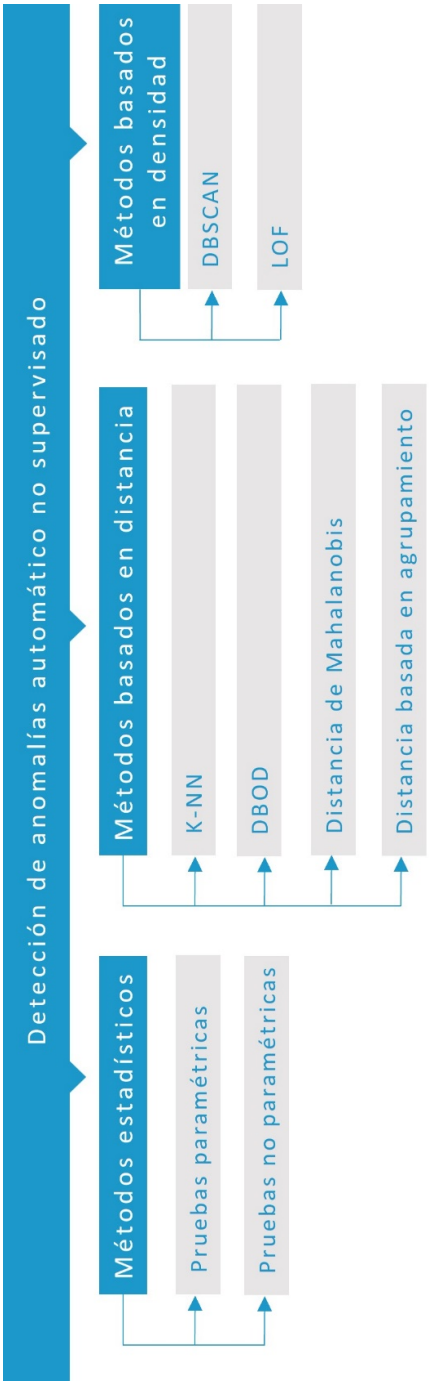
Aprendizaje Automático No Supervisado

Tema 8. Detección de anomalías

Índice

Esquema



8.1. Introducción y objetivos

Los problemas de detección de anomalías son una **aplicación común del aprendizaje automático.** Pero ¿qué es la detección de anomalías? Imagina que trabajas en una empresa que fabrica motores de aviones. A medida que los motores salen de la cadena de montaje, se realiza una fase de aseguramiento de la calidad en la cual se miden algunas características de los motores (ejemplo: calor generado y vibración). Supongamos que tenemos un conjunto de datos con m motores que han sido evaluados positivamente y han dado como resultado lo dibujado en la siguiente gráfica.

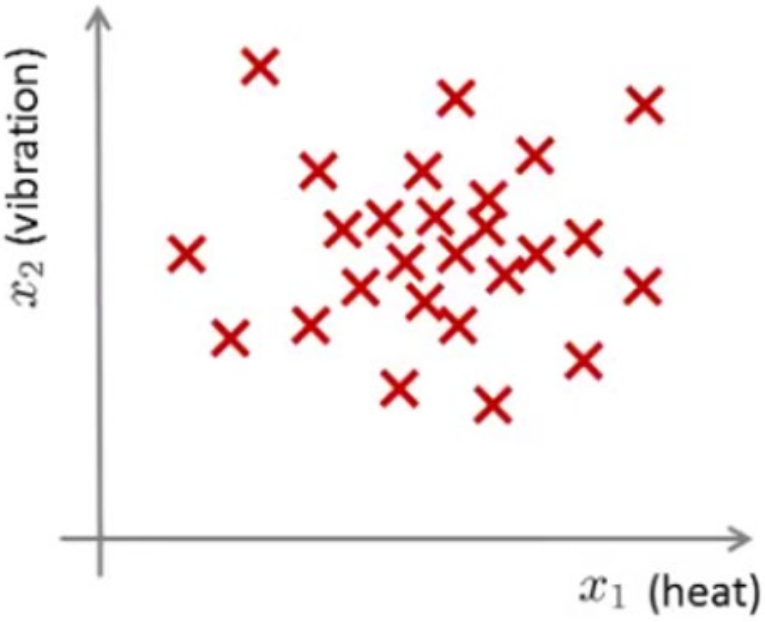


Figura 1. Distribución de valores de vibración y calor generado por motores. Fuente: Indira, 2020.

Supongamos que al día siguiente se fabrica un nuevo motor y se utiliza un método de detección de anomalías para comprobar su correcto funcionamiento, comparando el de este motor nuevo con respecto a los motores previos. Obtenemos una gráfica como la Figura 2.

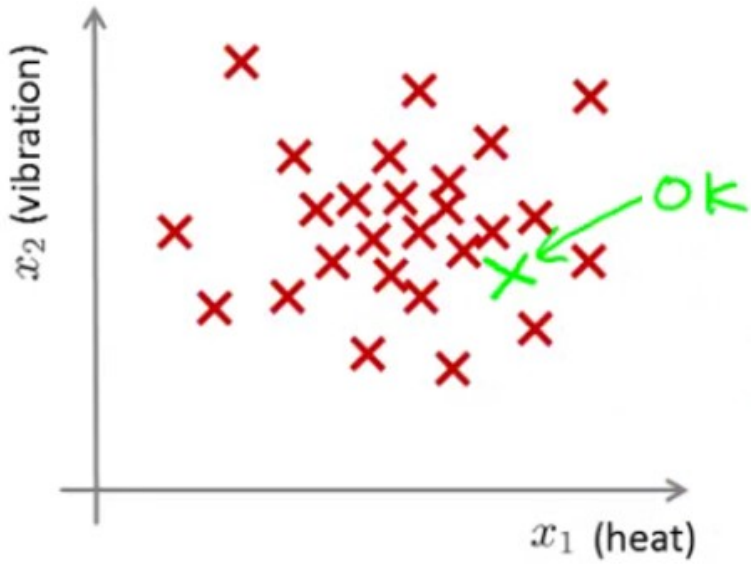


Figura 2. Distribución de valores de vibración y calor generado por motores. Ejemplo de una instancia correcta. Fuente: adaptado de Indira, 2020.

Lo más probable es que el motor funcione correctamente, pues su comportamiento es muy similar al de motores previos. Sin embargo, la gráfica es como la Figura 3.

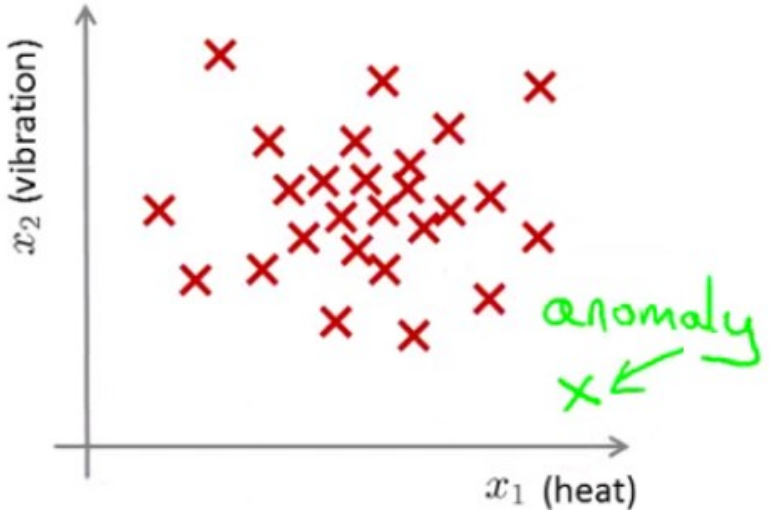


Figura 3. Distribución de valores de vibración y calor generado por motores. Ejemplo de una instancia anómala. Fuente: adaptado de Indira, 2020.

Lo más probable es que el motor presente algún tipo de anomalía.

Ideas clave

En este tipo de problemas partimos de un conjunto de datos que contiene registros normales, o bien la gran mayoría de ellos lo son. El objetivo es utilizar este conjunto como referencia y observar si existen nuevos ejemplos que son anómalos.

La detección de anomalías combina las técnicas de aprendizaje supervisado para generar un modelo de valores normales y, posteriormente, se utiliza este modelo con nuevos registros para detectar valores anómalos o inusuales.

Una anomalía es una observación que se desvía del comportamiento o patrón esperado de un conjunto de datos determinado. Estas técnicas también se conocen en la literatura con el nombre de detección de *outliers.* En esencia, un *outlier* es un valor poco habitual y, por tanto, puede ser considerado una anomalía.

La detección de anomalías se utiliza ampliamente en diversas aplicaciones, como detección de fraude, detección de intrusiones en la red, diagnóstico médico, mantenimiento predictivo, control de calidad de fabricación, monitoreo de redes sociales, robo de identidad, monitoreo de tráfico, violación de datos, video vigilancia, detección de *fake news,* entre otros.

Técnicas para detectar anomalías

Existen diversas técnicas que nos ayudan a detectar anomalías, pero las más comunes son (Desai, 2023):

- **Detección de anomalías con algoritmos supervisados:** se utilizan datos etiquetados para entrenar un modelo que incluye instancias normales y anómalas.
- **Detección de anomalías con algoritmos no supervisados:** el modelo aprende a identificar anomalías sin conocimiento previo de lo que constituye un comportamiento normal. Muy útil cuando no hay etiquetas disponibles o cuando las

anomalías son raras o difíciles de encontrar.

- **Detección de anomalías con algoritmos semisupervisados:** utiliza la combinación de datos etiquetados y no etiquetados para entrenar el modelo. El modelo aprende a identificar anomalías en los datos sin etiquetar en función de lo que aprende de los datos etiquetados.

- **Detección estadística de anomalías:** utilizando técnicas estadísticas para detectar anomalías basadas en la distribución de los datos. Implica identificar puntos de datos que quedan fuera de un rango predefinido de valores o que tienen propiedades estadísticas inusuales.

Cada técnica de detección de anomalías tiene sus propias fortalezas y debilidades, y la elección de la técnica depende de la aplicación específica y las características del conjunto de datos.

La detección de valores atípicos y el análisis de agrupamiento son tareas muy relacionadas. La agrupación encuentra la mayoría de los patrones en un conjunto de datos y los organiza en consecuencia, mientras que la detección de valores atípicos intenta capturar aquellos casos excepcionales que se desvían de la mayoría de los patrones.

Tipos de valores atípicos

Se pueden clasificar en tres categorías:

- **Valor atípico global:** hay una desviación significativa del resto de datos.

- **Valor atípico contextual:** el objeto se desvía significativamente según un contexto seleccionado. Por ejemplo, 28 grados centígrados es atípico para un invierno en Siberia, pero no un valor atípico para un verano en España.

- **Valor atípico colectivo:** cuando un subconjunto de datos en un conjunto se desvía del conjunto completo, incluso si los objetos de datos individuales pueden no ser

valores atípicos. Por ejemplo, un electrocardiograma cuando a un paciente le quitan los electrodos presenta un trazo diferente a un electrocardiograma normal.

En este tema veremos métodos estadísticos, algoritmos no supervisados de agrupamiento basados en distancias y basados en densidad que nos ayudan a detectar anomalías.

Ventajas y desventajas

Antes de entrar en materia veamos algunas ventajas y desventajas de los algoritmos de detección de anomalías.

Ventajas:

- **Detección temprana de anomalías:** ayuda a identificar problemas potenciales y oportunidades de mejora antes de que se vuelvan críticos. Tomando medidas proactivas se pueden prevenir o mitigar los impactos negativos.
- **Precisión mejorada:** una vez se eliminan los valores atípicos, los análisis estadísticos y los modelos de aprendizaje automático pueden lograr una mayor precisión y confiabilidad.
- **Seguridad mejorada:** la detección de anomalías puede ayudar a identificar posibles amenazas a la seguridad, como intrusiones en la red, fraude o ataques cibernéticos.
- **Eficiencia mejorada:** identificar y eliminar valores atípicos, se pueden optimizar operaciones, reducir errores y mejorar la productividad.

Desventajas:

- **Falsos positivos:** con la detección de anomalías se puede identificar puntos de datos normales como valores atípicos, lo que genera falsos positivos. Esto puede generar alertas o acciones innecesarias y una pérdida de tiempo y de recursos.
- **Preprocesamiento de datos:** para poder detectar bien las anomalías es necesario

realizar un preprocesamiento de datos cuidadoso y una selección de funciones para garantizar resultados precisos y confiables. Este proceso puede ser engorroso y consumir mucho tiempo y recursos.

- **Selección del modelo:** es un desafío encontrar el modelo correcto, ya que cada modelo tiene sus fortalezas y sus debilidades y puede ser más o menos adecuado para ciertos tipos de datos o aplicaciones.

- **Datos desequilibrados:** puede ser un desafío en conjuntos de datos desequilibrados, donde las anomalías son la minoría. Conduciendo a bajas tasas de detección y altas tasas de falsos negativos.

Como **objetivos** de este tema nos planteamos:

- Introducir los métodos de detección de anomalías y su principal aplicación.

- Describir cómo se pueden utilizar métodos estadísticos y de aprendizaje automático para detectar anomalías.

8.2. Métodos estadísticos para la detección de anomalías

Para llevar a cabo la tarea de detección de anomalías, existen diversos métodos estadísticos que se pueden aplicar tanto a variables individuales (univariables) como a múltiples variables simultáneamente (multivariables).

Los **métodos estadísticos univariables** se centran en el análisis de una sola variable a la vez. Estos métodos incluyen técnicas como el cálculo de la media y la desviación estándar para identificar valores que se desvían significativamente del comportamiento esperado, el uso del Z-Score para estandarizar las puntuaciones y detectar *outliers,* así como la aplicación de la media móvil para observar tendencias y detectar irregularidades a lo largo del tiempo. La extensión de estos métodos a escenarios multivariables también es posible.

Este tipo de métodos se basa en el análisis de las propiedades estadísticas de los datos para identificar casos que se desvían significativamente de la norma. Es decir, que las anomalías tienen atributos estadísticos distintos en comparación con los puntos normales.

En el ámbito de la estadística, las pruebas son fundamentales para la toma de decisiones y la prueba de hipótesis. Se utilizan pruebas paramétricas y no paramétricas para discernir patrones y relaciones importantes en los datos, validar suposiciones o desacreditar ciertas afirmaciones.

- Las **no paramétricas** son herramientas eficaces para detectar anomalías en conjuntos de datos donde no se pueden hacer suposiciones sobre la distribución de los datos. Aprovechando estas técnicas, podemos detectar anomalías basadas en métricas como la media y los cuantiles.

- Las **paramétricas** se basan en el supuesto de que los datos siguen una distribución

específica, normalmente distribución normal o gaussiana. También asume otras condiciones como homogeneidad de varianzas e independencia de observaciones. Si se cumplen dichos criterios, las pruebas paramétricas son herramientas poderosas para detectar anomalías. Las técnicas paramétricas comunes son la prueba de Z-Score y el test de T-Score.

Veamos en detalle algunas de ellas.

Pruebas no paramétricas

Las pruebas no paramétricas son útiles para detectar anomalías especialmente cuando no se pueden hacer suposiciones sobre la distribución de los datos. A continuación, presentamos algunos métodos no paramétricos como el método del percentil y la media móvil. Existen otras pruebas no paramétricas comunes para la detección de anomalías: test de Grubbs (test de Grubbs para un outlier), test de Tukey (prueba de rango intercuartílico), test de Wilcoxon (test de Wilcoxon para muestras apareadas), test de Kruskal-Wallis y test de Mann-Whitney U.

Método del percentil

El método del percentil define un umbral con base en el rango percentil de los puntos de datos. Identificar anomalías como valores que están por debajo o por encima de un percentil específico es útil cuando se trata de distribuciones asimétricas no gaussianas.

Utilicemos este método con el mismo conjunto de datos del ejemplo anterior:

```
# Definir percentiles para marcar las anomalías
lower_percentile = 1
upper_percentile = 99

# Calcular los valores de los percentiles
lower_bound = np.percentile(data, lower_percentile)
upper_bound = np.percentile(data, upper_percentile)

# Identificar anomalías
anomalies = np.where((data < lower_bound) | (data > upper_bound))
```

```
# Mostrar los datos y marcar las anomalías
plt.figure(figsize=(10, 6))
plt.plot(data, 'bo', markersize=5, label='Dato normal')
plt.plot(anomalies[0], data[anomalies], 'ro', markersize=7,
label='Anomalía')
plt.axhline(y=lower_bound, color='g', linestyle='--',
label=f'{lower_percentile} Percentil')
plt.axhline(y=upper_bound, color='r', linestyle='--',
label=f'{upper_percentile} Percentil')
plt.title('Detección de anomalías utilizando Percentiles')
plt.legend()
plt.show()

# Imprimir las anomalías detectadas
print(f'Anomalías detectadas: {data[anomalies]}')
```

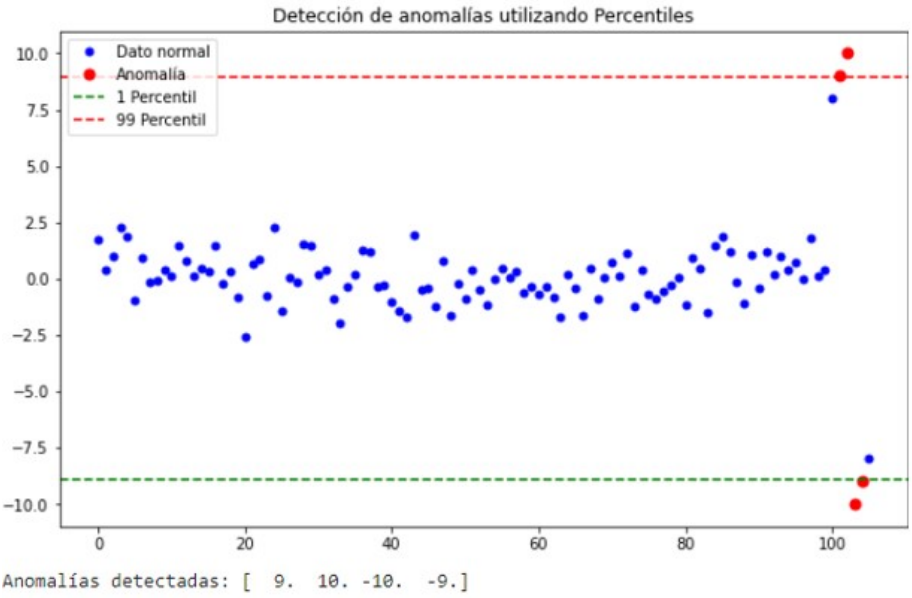


Figura 4. Detección de datos anómalos utilizando el método del percentil. Fuente: elaboración propia.

Método de la media móvil

Este método calcula la media móvil de los puntos de datos y compara cada dato con la media. Las anomalías suelen estar desviados significativamente del promedio móvil.

Veamos un ejemplo en Python:

```
import numpy as np
import matplotlib.pyplot as plt

# Generación de datos sintéticos de una serie temporal
np.random.seed(0)
data = np.random.normal(0, 1, 100) # 100 datos con distribución normal

# Añadir algunas anomalías
data[50] = 8
data[51] = -9
data[70] = 10
data[71] = -10

plt.figure(figsize=(10, 6))
plt.plot(data, 'bo-', markersize=5)
plt.title('Serie temporal con posibles anomalías')
plt.show()
```

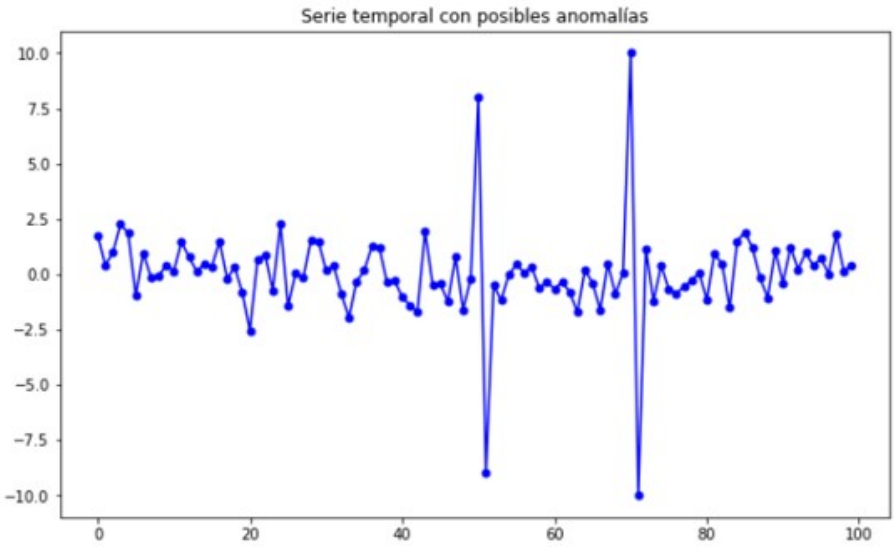


Figura 5. Ejemplo de serie temporal creada a partir de datos generados al azar. Fuente: elaboración propia.

```
# Definir la ventana para la media móvil
window_size = 10

# Calcular la media móvil
rolling_mean = np.convolve(data, np.ones(window_size)/window_size,
mode='valid')
```

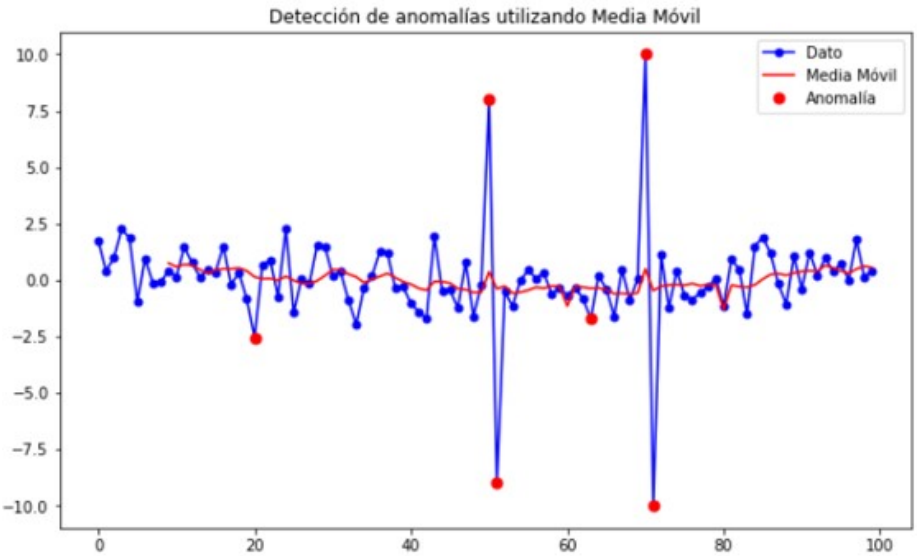
```
# Calcular la desviación estándar de la ventana móvil
rolling_std = np.array([np.std(data[i:i+window_size]) for i in
range(len(data) - window_size + 1)])

# Definir el umbral para considerar un punto como anomalía (por ejemplo, 2
desviaciones estándar)
threshold = 2

# Identificar anomalías
anomalies = []
for i in range(len(rolling_mean)):
    if np.abs(data[i + window_size - 1] - rolling_mean[i]) > threshold *
rolling_std[i]:
        anomalies.append(i + window_size - 1)

# Mostrar los datos, la media móvil y las anomalías
plt.figure(figsize=(10, 6))
plt.plot(data, 'bo-', markersize=5, label='Dato')
plt.plot(range(window_size - 1, len(data)), rolling_mean, 'r-',
label='Media Móvil')
plt.plot(anomalies, data[anomalies], 'ro', markersize=7, label='Anomalía')
plt.title('Detección de anomalías utilizando Media Móvil')
plt.legend()
plt.show()

# Imprimir las anomalías detectadas
print(f'Anomalías detectadas: {data[anomalies]}')
```




```
Anomalías detectadas: [ -2.55298982   8.         -9.         -1.7262826   10.
 -10.        ]
```

Figura 6. Ejemplo de detección de anomalía en una serie temporal utilizando la técnica de media móvil.

Fuente: elaboración propia.

Pruebas paramétricas

Estas técnicas se basan en parámetros estadísticos como la media y la desviación estándar para identificar puntos de datos que se desvían significativamente de la norma. Cada prueba actúa sobre un escenario diferente, por ejemplo, Z-Score y T-Score se utilizan para comparar las medias de uno o dos grupos. Mientras que ANOVA se utiliza para comparar las medias de tres o más grupos.

Z-Score

El Z-Score es una técnica estadística que nos permite detectar anomalías. Una vez calculado el valor de Z para cada punto de datos, aquellos que superen un umbral de x desviaciones estándar que se alejen de la media, se consideran anomalías.

Veamos un ejemplo donde el umbral está en 3 desviaciones estándar:

```
import numpy as np
import matplotlib.pyplot as plt

# Generación de datos sintéticos
np.random.seed(0)
data = np.random.normal(0, 1, 100) # 100 datos con distribución normal

# Añadir algunas anomalías
data = np.append(data, [8, 9, 10, -10, -9, -8])

plt.figure(figsize=(10, 6))
plt.plot(data, 'bo', markersize=5)
plt.title('Datos con posibles anomalías')
plt.show()
```

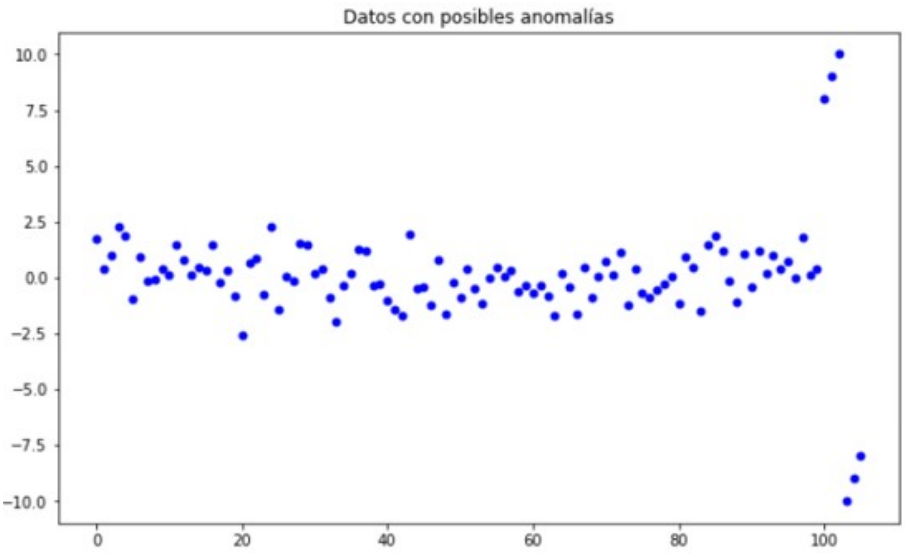


Figura 7. Diagrama de dispersión de datos generados aleatoriamente. Fuente: elaboración propia.

```
from scipy import stats

# Calcular la media y desviación estándar
mean = np.mean(data)
std_dev = np.std(data)

# Calcular el Z-Score
z_scores = stats.zscore(data)

# Determinar los umbrales para marcar anomalías
threshold = 3
anomalies = np.where(np.abs(z_scores) > threshold)

# Mostrar los datos y marcar las anomalías
plt.figure(figsize=(10, 6))
plt.plot(data, 'bo', markersize=5, label='Dato normal')
plt.plot(anomalies[0], data[anomalies], 'ro', markersize=7,
label='Anomalía')
plt.title('Detección de anomalías utilizando Z-Score')
plt.legend()
plt.show()

# Imprimir las anomalías detectadas
print(f'Anomalías detectadas: {data[anomalies]}')
```

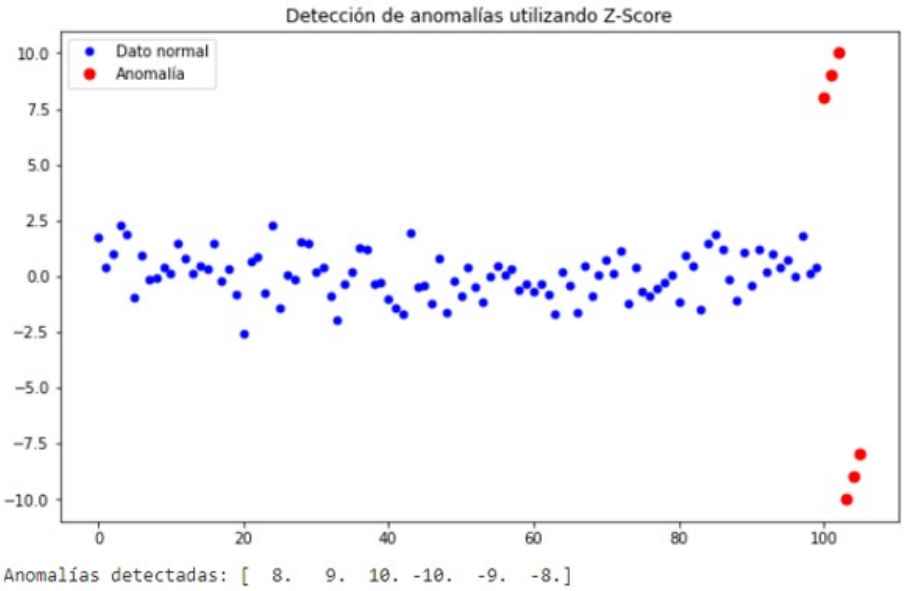


Figura 8. Detección de datos anómalos utilizando DBSCAN. Fuente: elaboración propia.

T-Score

T-Score es similar al Z-Score, pero el T-Score se utiliza principalmente cuando el tamaño de la muestra es pequeño y la desviación estándar de la población no es conocida.

Mostramos un ejemplo en Python utilizando datos sintéticos:

```
import numpy as np
import pandas as pd
from scipy import stats

# Generar datos de ejemplo
np.random.seed(0)
data = np.random.normal(50, 10, 30) # 30 datos con media 50 y desviación
estándar 10
data = np.append(data, [100, 110, 120]) # Añadir algunas anomalías

# Convertir a DataFrame para mayor comodidad
df = pd.DataFrame(data, columns=['Valor'])

# Calcular el T-Score
t_scores = stats.t.sf(np.abs(stats.t.cdf((df['Valor'] - df['Valor'].mean())
```

```
/ df['Valor'].std(), len(df)-1)), len(df)-1)

# Añadir el T-Score al DataFrame
df['T-Score'] = (df['Valor'] - df['Valor'].mean()) /
df['Valor'].std(ddof=1)

# Identificar las anomalías (por ejemplo, T-Score > 3 o T-Score < -3)
df['Anomalía'] = df['T-Score'].apply(lambda x: 'Sí' if np.abs(x) > 3 else
'No')

# Mostrar resultados
print(df)

# Si quieres mostrar solo las anomalías
print("\nAnomalías detectadas:")
print(df[df['Anomalía'] == 'Sí'])
```

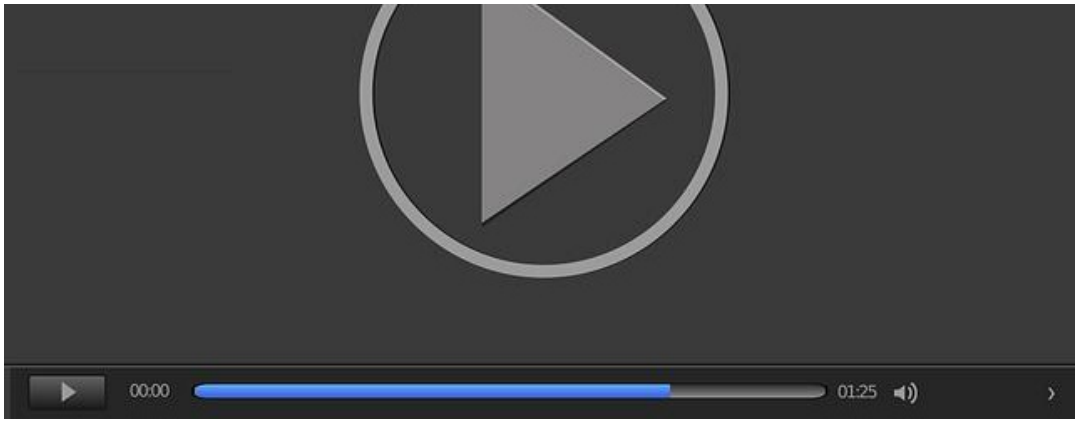
```
         Valor   T-Score Anomalía
0    67.640523  0.419058       No
1    54.001572 -0.281371       No
2    59.787380  0.015759       No
3    72.408932  0.663940       No
4    68.675580  0.472213       No
5    40.227221 -0.988754       No
6    59.500884  0.001046       No
7    48.486428 -0.564602       No
8    48.967811 -0.539881       No
9    54.105985 -0.276009       No
10   51.440436 -0.412899       No
11   64.542735  0.259971       No
12   57.610377 -0.096041       No
13   51.216750 -0.424386       No
14   54.438632 -0.258926       No
15   53.336743 -0.315514       No
16   64.940791  0.280413       No
17   47.948417 -0.592232       No
18   53.130677 -0.326096       No
19   41.459043 -0.925494       No
20   24.470102 -1.797963       No
21   56.536186 -0.151206       No
22   58.644362 -0.042941       No
23   42.578350 -0.868012       No
24   72.697546  0.678762       No
25   35.456343 -1.233763       No
26   50.457585 -0.463373       No
27   48.128161 -0.583001       No
28   65.327792  0.300287       No
29   64.693588  0.267718       No
30  100.000000  2.080882       No
31  110.000000  2.594432       No
32  120.000000  3.107983       Sí

Anomalías detectadas:
    Valor   T-Score Anomalía
32  120.0  3.107983       Sí
```

Figura 9. Detección de datos anómalos utilizando T-Score. Fuente: elaboración propia.

A continuación, veremos el vídeo ***Métodos estadísticos utilizados para detectar anomalías.***

Ideas clave



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=3635fb20-9410-4cc7-8157-b1bf001bf30b

8.3. Algoritmos basados en distancia

Los métodos basados en distancia detectan anomalías midiendo la distancia entre puntos de datos en el espacio de características. La idea principal es que los puntos de datos que están lejos de otros puntos se consideran anomalías.

Principios básicos

- **Distancia euclidiana:** a menudo se usa para medir la distancia entre puntos en el espacio de características.
- **Distancia al vecino más cercano:** los puntos de datos que tienen una distancia mayor que un umbral a su vecino más cercano se consideran anomalías.
- **Distancia media a k-vecinos más cercanos:** similar al anterior, pero considera la distancia media a los k-vecinos más cercanos para determinar si un punto es una anomalía.

Métodos comunes basados en distancia

k-Nearest Neighbors (k-NN) para detección de anomalías

- Detecta anomalías basándose en la distancia a los k vecinos más cercanos.
- Cómo funciona:
  - **Calcular distancias:** calcula la distancia de cada punto a sus k vecinos más cercanos.
  - **Determinar umbral:** define un umbral de distancia.
  - **Detección de anomalías:** los puntos cuya distancia a sus k vecinos más cercanos excede el umbral se consideran anomalías.

Distance-Based Outlier Detection (DBOD)

- Detecta anomalías considerando la distancia a un conjunto de vecinos.
- Cómo funciona:
  - **Definir vecinos:** para cada punto, encuentra sus k vecinos más cercanos.
  - **Calcular distancia:** calcula la distancia al vecino más cercano y a otros vecinos.
  - **Detección de anomalías:** los puntos con las mayores distancias son considerados anomalías.

Distancia de Mahalanobis

- Utiliza la distancia de Mahalanobis para detectar anomalías en datos multivariados considerando la correlación entre variables.
- Cómo funciona:
  - **Calcular la distancia:** usa la media y la covarianza de los datos para calcular la distancia de Mahalanobis.
  - **Detección de anomalías:** los puntos con una distancia de Mahalanobis alta se consideran anomalías.

Métodos de distancia basados en agrupamiento

- Usa algoritmos de *clustering* (como K-Means) para detectar anomalías basado en la distancia a los centroides de los clústeres.
- Cómo funciona:
  - **Aplicar *clustering:*** aplica un algoritmo de *clustering* para agrupar los datos.
  - **Calcular distancias:** calcula la distancia de cada punto al centroide de su clúster.
  - **Detección de anomalías:** los puntos con grandes distancias a sus centroides son considerados anomalías.

Ideas clave

8.4. Algoritmos basados en densidad

Los algoritmos basados en densidad identifican anomalías detectando puntos en el espacio de datos que están en regiones de baja densidad. Los puntos de datos que no tienen suficientes vecinos cercanos o están significativamente alejados de otros puntos se consideran anomalías. Tienen en cuenta lo siguiente:

- La densidad local de un punto se calcula considerando su proximidad a otros puntos. Los puntos en áreas de baja densidad se consideran anomalías.
- Los puntos que tienen una densidad significativamente menor que la densidad media de sus vecinos se consideran anomalías.

Los algoritmos más comunes son: Local Outlier Factor (LOF), Density-Based Spatial Clustering of Applications with Noise (DBSCAN) y One-Class SVM con Kernel Gaussiano (RBF).

Veamos en detalle el DBSCAN y el LOF:

DBSCAN

Una simple analogía nos ayuda a entender cómo funciona DBSCAN para detectar anomalías. Imaginemos que tenemos un montón de datos dibujados en un mapa y queremos encontrar los que son raros o no encajan en el grupo.

Algoritmo

Vamos a ver el paso a paso del algoritmo:

- Comenzamos seleccionando aleatoriamente un punto de datos.
- Definimos un radio ϵ y un número mínimo de puntos.
- Verifiquemos los puntos vecinos. Examinemos los puntos de datos dentro del radio

definido alrededor del punto seleccionado.

- Si hay al menos tantos puntos como lo definimos inicialmente dentro de un radio épsilon, consideramos que el punto seleccionado y los puntos cercanos conforman un grupo.

- Ahora, se repite el proceso con cada punto dentro del grupo recién formado. Si se encuentran puntos adicionales los agregamos al grupo. Continuamos de forma iterativa, expandiendo el grupo hasta que no se puedan agregar más puntos.

- Identificación de valores atípicos o ruido. Cualquier punto que este fuera del grupo después del proceso de expansión se etiqueta como valor atípico o ruido.

> Imaginemos que estamos en un campo con un grupo de personas dispersas. Algunas personas están juntas y otras están solas. DBSCAN nos ayuda a identificar dos cosas (Hiraltalsaniya, 2023):
>
> **Grupos de jugadores:** es como si cogiésemos a una persona cualquiera, le ponemos un *hula hoop* imaginario simulando una distancia máxima y comprobamos cuántas personas más hay dentro de ese *hula hoop*. Si el número de personas es suficiente, se conformaría el grupo.
>
> **Jugadores solitarios:** una vez conformado el grupo, se elige una persona dentro del grupo y se le coloca un *hula hoop* y comprobamos si hay más personas dentro. Las personas que están dentro del *hula hoop* se agregan al grupo y el proceso se continúa hasta que no existan más personas para agregar.
>
> Ahora viene lo interesante, cualquier persona que no esté en un grupo, es un caso atípico o jugador en solitario.

A continuación, vamos a poner un ejemplo de implementación en Python:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

# Generación de datos sintéticos
centers = [[0, 0], [5, 5]]
X, labels_true = make_blobs(n_samples=300, centers=centers,
cluster_std=0.5, random_state=0)

# Añadir ruido (anomalías)
np.random.seed(0)
X = np.vstack([X, np.random.uniform(low=-5, high=10, size=(20, 2))])

# Escalar los datos
X = StandardScaler().fit_transform(X)

# Aplicación del algoritmo DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
labels = db.labels_

# Identificación de núcleos y ruido
core_samples_mask = np.zeros_like(labels, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
n_noise = list(labels).count(-1)
print(f'Número de clusters estimados: {n_clusters}')
print(f'Número de puntos de ruido: {n_noise}')

# Visualización de los resultados
unique_labels = set(labels)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1,
len(unique_labels))]

for k, col in zip(unique_labels, colors):
    if k == -1:
        # Negro para el ruido.
        col = [0, 0, 0, 1]
    class_member_mask = (labels == k)
    xy = X[class_member_mask &amp;amp; core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
markeredgecolor='k', markersize=14)
    xy = X[class_member_mask &amp;amp; ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
markeredgecolor='k', markersize=6)
plt.title('Clusters detectados por DBSCAN')
```

```
plt.show()
```

```
Número de clusters estimados: 2
Número de puntos de ruido: 18
```

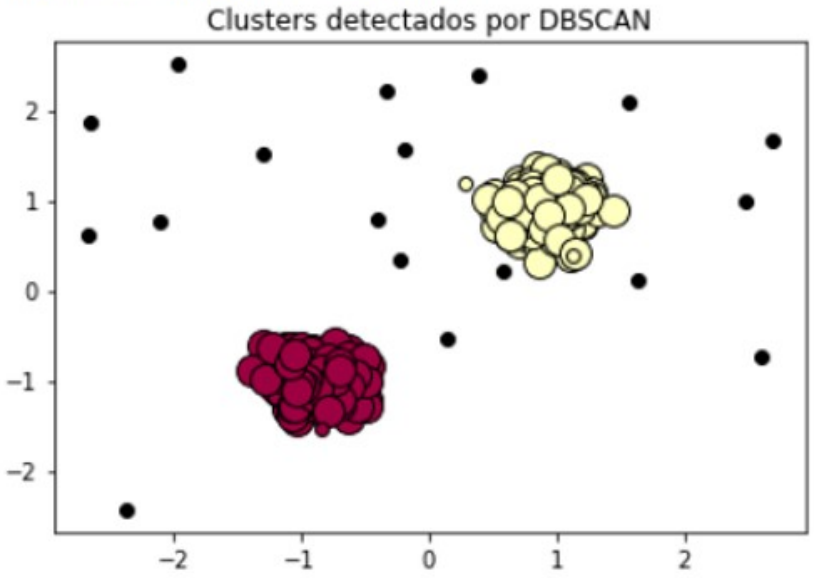
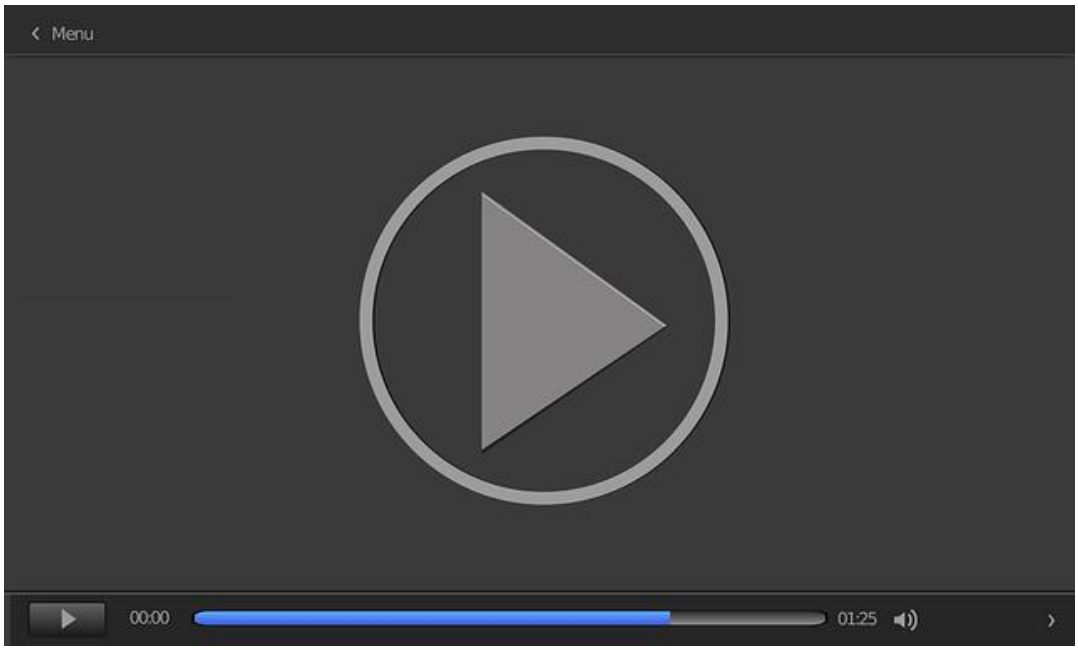


Figura 10. Detección de datos anómalos utilizando DBSCAN. Fuente: elaboración propia.

A continuación, veremos el vídeo ***DBSCAN para detectar anomalías.***



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=0f32b4f1-1a0f-4a0a-9e9f-b1bf00257807

Local Outlier Factor (LOF)

El LOF mide la densidad local de un punto en comparación con la densidad de sus vecinos más cercanos. Detecta anomalías en datos de alta dimensión y en clústeres de diferentes densidades.

¿Cómo funciona?

- **Distancia k-vecino más cercano:** para cada punto, se calcula la distancia al k-ésimo vecino más cercano.
- **Densidad local:** se calcula la densidad local de un punto como la inversa de la distancia promedio a sus k vecinos más cercanos.
- **Factor de outlier local (LOF):** compara la densidad local del punto con la densidad local de sus vecinos. Un LOF significativamente mayor que 1 indica una anomalía.

Ejemplo en Python

```
from sklearn.neighbors import LocalOutlierFactor
import numpy as np

# Generar datos de ejemplo
np.random.seed(0)
data = np.random.normal(50, 10, (100, 2))

# Añadir algunas anomalías
data = np.append(data, [[100, 100], [110, 110], [120, 120]], axis=0)

# Aplicar LOF para detección de anomalías
lof = LocalOutlierFactor(n_neighbors=20)
outlier_labels = lof.fit_predict(data)

# Los puntos con etiqueta -1 son anomalías
anomalías = data[outlier_labels == -1]
```

```
print(anomalías)
```

```
[[ 24.47010184  56.53618595]
 [ 72.69754624  35.45634325]
 [ 73.83144775  59.44479487]
 [ 71.63235949  63.36527949]
 [100.         100.        ]
 [110.         110.        ]
 [120.         120.        ]]
```

Figura 11. Detección de datos anómalos utilizando LOF. Fuente: elaboración propia.

8.5. Isolation Forest

Existen otros métodos como el Isolation Forest que no están basados ni en distancia ni en densidad. Es un algoritmo de aprendizaje automático no supervisado. Como su nombre lo indica, es un método basado en árboles de decisión, similar al Random Forest.

La idea principal del Isolation Forest se basa en que las anomalías son datos «poco frecuentes» y «diferentes» que se aíslan fácilmente. Es decir, se necesitan pocas particiones para aislar una anomalía (Liu, Ting y Zhou, 2008).

Algoritmo

Supongamos que tenemos los siguientes puntos:

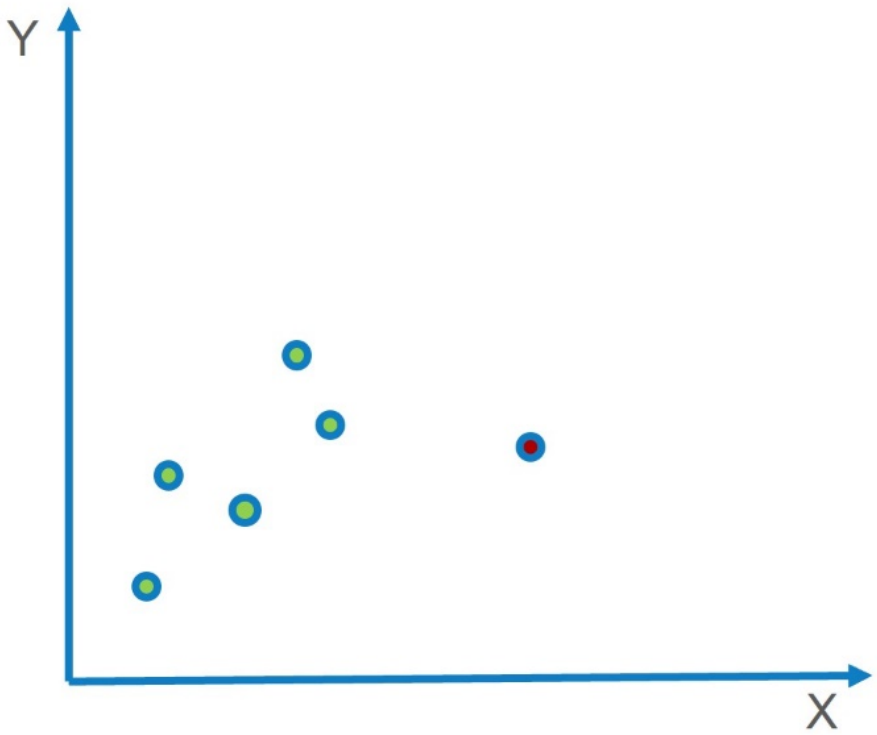


Figura 12. Paso 1, algoritmo Isolation Forest. Fuente: elaboración propia.

Se crean múltiples árboles de aislamiento *(isolation trees)*. Y en cada uno de los árboles se selecciona aleatoriamente una característica y un valor de corte para particionar el espacio.

Por ejemplo, un primer árbol selecciona una dimensión de forma aleatoria y se dividen aleatoriamente los datos a lo largo de la dimensión. En la Figura 13 se puede observar una división sobre el eje X.

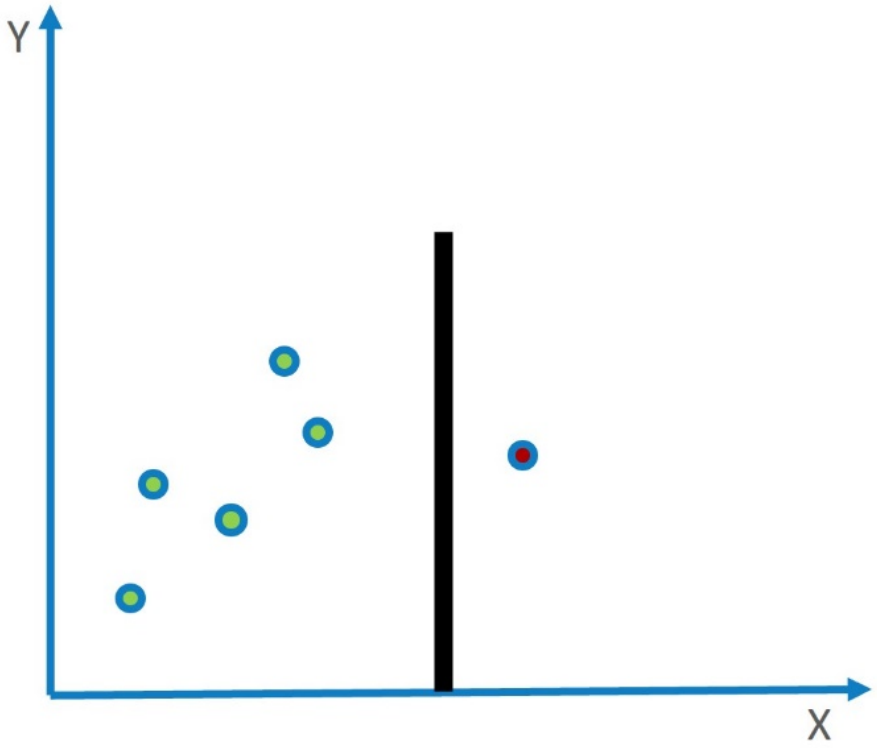


Figura 13. Selección aleatoria de dimensión y primera división de los datos. Fuente: elaboración propia.

Cada espacio define su propio subárbol. En el caso de la Figura 13, el corte separa un punto solitario del resto del conjunto. Como se muestra en la Figura 14.

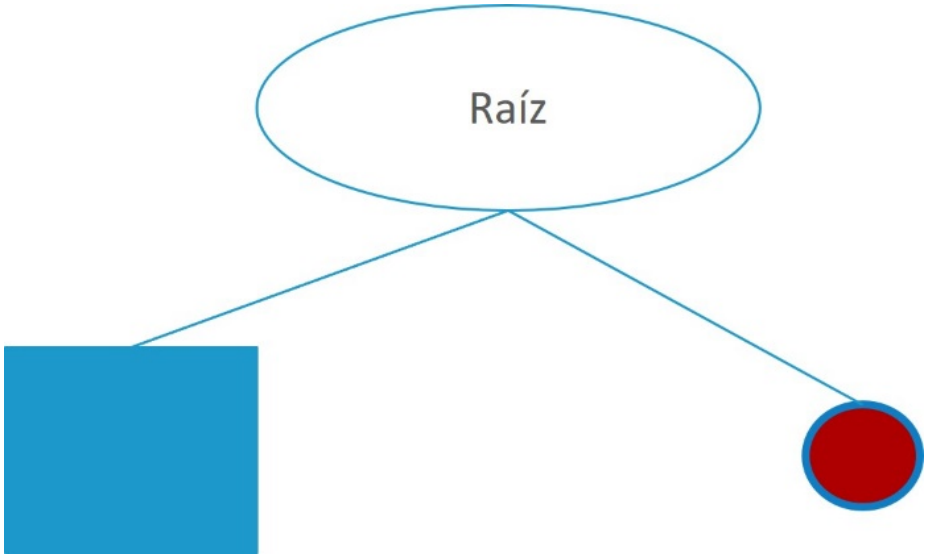


Figura 14. Subárbol generado a partir de la primera división sobre el eje X de los datos. Fuente: elaboración propia.

Veamos otro ejemplo de un segundo árbol donde se seleccionan divisiones diferentes. En la Figura 15 se puede ver como la primera división no aísla el valor atípico.

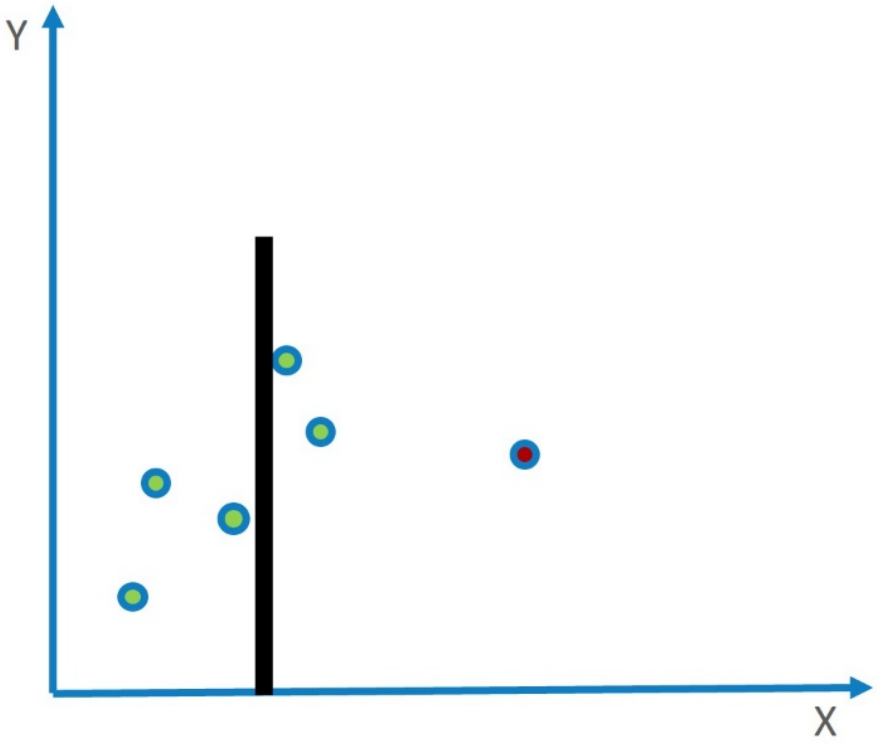


Figura 15. División de otro subárbol sobre el eje X. Fuente: elaboración propia.

En el segundo caso, el árbol tiene dos nodos, uno con los puntos de la izquierda y otro con los puntos de la derecha, como se muestra en la Figura 16.

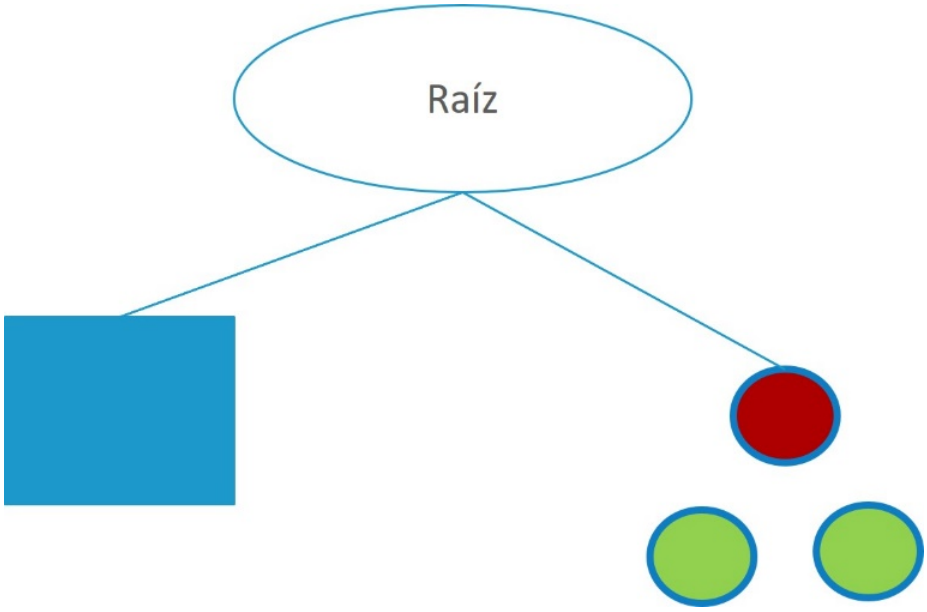


Figura 16. Segundo subárbol generado a partir de la división realizada en el eje X. Fuente: elaboración propia.

Este proceso se realiza recursivamente hasta que todos los puntos del conjunto de datos estén aislados o se llegue a la profundidad máxima establecida como parámetro.

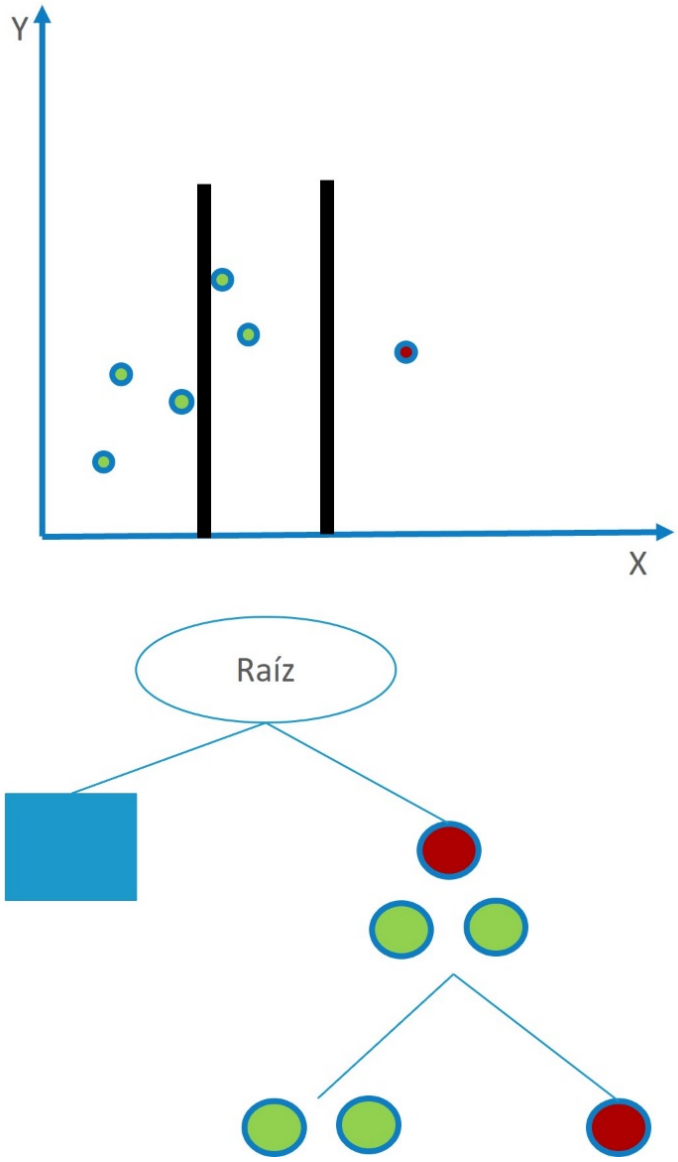


Figura 17. Segunda división realizada sobre el eje X y subárbol resultante. Fuente: elaboración propia.

Las instancias que quedan aisladas más rápidamente suelen ser anomalías, requieren menos particiones.

La profundidad promedio en las que una observación finaliza en todos los árboles es la que se utiliza para calcular la puntuación de una anomalía. Las puntuaciones bajas indican que la observación fue aislada rápidamente, y, por ende, es muy probable

que se trate de una anomalía. En la Figura 18 creamos un ejemplo hipotético donde analizamos una observación y contamos el número de divisiones en cada árbol antes del aislamiento definitivo, promediamos y ese es el puntuaje de anomalía que se le asigna.

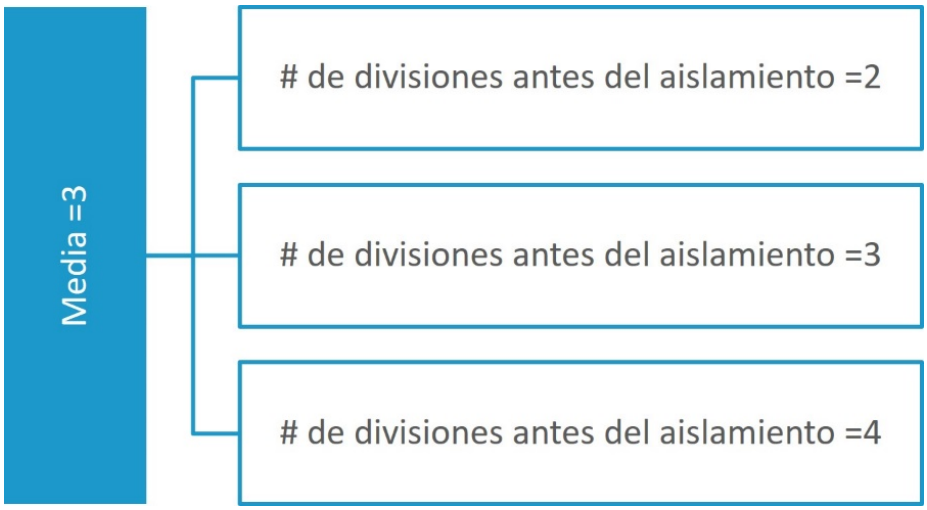


Figura 18. Promedio de divisiones de una observación en diferentes árboles. Fuente: elaboración propia.

Veamos un ejemplo de su implementación en Python:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import IsolationForest
import matplotlib.pyplot as plt

# Crear un DataFrame con datos sintéticos
np.random.seed(42)
data = np.random.normal(0, 1, 100).tolist() # Datos normales
data.extend([10, 15, -10, -15]) # Añadir anomalías

df = pd.DataFrame(data, columns=['value'])

# Visualizar los datos
plt.figure(figsize=(10, 6))
plt.plot(df['value'], label='Value')
plt.title('Datos Sintéticos con Anomalías')
plt.xlabel('Índice')
plt.ylabel('Valor')
plt.legend()
```

```
plt.show()

# Ajustar el modelo Isolation Forest
model = IsolationForest(contamination=0.1, random_state=42) # Ajustar el
parámetro contamination según tus necesidades
df['anomaly_score'] = model.fit_predict(df[['value']])

# Filtrar anomalías
df['anomaly'] = df['anomaly_score'] == -1

# Mostrar los resultados
print(df)

# Visualizar resultados
plt.figure(figsize=(10, 6))
plt.plot(df['value'], label='Value')
plt.scatter(df[df['anomaly']].index, df[df['anomaly']]['value'],
color='red', label='Anomaly')
plt.title('Detección de Anomalías usando Isolation Forest')
plt.xlabel('Índice')
plt.ylabel('Valor')
plt.legend()
plt.show().
```

8.6. Cuaderno de ejercicios

Ejercicio 1. Detección de anomalías con Z-Score

El Z-Score es una técnica estadística utilizada para identificar anomalías. Esta técnica calcula cuántas desviaciones estándar se aleja un punto de datos de la media del conjunto de datos. Con los siguientes datos sintéticos generados en Python responde a las preguntas planteadas.

```
import numpy as np

# Generación de datos sintéticos
np.random.seed(42)
data = np.random.normal(0, 1, 100) # 100 datos con distribución normal
data = np.append(data, [8, -8, 10, -10]) # Añadir algunas anomalías
```

- Calcula el Z-Score para cada punto de datos en el conjunto.
- Identifica los puntos de datos que se consideran anomalías usando un umbral de 3 desviaciones estándar.
- ¿Cuáles son las anomalías detectadas y por qué se consideran como tales?

Solución

1. Calcular el Z-Score:

```
import numpy as np
from scipy import stats

# Generación de datos sintéticos
np.random.seed(42)
data = np.random.normal(0, 1, 100)
data = np.append(data, [8, -8, 10, -10])

# Calcular el Z-Score
z_scores = stats.zscore(data)
```

2. Identificar anomalías con umbral de 3 desviaciones estándar:

```
threshold = 3
anomalies = np.where(np.abs(z_scores) > threshold)
```

3. Anomalías detectadas:

```
detected_anomalies = data[anomalies]
print(f'Anomalías detectadas: {detected_anomalies}')
```

Anomalías detectadas: [8. -8. 10. -10.].

Los valores que se encuentran a más de 3 desviaciones estándar de la media son considerados anomalías. En este caso, los valores 8, -8, 10 y -10 son detectados como anomalías.

Ejercicio 2. Detección de anomalías con media móvil

La media móvil es un método utilizado para analizar series temporales. Ayuda a suavizar las fluctuaciones de corto plazo y destacar las tendencias de largo plazo.

Con los datos sintéticos generados en Python responde a las siguientes preguntas:

- Calcula la media móvil para la serie temporal usando una ventana de tamaño 5.
- Identifica las anomalías en los datos comparando cada punto con la media móvil y usando un umbral de 2 desviaciones estándar.
- ¿Qué puntos de datos se consideran anomalías y cuál es la razón?

```
import numpy as np
import matplotlib.pyplot as plt

# Generación de datos sintéticos de una serie temporal
np.random.seed(0)
data = np.random.normal(0, 1, 100) # 100 datos con distribución normal
data[50] = 8 # Añadir anomalías
data[70] = -8

# Definir la ventana para la media móvil
window_size = 5
```

Solución

1. Calcular la media móvil para la serie temporal usando una ventana de tamaño 5.

```
import numpy as np
import matplotlib.pyplot as plt

# Generación de datos sintéticos de una serie temporal
np.random.seed(0)
data = np.random.normal(0, 1, 100)
data[50] = 8
data[70] = -8

# Definir la ventana para la media móvil
window_size = 5
rolling_mean = np.convolve(data, np.ones(window_size)/window_size,
mode='valid')
```

2. Identificar las anomalías en los datos comparando cada punto con la media móvil y usando un umbral de 2 desviaciones estándar.

```
rolling_std = np.array([np.std(data[i:i+window_size]) for i in
range(len(data) - window_size + 1)])
threshold = 2
anomalies = [i + window_size - 1 for i in range(len(rolling_mean)) if
np.abs(data[i + window_size - 1] - rolling_mean[i]) > threshold *
rolling_std[i]]
```

3. ¿Qué puntos de datos se consideran anomalías y cuál es la razón?

```
detected_anomalies = data[anomalies]
print(f'Anomalías detectadas: {detected_anomalies}')
```

Anomalías detectadas: [8. -8.].

Los puntos de datos que se desvían significativamente de la media móvil, más de 2 desviaciones estándar, son considerados anomalías. En este caso, los valores 8 y -8 son detectados como anomalías.

Ejercicio 3. Detección de anomalías con DBSCAN

Ideas clave

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) es un algoritmo de agrupamiento que identifica anomalías basándose en la densidad de puntos de datos.

Con los datos sintéticos generados en Python responde a las siguientes preguntas:

- ¿Cuántos clústeres se han detectado? ¿Cuántos puntos se consideran ruido (anomalías)?
- Explica cómo DBSCAN diferencia entre puntos de datos densamente agrupados y puntos aislados.
- Identifica y lista los puntos de datos considerados como anomalías.

```
from sklearn.datasets import make_blobs
from sklearn.cluster import DBSCAN
import numpy as np

# Generación de datos sintéticos
centers = [[0, 0], [5, 5]]
X, _ = make_blobs(n_samples=300, centers=centers, cluster_std=0.5,
random_state=0)
X = np.vstack([X, np.random.uniform(low=-5, high=10, size=(20, 2))]) #
Añadir ruido

# Aplicación del algoritmo DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
labels = db.labels_
```

Solución

1. ¿Cuántos clústeres se han detectado? ¿Cuántos puntos se consideran ruido (anomalías)?

```
from sklearn.datasets import make_blobs
from sklearn.cluster import DBSCAN
import numpy as np

# Generación de datos sintéticos
centers = [[0, 0], [5, 5]]
```

```
X, _ = make_blobs(n_samples=300, centers=centers, cluster_std=0.5,
random_state=0)
X = np.vstack([X, np.random.uniform(low=-5, high=10, size=(20, 2))])

# Aplicación del algoritmo DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
labels = db.labels_
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
n_noise = list(labels).count(-1)
```

2. Explicar cómo DBSCAN diferencia entre puntos de datos densamente agrupados y puntos aislados.

```
print(f'Número de clústeres estimados: {n_clusters}')
print(f'Número de puntos de ruido: {n_noise}')
```

Número de clústeres estimados: 2.

Número de puntos de ruido: 20.

DBSCAN detecta 2 clústeres y 20 puntos de ruido. Los puntos de ruido son considerados anomalías, ya que no pertenecen a ningún clúster.

3. Identificar y listar los puntos de datos considerados como anomalías.

```
anomalies = X[labels == -1]
print(f'Anomalías detectadas: {anomalies}')
```

Ejercicio 4. Detección de anomalías con Local Outlier Factor (LOF)

El Local Outlier Factor (LOF) mide la densidad local de un punto comparándola con la densidad de sus vecinos más cercanos.

Con los datos sintéticos generados en Python responde a las siguientes preguntas:

- ¿Cuántos puntos de datos se consideran anomalías según LOF?
- Explica brevemente cómo funciona el algoritmo LOF y cómo identifica anomalías.
- Lista las coordenadas de los puntos de datos que fueron detectados como

anomalías.

```
from sklearn.neighbors import LocalOutlierFactor
import numpy as np

# Generar datos de ejemplo
np.random.seed(42)
data = np.random.normal(50, 10, (100, 2))
data = np.append(data, [[100, 100], [110, 110], [120, 120]], axis=0) #
Añadir algunas anomalías

# Aplicar LOF para detección de anomalías
lof = LocalOutlierFactor(n_neighbors=20)
outlier_labels = lof.fit_predict(data)
```

Solución

1. ¿Cuántos puntos de datos se consideran anomalías según LOF?

```
from sklearn.neighbors import LocalOutlierFactor
import numpy as np

# Generar datos de ejemplo
np.random.seed(42)
data = np.random.normal(50, 10, (100, 2))
data = np.append(data, [[100, 100], [110, 110], [120, 120]], axis=0)

# Aplicar LOF para detección de anomalías
lof = LocalOutlierFactor(n_neighbors=20)
outlier_labels = lof.fit_predict(data)
```

2. Explicar brevemente cómo funciona el algoritmo LOF y cómo identifica anomalías.

```
anomalies = data[outlier_labels == -1]
print(f'Anomalías detectadas: {anomalies}')
```

Los puntos con una etiqueta de -1 son considerados anomalías por LOF debido a que tienen una densidad local significativamente menor que sus vecinos.

3. Listar las coordenadas de los puntos de datos que fueron detectados como anomalías.

Anomalías detectadas: [[100. 100.], [110. 110.] y [120. 120.]].

8.7. Referencias bibliográficas

Desai, U. (2023, abril 28). *Unsupervised machine learning with anomaly detection.* M e d i u m . https://utsavdesai26.medium.com/unsupervised-machine-learning-with-anomaly-detection-5fae4fd2c957

Hiraltalsaniya. (2023, diciembre 22). *Anomaly detection with unsupervised machine learning.* Medium. https://medium.com/simform-engineering/anomaly-detection-with-unsupervised-machine-learning-3bcf4c431aff

Indira. (2020, septiembre 25). *Machine Learning for Anomaly Detection- The Mathematics Behind It!* Medium. https://medium.com/srm-mic/machine-learning-for-anomaly-detection-the-mathematics-behind-it-7a2c3b5a755

Liu, F. T., Ting, K. M. y Zhou, Z. H. (2008). Isolation Forest. *Eighth IEEE International Conference on Data Mining,* 413-422. https://ieeexplore.ieee.org/document/4781136

¿Cuáles son los algoritmos más comunes de aprendizaje no supervisado para detectar anomalías?

LinkedIn. (s. f.). *¿Cuáles son los algoritmos de detección de anomalías no supervisados más comunes?* https://www.linkedin.com/advice/1/what-most-common-unsupervised-anomaly-detection-qqdvf

Describe las técnicas más comunes de detección de anomalías no supervisadas incluyen varios algoritmos eficaces y ampliamente utilizados.

Detección de anomalías en tiempo real

Moffitt, J. (2024, abril 1). Real-Time Anomaly Detection: Use Cases and Code Examples. Tinybird. https://www.tinybird.co/blog-posts/real-time-anomaly-detection

Se muestran algunos fragmentos de código de ejemplo que puede utilizar para inspirar su sistema de detección de anomalías en tiempo real.

Test

1. ¿Qué es una anomalía en el contexto del análisis de datos?

A. Un valor que es extremadamente común en el conjunto de datos.

B. Un valor que se ajusta perfectamente al modelo de datos.

C. Un valor que se desvía significativamente del comportamiento esperado del conjunto de datos.

D. Un valor que se encuentra dentro del rango intercuartílico.

2. ¿Cuál de los siguientes métodos se utiliza comúnmente para detectar anomalías en series temporales?

A. K-Means *clustering.*

B. PCA.

C. Media móvil.

D. Árboles de decisión.

3. ¿Qué mide el Z-Score en un conjunto de datos?

A. La distancia de un punto de datos al centroide más cercano.

B. El número de desviaciones estándar que un punto de datos se aleja de la media del conjunto de datos.

C. La densidad local de un punto de datos en comparación con sus vecinos más cercanos.

D. La correlación entre dos variables en un conjunto de datos.

4. ¿Cuál de los siguientes es un algoritmo basado en densidad para la detección de anomalías?

A. K-Nearest Neighbors (k-NN).

B. Support Vector Machines (SVM).

C. DBSCAN.

D. Árboles de regresión.

5. ¿Cuál es una desventaja común de los algoritmos de detección de anomalías?

A. No pueden manejar grandes volúmenes de datos.

B. Requieren siempre datos etiquetados.

C. Pueden generar falsos positivos.

D. No son compatibles con técnicas de *clustering.*

6. ¿Qué técnica estadística se basa en suponer que los datos siguen una distribución normal?

A. Método del percentil.

B. DBSCAN.

C. Z-Score.

D. LOF (Local Outlier Factor).

7. En el contexto de detección de anomalías, ¿qué es un *«outlier»*?

A. Un valor promedio en el conjunto de datos.

B. Un valor que está en la mediana del conjunto de datos.

C. Un valor que se encuentra dentro del rango intercuartílico.

D. Un valor que se desvía significativamente del resto de los datos.

8. ¿Qué tipo de algoritmo de detección de anomalías utiliza tanto datos etiquetados como no etiquetados?

A. Algoritmos supervisados.

B. Algoritmos no supervisados.

C. Algoritmos semisupervisados.

D. Algoritmos basados en reglas.

9. ¿Cuál de los siguientes métodos es más adecuado para detectar anomalías en datos multivariados considerando la correlación entre variables?

A. K-Means *clustering.*

B. Mahalanobis distance.

C. Media móvil.

D. Árboles de decisión.

10. ¿Cuál es una ventaja de la detección temprana de anomalías?

A. Reduce la necesidad de preprocesamiento de datos.

B. Mejora la precisión de los modelos sin eliminar valores atípicos.

C. Ayuda a identificar problemas potenciales antes de que se vuelvan críticos.

D. Elimina la necesidad de técnicas de aprendizaje supervisado.