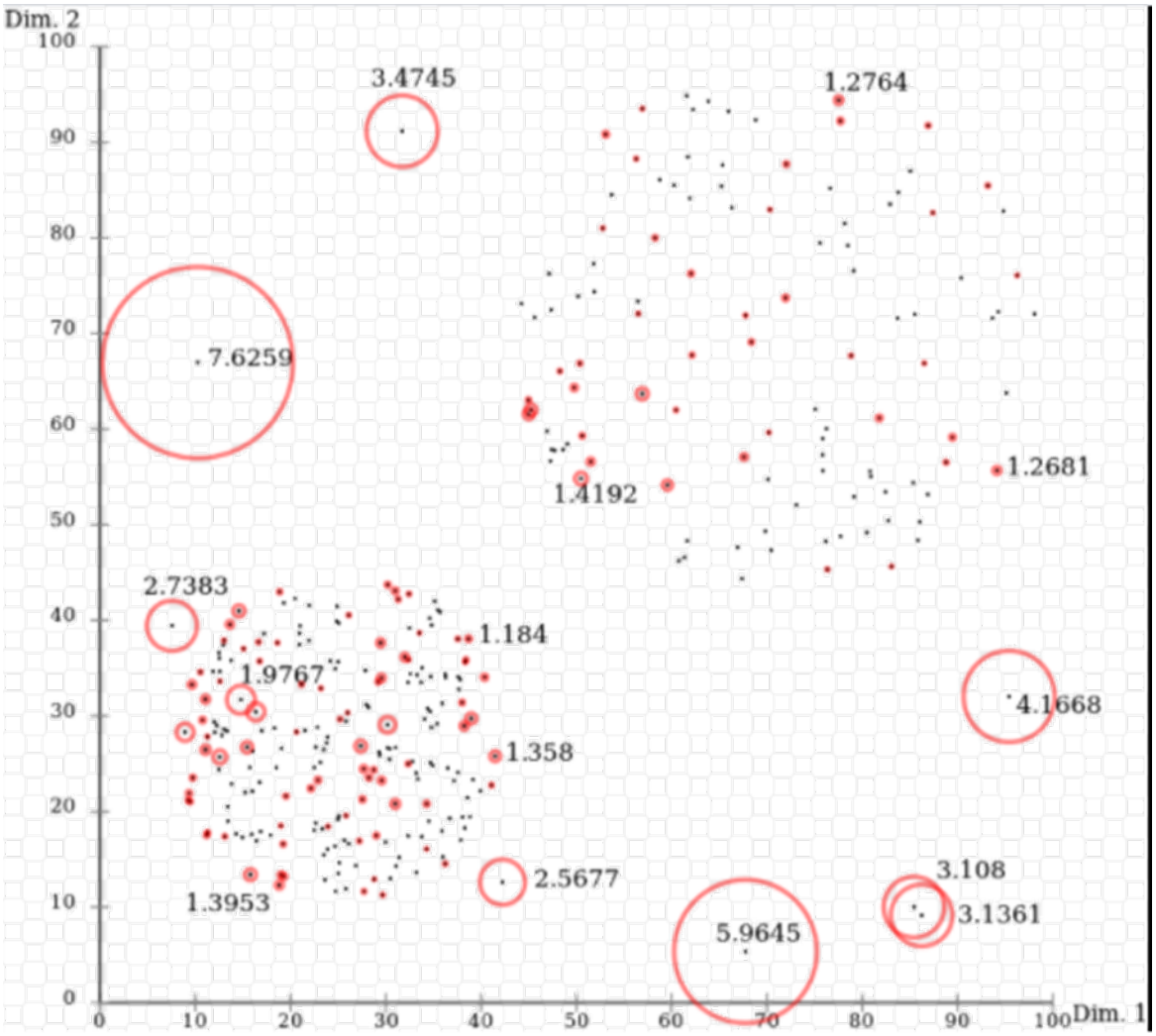
3. La distancia alcanzable de A también puede calcularse en las vecindades de A, es decir, calcular LRD_B, LRD_C y LRD_D y calcular la densidad promedio de los vecinos de A:

$$\frac{LRD_B + LRD_C + LRD_D}{3}$$

4. El factor de Outlier Local es el cociente entre el promedio de las densidades locales de los vecinos de A sobre la densidad local del punto A:

$$LOF_A = \frac{\frac{1}{3}\left(LRD_B + LRD_C + LRD_D\right)}{LRD_A}$$

Interpretación LOF: a continuación se muestra algunos valores del índice LOF:



En python de utiliza el módulo *LocalOutlierFactor* de la librería scikit_learn.

Ejemplo en python con datos simulados:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import LocalOutlierFactor
from sklearn.metrics import confusion_matrix
from sklearn import datasets
import random
from plotnine import ggplot, geom_point, aes, geom_abline
```

```
np.random.seed(42)
# datos de entrenamiento: 2100 numeros aleatorias de distribución normal
# con media cero y desviacion estandar 0.3
X_inliers = np.random.normal(0, 0.3, 200)
# Se convierte a 100 duplas para generar un scatter plot de 2 ejes
X_inliers = np.reshape(X_inliers, (-1, 2))
# Se concatenan, creando 200 duplas.
X_inliers = np.r_[X_inliers + 2, X_inliers - 2]
```

```
# Generacion de 20 duplas de outliers (10%) de números aleatorios entre -4 y 4
X_outliers = np.random.uniform(low=-4, high=4, size=(20, 2))
# Se concatenan en la variable X
X = np.r_[X_inliers, X_outliers]

n_outliers = len(X_outliers)
## Crea un arreglo de "unos" de la misma longitud que X
etiqueta = np.ones(len(X), dtype=int)
# Le asigna la etiqueta -1 a los outliers (los 20 ultimos)
etiqueta[-n_outliers:] = -1
```

Se puede acceder a las puntuaciones de anomalía de las muestras de entrenamiento mediante el atributo *negative_outlier_factor*.

```
# ajuste el modelo
clf = LocalOutlierFactor(n_neighbors = 20, contamination = 0.1)
# use fit_predict para calcular el número de errores al comparar
# con las etiquetas asignadas a la muestra de entrenamiento.
y_pred = clf.fit_predict(X)
n_errors = (y_pred != etiqueta).sum()
X_scores = clf.negative_outlier_factor_
# np.random.choice(X_scores, 20) # Vea una muestra de "puntajes"
```

```
plt.title("Local Outlier Factor (LOF)")
plt.scatter(X[:, 0], X[:, 1], color='k', s=3., label='Puntos a evaluar')
# los circulos son proporcionales al puntaje de outlier
radius = (X_scores.max() - X_scores) / (X_scores.max() - X_scores.min())
plt.scatter(X[:, 0], X[:, 1], s=1000 * radius, edgecolors='r',
            facecolors='none', label='Puntaje como anómalo')
plt.axis('tight')
```

```
## (-4.099800037239011, 3.463844771291937, -4.063381121129955, 3.656843731955662)
```

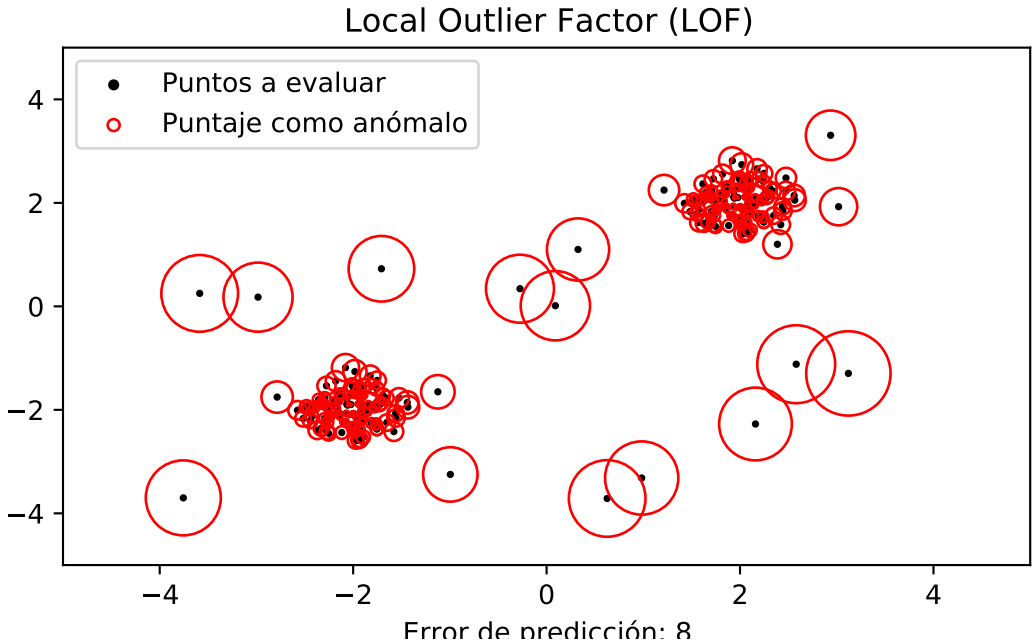
```
plt.xlim((-5, 5))
```

```
## (-5.0, 5.0)
```

```
plt.ylim((-5, 5))
```

```
## (-5.0, 5.0)
```

```
plt.xlabel("Error de predicción: %d" % (n_errors))
legend = plt.legend(loc='upper left')
legend.legendHandles[0]._sizes = [10]
legend.legendHandles[1]._sizes = [20]
plt.show()
```

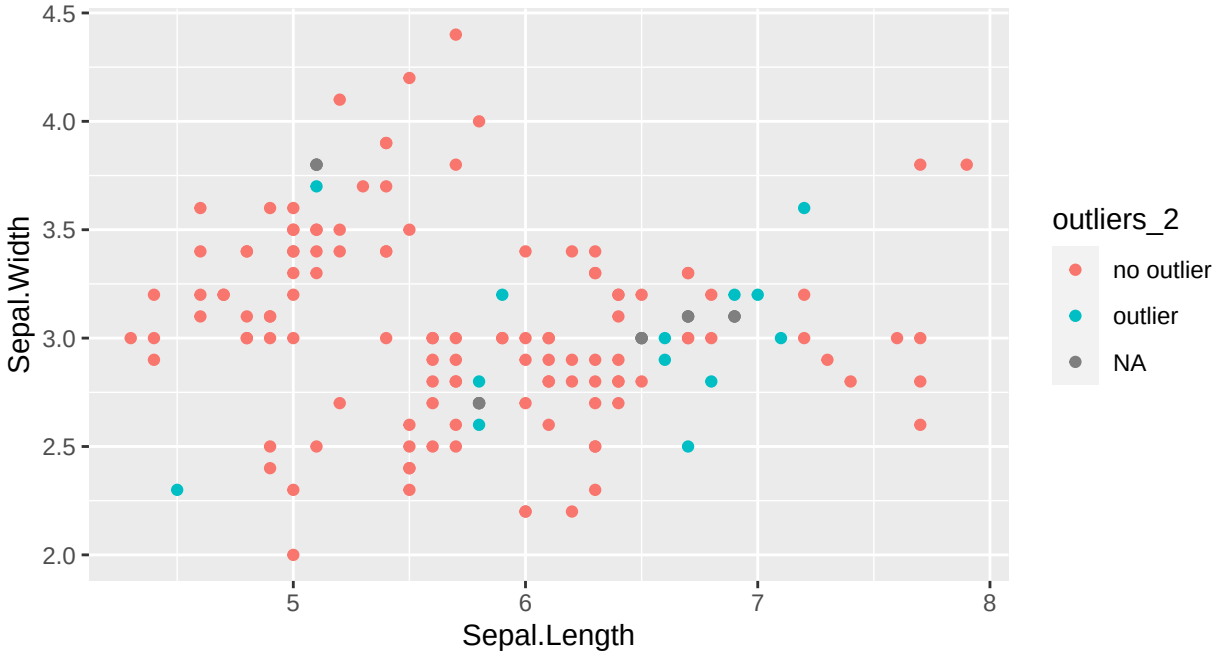


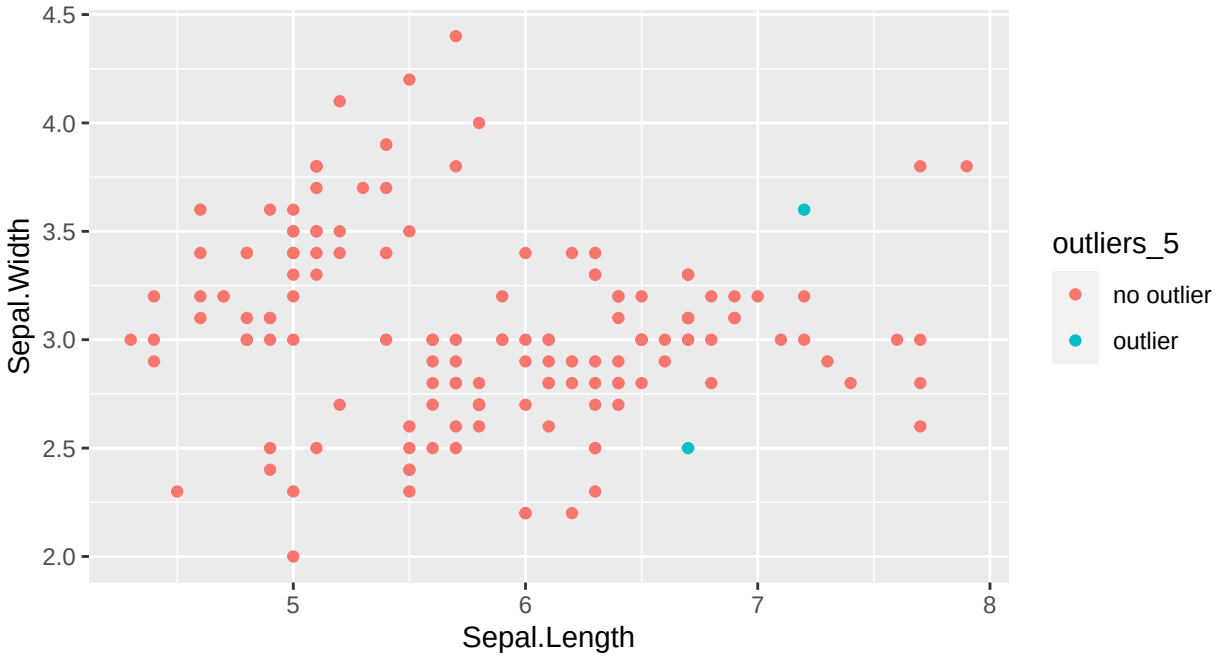
El error de predicción puede ser porque *coincidió* el número aleatorio *outlier* con los *inliers*, o por números aleatorios *inliers* muy apartados de la media. La matriz de confusión presenta esos errores:

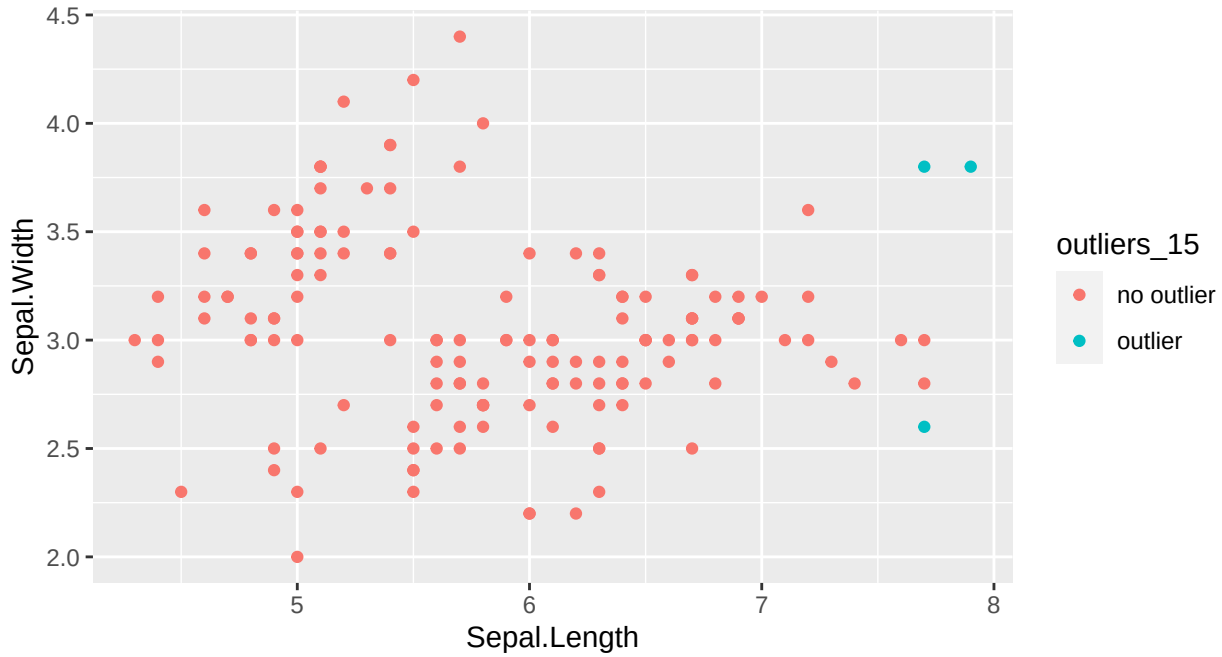
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(etiqueta, y_pred)
cm
```

```
## array([[ 17,   3],
##        [  5, 195]], dtype=int64)
```

Usualmente como regla empírica se considera que los valores con un LOF mayor a 1.5 se consideran como puntos anómalos aunque esto depende mucho del valor de k (número de vecinos).


Sepal.Width
4.5
4.0
3.5
3.0
2.5
2.0
5
6
7
8
Sepal.Length
outliers_2
no outlier
outlier
NA


Sepal.Width
4.5
4.0
3.5
3.0
2.5
2.0
5
6
7
8
Sepal.Length
outliers_5
no outlier
outlier




```
iris = datasets.load_iris()
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_df['target'] = iris['target']
iris_df.columns = ['sepal_length', 'sepal_width', 'petal_length',
       'petal_width', 'species']
```

```
clf = LocalOutlierFactor(n_neighbors = 5, novelty = False)
# use fit_predict para calcular el número de errores al comparar
# con las etiquetas asignadas a la muestra de entrenamiento.
X = iris_df[['sepal_length', 'sepal_width']]
y_pred = clf.fit_predict(X)
lof_scores = -1 * clf.negative_outlier_factor_
outlier_5 = lof_scores > 2
unique, counts = np.unique(outlier_5, return_counts=True)
unique, counts
# np.random.choice(X_scores, 20) # Vea una muestra de "puntajes"
```

```
## (array([False,  True]), array([148,   2], dtype=int64))
```

```
clf = LocalOutlierFactor(n_neighbors = 15, novelty = False)
# use fit_predict para calcular el número de errores al comparar
# con las etiquetas asignadas a la muestra de entrenamiento.
X = iris_df[['sepal_length', 'sepal_width']]
y_pred = clf.fit_predict(X)
lof_scores = -1 * clf.negative_outlier_factor_
outlier_15 = (lof_scores > 2)+0
unique, counts = np.unique(outlier_15, return_counts=True)
unique, counts
# np.random.choice(X_scores, 20) # Vea una muestra de "puntajes"
```

```
## (array([0, 1]), array([147,   3], dtype=int64))
```

```
iris_df['outlier_15'] = outlier_15
```

```
(ggplot(iris_df, aes('sepal_length', 'sepal_width', color = 'factor(outlier_15)'))
 + geom_point())
```

Árboles de decisión

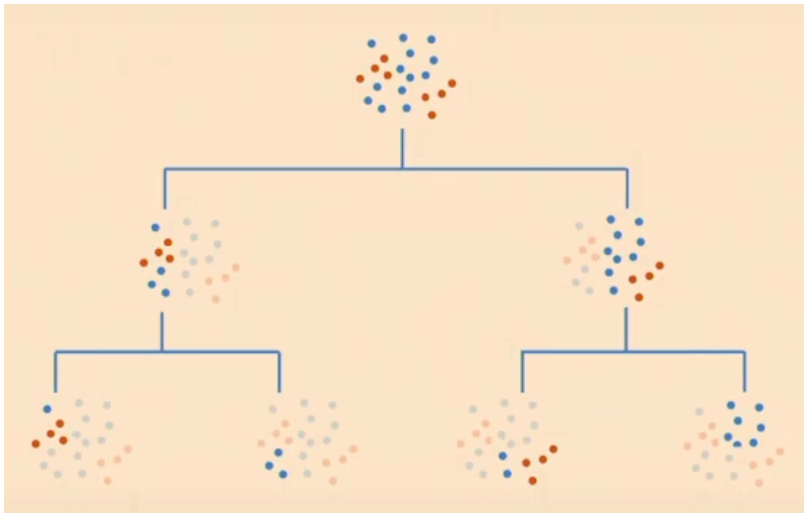


Figure 4: Árbol

Es una técnica previa al Machine Learning. Se utilizaba para estratificar variables en grupos lo más homogéneos posibles. Estratificación entendida en términos de muestreo. Lo que se busca es encontrar valores de corte en una ó mas variables explicativas de tal modo que la variable de interés quede dividida en grupos (estratos) lo más homogéneos posibles.

Es una técnica no supervisada para pronosticar el valor asignándole el promedio de dicho grupo o para clasificar la ubicación de la variable de interés en alguno de los grupos.

Para problemas en que se desea asignar el promedio del grupo, la técnica determina para cada variable una sucesión de puntos que divida la base en dos particiones (P_1 y P_2) y determina en cuál es menor la suma del cuadrado de los errores (SCE):

$$SCE = \sum_{i \in P_1} (y_i - \bar{y}_1)^2 + \sum_{i \in P_2} (y_i - \bar{y}_2)^2$$

En la Figura 5, de los diferentes cortes, probablemente el tercero es el que genere la menor SCE.

Figure 5: Puntos de corte para la partición

Se realiza tal selección para cada una de las variables predictoras y se elige para la primera partición la variable que obtenga el menor valor de SCE. Recursivamente se vuelve a realizar tal algoritmo para cada partición (P_1 y P_2) volviendo a particionar cada rama. Esto quiere decir que vuelve a utilizar variables que

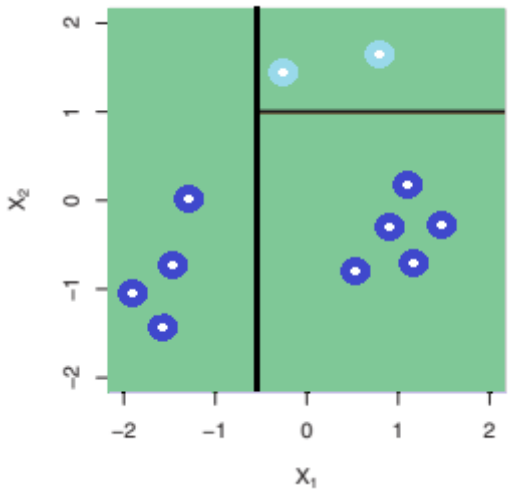


Figure 6: Esquema de funcionamiento de un árbol

ya han sido elegidas para una partición anterior. Se trata de una técnica computacionalmente intensa, pero veloz.

La Figura 6 representa el proceso. El primer corte se realizó sobre la variable X1. El segundo corte se realzó sobre la partición de la derecha respecto a la variable x2. El proceso puede continuar hasta que sólo haya un elemento en cada grupo.

A medida se profundiza un árbol se tiende al *sobreajuste*. Éste se limita fijando criterios acerca del número mínimo de nodos (hojas) por rama o la máxima profundidad.

En general, si no hay demasiadas variables respecto a los recursos computacionales disponibles, se permite crecer al árbol completamente y luego se *poda* (prune) con el objeto de hacer más sencillo el modelo. La sencillez o parsimonia se denomina, en este contexto, nivel de *pureza*. Al ser más sencillo es más eficiente, disminuirá el sobreajuste y no perderá eficacia.

En problemas de clasificación lo único que cambia es el criterio con que se subdivide el árbol. Un árbol tiene mayor *pureza* si cada partición tiene una mayor proporción de nodos de una de las *clases* o categorías en las que se están clasificando las variables.

La poda de un árbol utiliza la técnica de *regularización*, consistente en una penalización asociada al costo y/o complejidad del árbol. La complejidad se refiere al número de nodos terminales. A menor número, menos complejo.

$$SCE_{\text{Complexity Parameter}} = SCE + (\alpha)(\text{número de nodos terminales})$$

A menor penalización, mayor complejidad, y viceversa.

Para seleccionar la mejor *poda* se evalúan los datos para una sucesión de diferentes valores del parámetro de complejidad (α ó CP).

Los árboles tienen la ventaja de que es entendible el resultado, se pueden identificar las variables más importantes para la clasificación o regresión y maneja sin problemas los *missing data*. Los ignora, pero utiliza la información de las restantes variables de las que sí se tiene información. Y maneja indistintamente variables numéricas, densas o disgregadas, y categóricas.

Se presenta una ligera desventaja si hay variables muy correlacionadas. Si hay dos predictores muy correlacionados, la selección de uno de estos dos para la partición es como si se realizara al azar ya que la diferencia se deberá a diferencias aleatorias en los datos de entrenamiento. Si se eliminara de la data una de

las dos variables, la otra podría ser seleccionada dos veces, en dos ramas distintas, realzando su importancia relativa, lo cual tal vez no ocurriría si están presentes ambas. También puede ser desventajoso el hecho de que variables con mayor cardinalidad en el número de diferentes valores son más favorecidos que aquellas con relativamente pocos valores distintos.

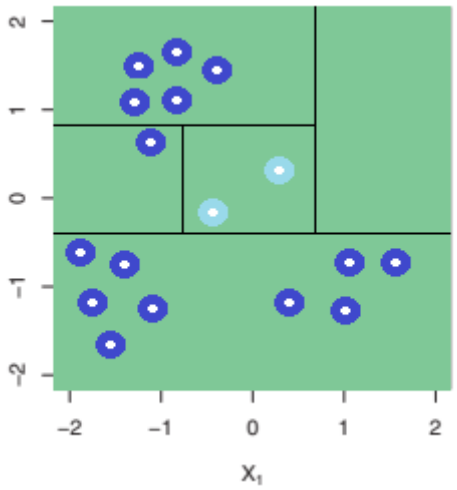


Figure 7: Desventaje del árbol

Obsérvese que el espacio de partición de la data es rectangular. Los árboles pueden dar resultados menos óptimos que otras técnicas. Y alterando levemente los datos, el árbol resultante puede llegar a ser sustancialmente diferente. Por esa razón surgieron las técnicas de *ensamble* de árboles, que promedian los resultados de multitud de árboles con el objeto de obtener resultados estables y que han probado obtener resultados incluso mejores que otras técnicas.

Isolation forest[1]

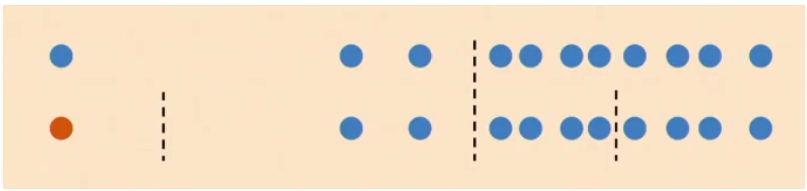


Figure 8: Árbol aislado

Para aislar un dato se selecciona aleatoriamente una característica y luego se selecciona aleatoriamente un valor de separación.

Dado que la partición se aplica de nuevo a los resultado se puede representar mediante una estructura de árbol. El número de divisiones necesarias para aislar una muestra es equivalente a la longitud de la ruta desde el nodo raíz hasta el nodo de terminación. En el ejemplo de la Figura 8 bastó con dos procesos de división, pero se podría continuar, como en la Figura 9.

De lo que se trata es de aislar datos en un conjunto de árboles independientemente unos de otros, cada uno a partir de un conjunto de datos diferente y *ensamblar* un resultado. Por ello se denomina un *ensamble* de árboles. Para encontrar árboles diferenciados se construyen submuestras de datos mediante la técnica del *bootstrap*: un muestreo con repetición del tamaño del conjunto original. Si bien no es intuitivo el método

[1] Gráfica tomada de "cubonacci"

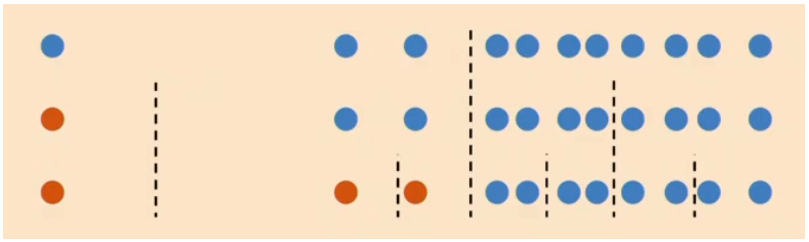


Figure 9: Árbol aislado

de emsamblaje (bagging en inglés), este método propuesto en 1996 funciona muy bien para muchos tipos de modelos. Obtiene mejora en las predicciones y reduce la varianza de éstas. Luego hubo una mejorá que se denominó *Random Forest*. Es una mejora frente al paradigma del *bagging* en el sentido que evita que haya correlación entre árboles. ¿Cómo? Además de *muestrear con reemplazo individuos*, *muestrea variables aleatoriamente sin reemplazo*. Si bien el método del Isolation Forest no es el mismo, el principio sí, y por ello se denominó de esta manera.

Para determinar la predicción del Isolation Forest se utiliza *el voto de la mayoría* (la moda) para problemas de clasificación y el *promedio* para los problemas de regresión. Puede ser un promedio ponderado.

La partición aleatoria produce trayectorias notablemente más cortas para las anomalías, por lo tanto, cuando un bosque de árboles aleatorios produce colectivamente recorridos más cortos para muestras particulares, es muy probable que lo sean.

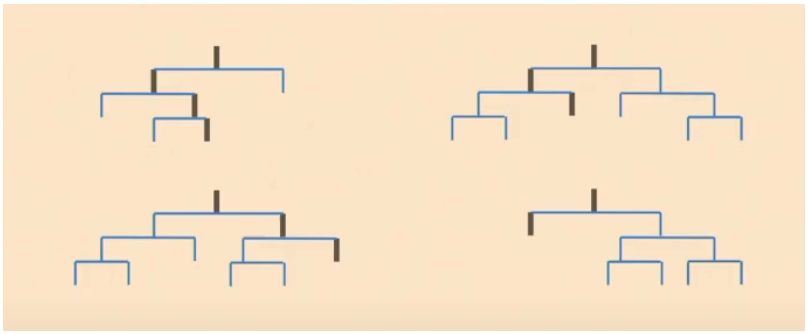


Figure 10: Ensamble de árboles

Ejemplo en python

```
from sklearn.ensemble import IsolationForest
import warnings
warnings.filterwarnings("ignore")
```

```
df = pd.read_csv('medidas_cuerpo2.csv')
df.head(6)
```

```
##    ID  Peso Sexo  Estatura  circun_cuello  circun_muneca
## 0   1  47.6    F      1.57           29.5           13.9
## 1   2  68.1    M      1.66           38.4           16.0
## 2   3  68.0    M      1.90           36.5           16.6
## 3   4  80.0    M      1.76           38.0           17.1
## 4   5  68.1    M      1.83           38.0           17.1
## 5   6  56.1    F      1.66           33.0           14.7
```

Se procede a definir el modelo:

```
model=IsolationForest(n_estimators = 50, max_samples = 'auto', \
    contamination = float(0.1), max_features = 1.0)
model.fit(df[['Peso']])
```

```
## IsolationForest(contamination=0.1, n_estimators=50)
```

Se añaden los puntajes y las etiquetas en la base de datos:

```
df['puntaje'] = model.decision_function(df[['Peso']])
df['anomalo'] = model.predict(df[['Peso']])
df[["Peso", "puntaje", "anomalo"]].head(6)
```

```
##    Peso   puntaje  anomalo
## 0  47.6  0.003558        1
## 1  68.1  0.181116        1
## 2  68.0  0.179215        1
## 3  80.0 -0.003558       -1
## 4  68.1  0.181116        1
## 5  56.1  0.113067        1
```

Imprímanse sólo los anómalos:

```
anomalia = df.loc[df['anomalo'] == -1]
anomalia_index = list(anomalia.index)
print(anomalia)
```

```
##     ID   Peso Sexo  Estatura  circun_cuello  circun_muneca   puntaje  anomalo
## 3    4   80.0    M      1.76           38.0           17.1 -0.003558       -1
## 10  11  102.2    M      1.79           41.5           17.1 -0.243520       -1
## 11  12   46.7    F      1.49           31.5           13.8 -0.045535       -1
```

Sea ahora el análisis multidimensional:

```
df = pd.read_csv('medidas_cuerpo2.csv')
df = df[["Peso", "Estatura", "circun_cuello", "circun_muneca"]]
model = IsolationForest(max_samples = 100)
model.fit(df)
```

```
## IsolationForest(max_samples=100)
```

Las cuatro variables se usan como *predictores*:

```
df['puntaje'] = model.decision_function(df[["Peso", "Estatura", "circun_cuello", "circun_muneca"]])
df['anomalo'] = model.predict(df[["Peso", "Estatura", "circun_cuello", "circun_muneca"]])
```

Resultado:

```
anomalia = df.loc[df['anomalo'] == -1]
anomalia_index = list(anomalia.index)
print(anomalia)
```

```
##      Peso  Estatura  circun_cuello  circun_muneca   puntaje  anomalo
## 0    47.6      1.57           29.5           13.9 -0.117975       -1
## 10  102.2      1.79           41.5           17.1 -0.128661       -1
## 11   46.7      1.49           31.5           13.8 -0.119985       -1
## 14   52.5      1.52           32.5           14.4 -0.041161       -1
## 16   79.1      1.82           38.0           18.0 -0.031802       -1
```

Los mismos tres casos que etiquetó con solo el Peso.

Ejemplo

```
df = pd.read_csv('PimaIndiansDiabetes.csv', index_col=[0])
df.head(6)
```

```
##    pregnant  glucose  pressure  triceps  insulin  mass  pedigree  age diabetes
## 1         6    148.0      72.0     35.0      NaN  33.6     0.627   50      pos
## 2         1     85.0      66.0     29.0      NaN  26.6     0.351   31      neg
## 3         8    183.0      64.0      NaN      NaN  23.3     0.672   32      pos
## 4         1     89.0      66.0     23.0     94.0  28.1     0.167   21      neg
## 5         0    137.0      40.0     35.0    168.0  43.1     2.288   33      pos
## 6         5    116.0      74.0      NaN      NaN  25.6     0.201   30      neg
```

Este conjunto de datos es originalmente del Instituto Nacional de Diabetes y Enfermedades Digestivas y Renales de EEUU. El objetivo del conjunto de datos es predecir de forma diagnóstica si un paciente tiene diabetes o no, basándose en determinadas medidas de diagnóstico incluidas en el conjunto de datos. Se impusieron varias restricciones a la selección de estas instancias de una base de datos más grande. En particular, todos los pacientes aquí son mujeres de al menos 21 años de edad de origen indio Pima.

Se utilizará para detectar un conjunto de individuos *anómalos* en sus medidas médicas, independientemente de que temngan diabetes o no..

Se convierte la etiqueta a un número binario. “1” si tiene diabetes, “0” de lo contrario.

```
df[['diabetes']] = np.where(df['diabetes'].str.contains("pos"), 1, 0)
df.head(6)
```

```
##    pregnant  glucose  pressure  triceps  insulin  mass  pedigree  age  diabetes
## 1         6    148.0      72.0     35.0      NaN  33.6     0.627   50         1
## 2         1     85.0      66.0     29.0      NaN  26.6     0.351   31         0
## 3         8    183.0      64.0      NaN      NaN  23.3     0.672   32         1
## 4         1     89.0      66.0     23.0     94.0  28.1     0.167   21         0
## 5         0    137.0      40.0     35.0    168.0  43.1     2.288   33         1
## 6         5    116.0      74.0      NaN      NaN  25.6     0.201   30         0
```

Obsérvese que hay datos con *missing values*. Infortunadamente, a diferencia de los árboles de decisión, *Isolation Forest* requiere sólo casos con datos completos.

```
# Se elimina la etiqueta que no interesa
# X = df.iloc[:,:-1]
X = df
# Sólo datos completos, sin NaN
X.dropna(axis = 0, how = 'any', inplace = True)
X.head(6)
```

```
##     pregnant  glucose  pressure  triceps  ...  mass  pedigree  age  diabetes
## 4          1     89.0      66.0     23.0  ...  28.1     0.167   21         0
## 5          0    137.0      40.0     35.0  ...  43.1     2.288   33         1
## 7          3     78.0      50.0     32.0  ...  31.0     0.248   26         1
## 9          2    197.0      70.0     45.0  ...  30.5     0.158   53         1
## 14         1    189.0      60.0     23.0  ...  30.1     0.398   59         1
## 15         5    166.0      72.0     19.0  ...  25.8     0.587   51         1
##
## [6 rows x 9 columns]
```

Se ajusta el modelo, y como se tienen un data frame de pandas, se reforma a una matriz, la entrada requerida para el modulo de Isolation Forest.

```
model = IsolationForest(max_samples = 100) # Puede colocarse como parámetro contamination = valor
X_matrix = X.values.reshape(-1, 9) # Número de features
len(X_matrix) # filas
```

```
## 392
```

```
len(X_matrix[0]) # columnas
```

```
## 9
```

```
model.fit(X_matrix)
```

```
## IsolationForest(max_samples=100)
```

Todas las variables se usan como *predictores*.

Se añaden las columnas al data frame de pandas:

```
X['puntaje'] = model.decision_function(X_matrix)
X['anomalo'] = model.predict(X_matrix)
```

Resultado:

```
# anomalia = X.loc[X['anomalo'] == -1]
# df.loc[df['column_name'] == some_value]
anomalia = X[X['anomalo'] == -1]
anomalia_index = list(anomalia.index)
print(anomalia)
```

```
##      pregnant  glucose  pressure  triceps  ...  age  diabetes   puntaje  anomalo
## 5           0    137.0      40.0     35.0  ...   33         1 -0.092184       -1
## 7           3     78.0      50.0     32.0  ...   26         1 -0.009807       -1
## 9           2    197.0      70.0     45.0  ...   53         1 -0.113865       -1
## 14          1    189.0      60.0     23.0  ...   59         1 -0.122694       -1
## 15          5    166.0      72.0     19.0  ...   51         1 -0.002194       -1
## ..        ...      ...       ...      ...  ...  ...       ...       ...      ...
## 741        11    120.0      80.0     37.0  ...   48         1 -0.011492       -1
## 745        13    153.0      88.0     37.0  ...   39         0 -0.031020       -1
## 748         1     81.0      74.0     41.0  ...   32         0 -0.013174       -1
## 754         0    181.0      88.0     44.0  ...   26         1 -0.074279       -1
## 764        10    101.0      76.0     48.0  ...   63         0 -0.060115       -1
##
## [94 rows x 11 columns]
```

Por defecto toma el 10% de los casos como anómalos.

Hay tanto personas diabéticas como no diabéticas.

Ejercicio

Realice el mismo ejercicio, pero separando los individuos con diabetes y sin diabetes.