Técnicas de Aprendizaje Automático

Tema 3. Datos ausentes y normalización

Índice

Esquema

3.1. Introducción y objetivos

La selección de atributos es una de las principales etapas para la creación de modelos de aprendizaje automático, puesto que, en la mayoría de los casos, los datos contienen más información de la necesaria. La selección de atributos no solo mejora la calidad del modelo, sino el uso de recursos computacionales, como el uso eficiente del almacenamiento y memoria durante el proceso de entrenamiento.

Adicional a la selección de atributos, se realizan **transformaciones**. La transformación de los datos resulta necesaria para su adecuación a los problemas de clasificación, dado que los datos no suelen encontrarse definidos bajo las mismas escalas numéricas y en ciertos casos siguen distribuciones diferentes. En este tema veremos diferentes técnicas de selección de atributos y de transformaciones que nos ayudarán a tener mejores modelos de aprendizaje automático.

- Identificar los conceptos básicos relacionados a valores ausentes y su importancia en el análisis de datos.
- Reconocer las técnicas y métodos más utilizados en la detección de datos redundantes.
- Conocer los métodos más empleados en la detección de valores duplicados en una base de datos.
- Determinar los diferentes mecanismos para la imputación de datos y su papel en el procesamiento y análisis de un conjunto de datos.
- Aplicar técnicas y métodos para la normalización de atributos de datos numéricos.

3.2. Encontrando atributos redundantes

En la minería de datos es necesario realizar la integración y preprocesamiento de la información para utilizarla dentro del modelo de entrenamiento. En primer lugar, se recomienda realizar la selección de los atributos que mejor representan los datos. Luego, se debe garantizar que los atributos del conjunto de datos no se deriven de cualquier otro atributo del conjunto de datos; esto es a lo que se denomina **«datos redundantes»**.

De este modo, la falta de visión global de los conjuntos de datos, la dispersión y la existencia de datos redundantes han originado la aparición de técnicas para la localización de tales atributos perdidos, por ejemplo, correlación y covarianza, y prueba X2 (Jain, 2023).

Correlación y covarianza

Tal como se definió en el tema anterior, el coeficiente de Pearson es un parámetro estadístico que nos permite establecer la asociación lineal entre dos variables cuantitativas y continuas (Flores, 2020). Este hecho puede ser utilizado para determinar redundancia en los atributos planteado de la siguiente manera: sean X1 y X2 atributos de un conjunto de datos, se puede afirmar que cuanto mayor es el coeficiente de correlación, más fuerte es la correlación de los atributos. Por lo tanto, es posible prescindir de uno de ellos (X1 o X2) debido a que la correlación perfecta implica redundancia en las variables. Si el coeficiente de correlación es 0, entonces los atributos son independientes (sugiere que no hay redundancia en los datos); si el coeficiente es negativo, entonces un atributo desalienta al otro, es decir, si el valor de un atributo aumenta, el valor del otro disminuye (Díaz, 2007). Un **ejemplo básico de redundancia** se presenta a continuación:

	nombre	es_hombre	es_mujer
1	angie	0	1
2	juan	1	0
3	diego	1	0
4	pedro	1	0
5	ana	0	1
6	jose	1	0
7	carolina	0	1

Figura 1. Datos de sexo de personas. Fuente: https://media.geeksforgeeks.org/wp-content/uploads/20190416182411/data-redundancy1.png

En la Figura 1 se aprecia un conjunto de datos que hace referencia a los datos personales de siete sujetos. El atributo es_hombre corresponde a 1 si la persona es un hombre, de lo contrario, es 0. El atributo es_mujer será 1 si la persona correspondiente es una mujer, de lo contrario, será 0. Analizando este hecho, si una persona no es hombre, entonces es una mujer, lo cual implica que los dos atributos están altamente correlacionados (100 %) y que uno determina al otro. Por tanto, uno de estos atributos se volvió redundante y, por consiguiente, uno de los dos se puede eliminar sin pérdida de información.

Prueba de datos nominales χ2

Cuando el tipo de datos de la característica que se va a probar y la variable de destino son categóricos (es decir, se tiene un problema de clasificación), podemos usar la **prueba Chi-cuadrado**, también conocida como chi-cuadrado de Pearson (Sakkaf, 2020).

Esta prueba se realiza sobre datos nominales. Siendo X_1 y X_2 dos atributos en un conjunto de datos, se elabora una tabla de contingencia para representar tuplas de datos. Ver ecuación (1):

$$\chi^2 = \sum\left[\left(Y_0 - Y_e\right)^2\right]/Y_e$$

Donde Y_o es el recuento real de los valores observados y Y_e , los valores esperados de los eventos conjuntos de la tabla de contingencia. El χ^2 comprueba la hipótesis de que X_1 y X_2 son independientes. Si se puede rechazar esta hipótesis, podemos decir que X_1 y X_2 están correlacionados estadísticamente y uno de ellos puede descartarse.

A continuación, se explica como ejemplo la aplicación de esta prueba a través de ocho pasos (ver Tabla 1) (Sakkaf, 2020):

Pasos	Concepto
1. Definir la hipótesis Para el ejemplo, se considera: **H0:** Las características Género e Interés son independientes (no están asociadas). **H1:** El género y el interés no son independientes (están asociados).	Una prueba de hipótesis es un método estadístico que evalúa dos afirmaciones (hipótesis) y determina qué afirmación es verdadera. En la prueba de hipótesis, el enunciado inicial se llama Hipótesis nula, denotada como H0, y la segunda, que generalmente es complementaria a la primera, se llama Hipótesis alternativa, denotada como H1.
2. Nivel de significación α (alfa) Para el ejemplo, se considera: α = 0,05	Es una medida que se usa para determinar si la hipótesis nula debe rechazarse o no. Para que la hipótesis nula sea rechazada, el valor p debe ser menor que el nivel de significancia. Generalmente se prefieren valores de α más bajos que pueden estar en el intervalo de 0,01 a 0,10.
3. Creación de tablas de contingencia o tablas cruzadas	Una tabla de contingencia es aquella que muestra la distribución de frecuencia de una variable en filas y otra en columnas, utilizada para estudiar la correlación entre las dos variables. Los valores de la tabla se denominan valores observados.
4. Cálculo de la frecuencia esperada	Se refiere al recuento de frecuencia esperado para cada celda. La fórmula para contar la frecuencia esperada es: $E = (total\ de\ la\ fila * total\ de\ la\ columna) / total general$
5. Cálculo del Chi-cuadrado χ2	Es la suma de la diferencia al cuadrado entre las frecuencias observadas y esperadas dividida por la frecuencia esperada para todas las celdas.
6. Cálculo de los grados de libertad	Se realiza a través de la siguiente fórmula: $df = (total\_rows - 1) * (total\_cols - 1)$
7. Valor de p	Puede ser calculado usando los valores de Chi Cuadrado y los grados de libertad.
8. Decidir si rechazar o mantener la hipótesis nula	Si el valor p es menor que el valor de significancia, se rechaza la Hipótesis nula y si el valor de p es mayor que el valor de significancia, no se rechaza.

Tabla 1. Pasos para aplicar la prueba de chi-cuadrado para la selección de características. Fuente: elaborado a partir de Sakkaf (2020).

Lo anteriormente mencionado en la Tabla 1 se puede presentar a través de Python usando la biblioteca *scipy*. A continuación, se ejecuta una prueba de chi-cuadrado para comprobar si existe alguna relación entre dos variables categóricas o no (Sakkaf, 2020):

```
from scipy.stats import chi2_contingency
```

```
tabla_contingencia= [[20, 30, 15], [20, 15, 30]]

stat, p, dof, expected = chi2_contingency(tabla_contingencia)

alfa= 0.05

print('Significancia=%.3f, p=%.3f' % (alfa, p))

if p <= alfa:

    print('Variables son asociadas (Rechazo de H0)')

else:

    print('Variables no son asociadas (Falló Rechazo de H0)')
```

Es posible obtener los parámetros chi-cuadrado, el valor p, los grados de libertad y las frecuencias esperadas a partir de la tabla de contingencias. El ejemplo toma un valor de significancia de $\alpha = 0{,}05$ y finalmente se determina si la hipótesis nula planteada es aceptada o rechazada. En la Figura 2 se muestran los datos que se imprimen en consola.

```
significance=0.050, p=0.007
Variables are associated (reject H0)
```

Figura 2. Salida de la consola. Fuente: elaboración propia.

3.3. Detectando registros duplicados

Cuando se realiza el registro de valores en bases de datos a partir de fuentes no estandarizadas, los datos presentan a menudo valores ausentes o valores duplicados. La información puede encontrarse consignada dos o más veces, lo cual afecta considerablemente la calidad de los resultados (Chavez, 2020). Se conocen como datos duplicados (*Record Linkage* en inglés) a aquellos cuyos valores coinciden en todas las variables de estudio seleccionadas por el analista de datos; esto se debe a diferentes causas como la entrada incorrecta de datos, la introducción repetida de un mismo valor o la corrupción de los datos (IBM, 2019).

Por datos duplicados no solo se hace referencia a una misma entidad, sino a los atributos o instancias que a pesar de tener contenidos diferentes deberían ser el mismo (Amón y Jiménez, 2009). Por consiguiente, la detección de registros duplicados pretende explorar en una o más fuentes los datos que pudiendo ser únicos, no lo son al presentar representaciones distintas (Guevara y Amón, 2016). Una base de datos que incluya nombres, direcciones u otros datos puede verse fácilmente afectada con entradas duplicadas (ver Figura 3), como resultado de la manipulación de varias personas o el ingreso de datos en diferentes momentos y bajo diferentes circunstancias. Dado lo anterior, un algoritmo resulta necesario para reunir todos los posibles registros duplicados y así fusionarlos o eliminar alguno de ellos (DeShon, 2016).

```
import pandas as pd

data = {'Id': [1, 2, 3, 4, 5, 6, 7, 7], 'Primer_nombre': ['Angela',
'Adrian', 'Theodoro', 'Angela', 'Adrian', 'Beatriz', 'Olivia', 'Olivia'],
'Primer_apellido': ['Castro', 'Guzman', 'Rivadeneira', 'Castillo', 'Casas',
'Perez', 'Apraez', 'Apraez'], 'Edad': [27, 31, 36, 27, 53, 48, 36, 36],
'Estatura': [1.67, 1.80, 1.61, 1.77, 1.88, 1.69, 1.62, 1.62]}

df = pd.DataFrame(data, columns = ['Id', 'Primer_nombre',
```

```
'Primer_apellido', 'Edad', 'Estatura'])

df
```

	Id	Primer_nombre	Primer_apellido	Edad	Estatura
0	1	Angela	Castro	27	1.67
1	2	Adrian	Guzman	31	1.80
2	3	Theodoro	Rivadeneira	36	1.61
3	4	Angela	Castillo	27	1.77
4	5	Adrian	Casas	53	1.88
5	6	Beatriz	Perez	48	1.69
6	7	Olivia	Apraez	36	1.62
7	7	Olivia	Apraez	36	1.62

Figura 3. Base de datos que incluye cuatro variables para siete registros de los cuales el Id 7 se encuentra duplicado. Fuente: elaborado a partir de Analytics Lane, 2018.

El conjunto de datos hace referencia a cuatro variables: Id (identificación), primer nombre, primer apellido, edad y estatura, correspondiente a siete sujetos. Se puede observar que los registros de la fila 6 y 7 son los mismos, por lo cual se encuentra duplicada. El método `duplicated()` del dataframe muestra esto en Python:

```
df.duplicated()
```

```
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7     True
dtype: bool
```

Figura 4. Identificación del número de fila que se encuentra duplicada en la base de datos. Fuente: elaborado a partir de Analytics Lane, 2018.

Una vez identificados los datos de la fila que se encuentran duplicados se hace uso del método drop_duplicates() de Python para eliminar estos registros, así:

```
df = df.drop_duplicates()
```

3.4. Mecanismos para reemplazar datos ausentes

Los valores ausentes son valores para atributos que no se introdujeron o se perdieron en el proceso de registro. Los valores ausentes dificultan la realización del análisis de datos y también puede plantear problemas graves a los investigadores. De hecho, el manejo inadecuado de los valores ausentes en el análisis puede introducir sesgos y dar lugar a conclusiones engañosas que se extraigan de un estudio de investigación; también puede limitar la generalización de los resultados de la investigación. Muchos conjuntos de datos industriales y de investigación contienen valores ausentes en sus valores de atributo. La presencia de tales imperfecciones generalmente requiere una etapa de preprocesamiento en la que los datos se preparan y limpian, para que sean útiles y suficientemente claros para el proceso de extracción de conocimiento (Pyle, 1999). Existen varias razones para su existencia, como procedimientos de entrada de datos manual, errores de equipo y mediciones incorrectas (Little y Rubin, 1987).

Es importante definir los mecanismos que conducen a la introducción de valores ausentes; los supuestos que se hacen sobre el mecanismo de pérdida de datos y el patrón de los valores ausentes pueden afectar a la elección del método de tratamiento que podría aplicarse correctamente, si es que se podría aplicar. Comprender las razones por las que se pierden datos es importante para manejar correctamente los datos restantes (Little y Rubin, 1987). Si faltan valores completamente al azar, es probable que la muestra de datos aún sea representativa de la población. Sin embargo, si los valores faltan sistemáticamente, el análisis puede estar sesgado. Por ejemplo, en un estudio de la relación entre el coeficiente intelectual y los ingresos, es posible que haya una tendencia de los participantes con un coeficiente intelectual superior al promedio a saltarse la pregunta sobre su salario. Los análisis que no tienen en cuenta esta pérdida aleatoria pueden fallar en encontrar una asociación positiva entre el coeficiente intelectual y el salario. El

manejo de los valores ausentes en aprendizaje automático puede abordarse a través de dos mecanismos diferentes: **eliminación de valores ausentes** e **imputación de valores ausentes con estimados,** como media, moda y mediana. También, cabe denotar que estos mecanismos son importantes, dado que muchos algoritmos no pueden manejar datos ausentes, por tanto, es importante en cualquiera de los casos identificar los valores ausentes de un conjunto de datos (Hand *et al.*, 2008).

Mecanismos de pérdida de datos

Cuando la probabilidad de que falte una observación puede depender de las variables observadas, pero no de las ausentes, podemos afirmar que los datos ausentes están **perdidos al azar** (MAR, por sus siglas en inglés — *Missing at Random*). En el caso del mecanismo de datos faltantes de MAR, dado un valor o valores particulares para un conjunto de características pertenecientes a las variables observadas, la distribución del resto de características es la misma entre los casos observados que entre los casos faltantes (Deng, 2012).

La **pérdida completamente al azar** (MCAR, por sus siglas en inglés — *Missing Completely at Random*) es un caso especial de MAR en el que la distribución de un ejemplo que tiene un valor ausente para un atributo no depende ni de los datos observados ni de los no observados (Deng, 2012). Los valores de los datos faltantes se constituyen como otra posible muestra de la distribución de probabilidad. Cuando los datos son MCAR, el análisis realizado sobre los datos no tiene sesgos; sin embargo, los datos rara vez son MCAR.

Un tercer caso surge cuando MAR no se aplica, ya que el valor ausente depende tanto del resto de valores observados como del valor propio en sí. Este modelo generalmente se denomina **pérdida NO al azar** (NMAR, por sus siglas en inglés — *Not missing at Random*) (Deng, 2012). Este modelo de ausencias es un desafío, ya que la única forma de obtener una estimación no sesgada es modelar también las ausencias. Se trata de una tarea muy compleja en la que se debe crear un modelo

que contabilice los datos faltantes que luego deberían incorporarse a un modelo más complejo utilizado para estimar los valores ausentes.

Identificar valores ausentes

La mayoría de los datos tienen valores perdidos, tal como se menciona a continuación: «Los datos ausentes no son datos extraños en conjuntos de datos reales. De hecho, la probabilidad de que falte al menos un dato aumenta a medida que aumenta el tamaño del conjunto de datos» (Kuhn y Johnson, 2019).

Antes de describir algunos métodos para identificar valores ausentes, es necesario hablar de algunos formatos como un archivo de valores separados por comas (CSV, del inglés *Comma-Separated Values*). Para ejemplificar un poco mejor cargaremos la base de datos Pima Indians Diabetes.

```
# Carga la biblioteca

from pandas import read_csv

# Función que permite leer un archivo CSV y asignarlo a un Dataframe

dataset = read_csv('pima-indians-diabetes.csv', header=None)

# Resumen de los datos

print(dataset.describe())
```

Ejecutando las líneas de código anteriores se obtiene la siguiente salida:

	0	1	2	...	6	7	8
count	768.000000	768.000000	768.000000	...	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	...	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	...	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	...	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	...	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	...	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	...	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	...	2.420000	81.000000	1.000000

Figura 5. Resumen del conjunto de datos Pima Indians Diabetes. Fuente: elaboración propia.

En la Figura 5, se observa que hay atributos que tienen un valor mínimo de cero 0. Este valor no tiene sentido de acuerdo con los atributos, por lo que se puede deducir que **posee un valor no válido o faltante**. No solo el valor de 0, con frecuencia se puede indicar con datos fuera de rango; quizás un número negativo, en un campo numérico que normalmente es solo positivo, o simplemente se identifican por la etiqueta 'NaN'. En específico, los atributos que contienen cero en el valor mínimo son: concentración de glucosa, presión sanguínea [mm Hg], espesor del pliegue cutáneo del tríceps [mm], insulina sérica [mu U / ml] e índice de masa corporal; tal como se muestra en la Figura 6.

```
# Imprimir los 20 primeros datos

print(dataset.head(20))
```

	0	1	2	3	4	5	6	7	8
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1
10	4	110	92	0	0	37.6	0.191	30	0
11	10	168	74	0	0	38.0	0.537	34	1
12	10	139	80	0	0	27.1	1.441	57	0
13	1	189	60	23	846	30.1	0.398	59	1
14	5	166	72	19	175	25.8	0.587	51	1
15	7	100	0	0	0	30.0	0.484	32	1
16	0	118	84	47	230	45.8	0.551	31	1
17	7	107	74	0	0	29.6	0.254	31	1
18	1	103	30	38	83	43.3	0.183	33	0
19	1	115	70	30	96	34.6	0.529	32	1

Figura 6. Primeros 20 datos del conjunto de datos Pima Indians Diabetes. Fuente: elaboración propia.

Cabe resaltar que algunos de los atributos pueden tener valor 0 en sus datos (por ejemplo: número de embarazos), por ende, es importante diferenciarlos, y en lugar de ello usar la etiqueta 'NaN' agregando el comando `replace(0, 'nan')` .

```
# Reemplaza 0 por 'nan'

dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, 'nan')

# Imprimir los 20 primeros datos

print(dataset.head(20))
```

	0	1	2	3	4	5	6	7	8
0	6	148	72	35	nan	33.6	0.627	50	1
1	1	85	66	29	nan	26.6	0.351	31	0
2	8	183	64	nan	nan	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	nan	nan	25.6	0.201	30	0
6	3	78	50	32	88	31	0.248	26	1
7	10	115	nan	nan	nan	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	nan	nan	nan	0.232	54	1
10	4	110	92	nan	nan	37.6	0.191	30	0
11	10	168	74	nan	nan	38	0.537	34	1
12	10	139	80	nan	nan	27.1	1.441	57	0
13	1	189	60	23	846	30.1	0.398	59	1
14	5	166	72	19	175	25.8	0.587	51	1
15	7	100	nan	nan	nan	30	0.484	32	1
16	0	118	84	47	230	45.8	0.551	31	1
17	7	107	74	nan	nan	29.6	0.254	31	1
18	1	103	30	38	83	43.3	0.183	33	0
19	1	115	70	30	96	34.6	0.529	32	1

Figura 7. Primeros 20 datos con reemplazo del conjunto de datos Pima Indians Diabetes. Fuente: elaboración propia.

Y finalmente, mediante la función `dataset.isnull().sum()` permite conocer el número total de valores ausentes por cada uno de los atributos.

```
0      0
1      5
2     35
3    227
4    374
5     11
6      0
7      0
8      0
dtype: int64
```

Figura 8. Conteo de valores ausentes del conjunto de datos Pima Indians Diabetes. Fuente: elaboración propia.

Mecanismo de imputación por eliminación

Una de las maneras más simples para tratar datos ausentes es eliminar por completo cada uno de los atributos «columnas» o instancias «filas» del conjunto de datos (Brownlee, 2020; Raschka, 2015).

```
# Resumen de los datos originales

print(dataset.shape)

# Eliminar filas con valores ausentes

dataset.dropna(inplace=True)

# Resumen de los datos luego de la eliminación

print(dataset.shape)
```

Para este ejemplo existe una eliminación del 48,9 % de los datos, lo que indicaría que no es viable. Estos métodos son prácticos solo cuando los datos contienen un número relativamente pequeño de ejemplos con valores ausentes y cuando el análisis de los ejemplos completos no conducirá a sesgos graves durante la inferencia y puede tener un efecto sobre las métricas de rendimiento de los algoritmos.

Mecanismo de imputación por media, mediana y moda

De manera simple, se pueden utilizar algunas técnicas de interpolación para estimar valores ausentes a partir de otras instancias de un conjunto de datos. Una de las técnicas de interpolación más conocidas es la **imputación por media**, **mediana y moda**, que consiste en sustituir el valor ausente por el valor estadístico de toda la columna de atributos (Raschka, 2015). En Python puede usarse la siguiente

codificación con el fin de obtener los resultados requeridos.

```
# Biblioteca para usar datos con nan

from numpy import nan

from numpy import isnan

# Se usa arreglo de datos de numpy

values = dataset.values

# Define imputador

imputer = SimpleImputer(missing_values=nan, strategy='mean')
```

Transformar el conjunto de datos

```
transformed_values = imputer.fit_transform(values)

# Realiza el conteo total de valores ausentes

print('Valores ausentes: %d' % isnan(transformed_values).sum())
```

Con el fin de configurar el parámetro estadístico basta con cambiar la estrategia del imputador 'mean' SimpleImputer(missing_values=nan, strategy='mean') por 'median' o 'moda'. Finalmente, mediante la función print('Valores ausentes: %d' % isnan(transformed_values).sum()) es posible comprobar que el reemplazo ha sido efectivo, dado que su valor es cero.

3.5. Otros métodos de imputación de valores ausentes

Existe una amplia familia de métodos de imputación, desde técnicas simples de imputación como sustitución con la media, como K-Nearest Neighbour, hasta los que analizan las relaciones entre atributos, como basado en Máquinas de Vectores de Soporte, basado en clustering, regresiones logísticas, procedimientos de máxima verosimilitud e imputación múltiple (Batista y Monard, 2003). Las técnicas de imputación basadas en el muestreo de distribuciones de datos estimadas se dividen entre procedimientos de imputación simple, como los procedimientos de maximización de expectativas, y procedimientos de imputación múltiple que son más fiables y potentes, pero más difíciles y restrictivos de aplicar (Schaffer, 1987).

El manejo de valores ausentes es independiente del algoritmo de aprendizaje utilizado. Por este motivo, el usuario puede seleccionar el método más adecuado a cada situación que afronta (Batista y Monard, 2003). Los procedimientos de imputación se hicieron muy populares para los datos cuantitativos, por lo que fueron fácilmente adoptados en otros campos del conocimiento, como la bioinformática, la ciencia climática, la medicina, etc. Los métodos de imputación propuestos en cada campo se adaptan a las características comunes de los datos analizados en él.

E l **algoritmo de esperanza-maximización** estima los parámetros de una distribución de probabilidad. Para nuestro caso, esto se puede lograr a partir de datos incompletos. De forma iterativa maximiza la verosimilitud de los datos observados completos considerados como una función dependiente de los parámetros. Para usar EM para la imputación, primero se debe elegir un conjunto plausible de parámetros; es decir, debemos asumir que los datos siguen una distribución de probabilidad.

Ideas clave

El algoritmo EM funciona mejor con distribuciones de probabilidad que son fáciles de maximizar, como modelos de mezcla gaussiana (Dempster *et al.*, 1977).

Los métodos de máxima verosimilitud como EM tienden a subestimar los errores inherentes producidos por el proceso de estimación, llamados formalmente **errores estándar**. El **método de Imputación Múltiple** (MI, por sus siglas en inglés — *Multiple Imputation*) fue diseñado para tener esto en cuenta y ser un método de imputación menos sesgado, a costa de ser computacionalmente costoso. MI genera múltiples valores imputados a partir de los datos observados; llena los datos incompletos mediante la resolución repetida de los datos observados. Mientras que EM genera una sola imputación a partir de los parámetros estimados en cada paso, MI realiza varias imputaciones que producen varios conjuntos de datos completos. Esta imputación repetida se puede realizar gracias al uso de los **métodos de Markov Chain Monte Carlo**, ya que las diversas imputaciones se obtienen introduciendo un componente aleatorio, generalmente a partir de una distribución normal estándar (Scheuren, 2005).

3.6. Normalización Min-Max

En la mayoría de los casos, los conjuntos de datos contienen atributos que varían altamente en magnitudes, unidades y rango. Estos atributos tienen un significado en el dominio original de donde se obtuvieron; sin embargo, algunos algoritmos de aprendizaje automático utilizan la distancia euclidiana entre atributos de datos ignorando las unidades, para clasificar las nuevas instancias de un conjunto de datos. Los atributos con magnitudes altas pesarán mucho más en los cálculos de distancia que las características con magnitudes bajas, siendo necesario realizar una serie de pasos de manipulación que transforman los atributos originales o generan nuevos atributos con mejores propiedades que ayuden al poder predictivo del modelo. Llevar todas las características al mismo nivel de magnitudes se puede lograr mediante el ajuste por escalado de características, y los nuevos atributos generalmente se denominan **variables de modelado** o **variables analíticas** (García *et al.*, 2015).

E l **ajuste por escalado de atributos** es uno de los pasos críticos durante el preprocesamiento de datos antes de crear un modelo de aprendizaje automático. El escalado puede marcar la diferencia entre un modelo débil y uno superior. Las técnicas más comunes de escalado de características son la **normalización** y la **estandarización**, como se verá más adelante en la sección Z-score.

El ajuste por escalado de atributos es necesario para una predicción y unos resultados correctos debido a las siguientes razones (Gil *et al.*, 2013):

- Los coeficientes de regresión están directamente influenciados por la escala de los atributos.
- Las características con mayor escala dominan sobre las características de menor escala.

Ideas clave

- Los algoritmos iterativos de primer orden, como descenso por gradiente, se puede ejecutar fácilmente si tenemos valores escalados.
- Algunos de los algoritmos reducen el tiempo de ejecución cuando se escalan las características.
- Algunos algoritmos se basan en distancias euclidianas, las cuales son muy sensibles a las escalas de los atributos.

E l **ajuste por escalado** de características es esencial para los algoritmos de aprendizaje automático que calculan distancias entre datos. Como se explicó anteriormente, el algoritmo ML es sensible a las escalas relativas de los atributos, lo que generalmente ocurre cuando se usan los valores numéricos de las características en lugar de su rango. Dado que el rango de valores de los datos sin procesar varía ampliamente, en algunos algoritmos de aprendizaje automático, las funciones objetivas no funcionan correctamente sin normalización. Los algoritmos que dependen de la distancia y las curvas gaussianas son sensibles a la escala de los atributos y, por ende, susceptibles a utilizar datos normalizados; algunos ejemplos son regresión lineal y logística, redes neuronales artificiales, máquinas de vectores de soporte, clustering por K-Means, K-Nearest Neighbors, análisis de componentes principales y descenso por gradiente (Raschka, 2014).

El ajuste por escalado es una transformación monótona de variables, y estas no afectan a los algoritmos que se basan en reglas, por ende, no requieren normalización. Los algoritmos basados en árboles utilizan reglas (series de desigualdades) y no necesitan normalización de las características, por ejemplo: Árboles de Clasificación y Regresión (CART, por sus siglas en inglés), Bosques aleatorios y Árboles de decisión impulsados por gradientes. Ciertos algoritmos como Análisis Discriminante Lineal (LDA) y Naive Bayes están equipados en su diseño para manejar el problema de diferentes características y, en consecuencia, dan un peso a cada característica (Roy, 2020). Es posible que el ajuste por escalado de

atributos en estos algoritmos no tenga mucho efecto.

El ajuste mín-máx escala todos los valores numéricos $x(i)$ de un atributo numérico a un rango específico definido por $[x\ min, x\ max]$, donde $x\ min$ corresponde al mínimo y $x\ max$ del conjunto de datos. Para obtener un nuevo valor transformado $x\ norm\ (i)$ se utiliza la siguiente expresión a cada valor $x\ (i)$ (García *et al.*, 2015):

```
xnorm(i)=(x(i)-xmin)/(xmax-xmin)
```

Normalización generalmente se refiere a un caso particular de ajuste mín-máx en el que el intervalo final es [0,1], es decir, xmin=0 y xmax=1. El intervalo [-1,1] también es típico al normalizar los datos. El uso de una normalización para reescalar todos los datos al mismo rango de valores evitará que aquellos atributos con una gran diferencia xmax-xmin dominen sobre los demás en el cálculo de la distancia y engañen al proceso de aprendizaje. Esta normalización es conocida por acelerar el proceso de aprendizaje en las Redes Neuronales Artificiales ayudando a que los pesos converjan más rápido (García *et al.*, 2015).

Python proporciona la librería de `preprocessing`, que contiene la función `normalize` para normalizar los datos. Toma una matriz como entrada y normaliza sus valores entre 0 y 1, luego devuelve una matriz de salida con las mismas dimensiones que la entrada. Sin embargo, existe un enfoque aún más conveniente utilizando el módulo de `preprocessing` de una de las librerías de aprendizaje automático de código abierto de Python, SciKit-Learn. Mediante la función `MinMaxScaler` se transforman los datos para ajustar a un rango entre 0 y 1.

Ajuste Mín-Máx

En este ejercicio se usa la base de datos Breast Cancer del repositorio de machine learning UCI. Una vez se carga el dataset, se implementa la siguiente codificación para obtener los datos normalizados:

```
from sklearn.preprocessing import MinMaxScaler
```

```
min_max_scaler = MinMaxScaler()

X_norm = min_max_scaler.fit_transform(X)

df_cancer_norm = pd.DataFrame(X_norm)

df_cancer_norm["diagnosis"] = df_cancer["diagnosis"]

df_cancer_norm.columns = df_cancer.columns

df_cancer_norm.describe()
```

Como resultado de la ejecución del código anterior, el rango de los datos ahora es [0,1] como se observa en la Figura 9:

	diagnosis	radius	texture	perimeter
count	569.000000	569.000000	569.000000	569.000000
mean	0.338222	0.323965	0.332935	0.216920
std	0.166787	0.145453	0.167915	0.149274
min	0.000000	0.000000	0.000000	0.000000
25%	0.223342	0.218465	0.216847	0.117413
50%	0.302381	0.308759	0.293345	0.172895
75%	0.416442	0.408860	0.416765	0.271135
max	1.000000	1.000000	1.000000	1.000000

Figura 9. Ajuste Mín-Máx de cuatro atributos de la base de datos Breast Cancer. Fuente: elaboración propia.

3.7. Normalización Z-score

La normalización se usa cuando se quieren restringir los valores entre dos números, típicamente entre [0,1] o [-1,1]; además, los datos se presentan sin unidades. Entre las normalizaciones más empleadas para problemas de clasificación de datos se encuentra **Z-score,** que consiste en la estandarización de un conjunto de datos, transformándolos a un espacio de distribución normal con una media igual a 0 y desviación estándar de 1 (Gil *et al.*, 2014). Por lo tanto, los valores para un atributo x_i son normalizados con base en la media y la desviación estándar. A continuación, en la Figura 10, se muestra cómo se ven los datos después de escalar en el plano xy.

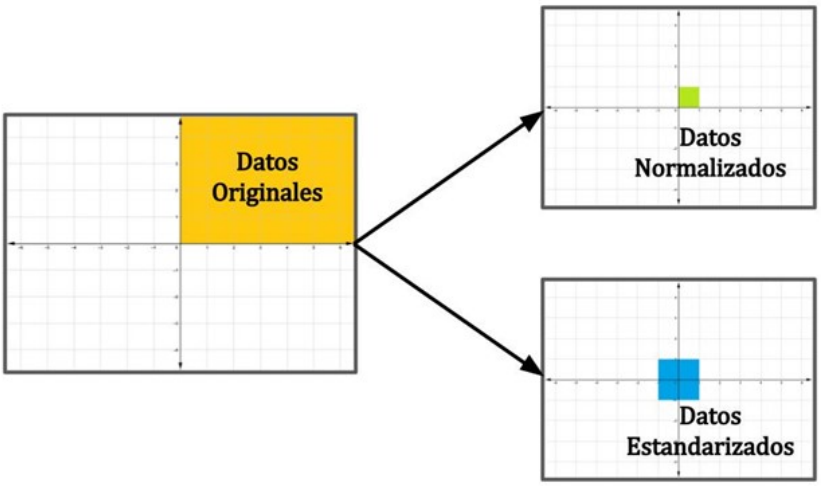

Figura 10. Ajuste por escalada de atributos. Fuente: elaboración propia.

En algunos casos, la normalización mínimo-máximo no es factible, especialmente cuando se desconocen el mínimo o máximo de un atributo $x(i)$. Incluso la presencia de valores atípicos puede sesgar esta normalización al agrupar los valores y limitar la precisión digital disponible para representarlos (García *et al.*, 2015).

Al aplicar esta transformación, los valores de los atributos ahora presentan una media igual a 0 y una desviación estándar de 1; además, el resultado obtenido de la fórmula permite saber lo lejos que se encuentra un dato concreto respecto a la media. Por otro lado, permite la estandarización de los datos favoreciendo la comparación de conjuntos distintos, porque están en unidades diferentes o porque se trata de valores muy dispares. Por ejemplo, si un profesor desea comparar la nota de un alumno de una asignatura de pregrado con la nota de otro, se puede valer del z-score, el cual le informará si los alumnos que se quieren comparar están por encima o por debajo de la media de su clase y cuán lejos están de la misma, permitiéndole una comparación de las notas de una manera más real y exacta.

En Python, se puede también realizar una estandarización de los datos; es decir, se puede cambiar la distribución de cada atributo para que tenga una media de cero y una desviación estándar de uno (variación unitaria), como lo hace la normalización Z-score (Brownlee, 2020), a través de la biblioteca scikit-learn con su sklearn.datasets :

La función StandardScaler de sklearn.preprocessing proporciona una función para estandarizar los datos.

Ajuste Z-SCore

En este ejercicio utilizaremos la base de datos Breast Cancer depositada en el repositorio de machine learning UCI. Una vez se carga el dataset, se implementa la siguiente codificación para obtener los datos normalizados:

```
from sklearn.preprocessing import StandardScaler

standar_scaler = StandardScaler()

X_norm = standar_scaler.fit_transform(X)

df_cancer_norm = pd.DataFrame(X_norm)
```

```
df_cancer_norm["diagnosis"] = df_cancer["diagnosis"]

df_cancer_norm.columns = df_cancer.columns

df_cancer_norm.describe()
```

Como resultado, la media de los datos es 0 con una variación de una desviación estándar como se observa en la Figura 11:

	diagnosis	radius	texture	perimeter
count	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02
mean	-1.256562e-16	1.049736e-16	-1.272171e-16	-1.900452e-16
std	1.000880e+00	1.000880e+00	1.000880e+00	1.000880e+00
min	-2.029648e+00	-2.229249e+00	-1.984504e+00	-1.454443e+00
25%	-6.893853e-01	-7.259631e-01	-6.919555e-01	-6.671955e-01
50%	-2.150816e-01	-1.046362e-01	-2.359800e-01	-2.951869e-01
75%	4.693926e-01	5.841756e-01	4.996769e-01	3.635073e-01
max	3.971288e+00	4.651889e+00	3.976130e+00	5.250529e+00

Figura 11. Resumen estadístico de datos con estandarización. Fuente: elaboración propia.

Tanto el ajuste por normalización como por estandarización pueden representarse en un diagrama de dispersión. Mediante la herramienta pyplot de la librería Matplotlib y usando el comando plt.scatter() , se elabora un diagrama de dispersión evaluando dos de las variables, *radius* y *texture*. En verde se representan los datos originales de entrada, en rojo, los estandarizados, y en azul, los normalizados:

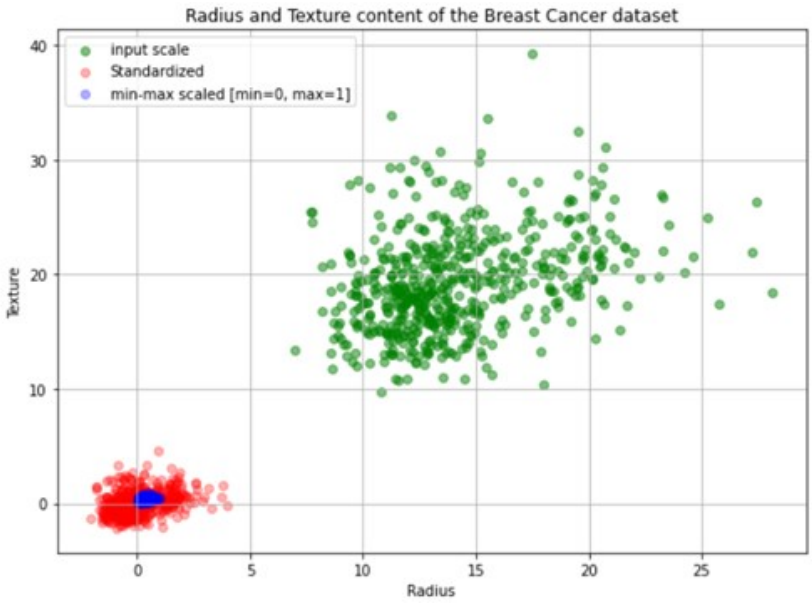

Figura 12. Resumen estadístico de datos con estandarización. Fuente: elaboración propia.

Finalmente, en la Figura 12 se puede observar gráficamente lo que ocurre con los atributos radio y textura, luego del proceso de normalización Min-Max y estandarización con z-score.

3.8. De nominal a binario

Los algoritmos de aprendizaje automático como las máquinas de vector de soporte y las redes neuronales artificiales no pueden lidiar correctamente con los atributos nominales; por ende, la presencia de estas características en un conjunto de datos puede resultar problemática (García *et al.*, 2015). Para solucionar este inconveniente se podría transformar cada variable nominal a una numérica, en la que cada valor nominal se codifica como un número entero empezando desde 0 o 1 en adelante. No obstante, esta opción no es recomendable, debido a que se asume un orden de jerarquía que no existe en los valores del atributo. Esta transformación de valores nominales a una sucesión de enteros (que pueden usarse en operaciones) establece relaciones desiguales entre pares de valores nominales, lo cual no es correcto.

Una transformación alternativa es mapear cada atributo nominal a un conjunto de atributos recién generados; esta técnica es conocida como **codificación en caliente**. En la transformación de nominal a binario, se sustituye la variable nominal por un nuevo conjunto de atributos binarios: si el atributo nominal tiene N valores diferentes, este se reemplaza por un conjunto de N atributos binarios, representando cada uno de los valores posibles. Para cada instancia, solo uno de los N atributos recién creados tendrá un valor de 1, mientras que el resto tendrá el valor de 0. La variable que tiene el valor 1 es la relacionada con el valor original del antiguo atributo nominal (García *et al.*, 2015). Esta transformación, también conocida como transformación 1 a N, puede llevarse a cabo en Python utilizando el `OneHotEncoder` que se implementa en el módulo preprocessing de la librería SciKit-Learn. Por otro lado, una manera más apropiada de ejecutar la transformación nominal a binaria es usando la herramienta `get_dummies` incorporada en la biblioteca Pandas.

Transformación Nominal a Binario

Para ilustrar esta transformación se usan los datos personales de cinco sujetos respecto a su sexo, una variable nominal con dos posibles valores. Después de importar el módulo Pandas, se crea una lista y se ingresan los datos. Luego, mediante la herramienta DataFrame y el comando `print(data_frame)` se genera una visualización de los datos categóricos en columnas. Finalmente, se usa `get_dummies()` para convertir los datos categóricos en binarios para la columna de Sexo.

```
import pandas as pd

data = [["Juan", "Hombre"], ["Pedro", "Hombre"],

["Angie", "Mujer"], ["Paola", "Mujer"],

["Dago", "Hombre"]]

data_frame = pd.DataFrame(data, columns=["Nombre", "Sexo"])

print(data_frame)

df_one = pd.get_dummies(data_frame["Sexo"])

print(df_one)
```

Una línea extra de código se añade con `print(df_one)` para representar los datos transformados. Los dos dataframes descritos se muestran en la Figura 13:

	Nombre	Sexo
0	Juan	Hombre
1	Pedro	Hombre
2	Angie	Mujer
3	Paola	Mujer
4	Dago	Hombre

	Hombre	Mujer
0	1	0
1	1	0
2	0	1
3	0	1
4	1	0

Figura 13. Transformación de un atributo nominal a valores binarios en Python. Fuente: elaboración propia.

3.9. Cuaderno de ejercicios

- **Flavors_of_cacao.csv:** con el siguiente conjunto de datos (https://github.com/clauswilke/dviz.supp/blob/master/data-raw/cacao/flavors_of_cacao.csv) imputa los valores faltantes de la variable Bean-Type aplicando las siguientes técnicas:
  - Imputación por eliminación.
  - Imputación mediante la moda.
  - Investiga otro método para imputar valores faltantes y aplícalo.
- Genera el siguiente conjunto de datos:
  - Realiza la imputación por eliminación.
  - Realiza la imputación mediante la moda o promedio o media.

```
import pandas as pd

from io import StringIO

csv_data = '''A,B,C,D

1.0,2.0,3.0,4.0

5.0,6.0,,8.0

0.0,11.0,12.0,'''

df = pd.read_csv(StringIO(csv_data))

df
```

Ideas clave

- Con el siguiente conjunto de datos: data = np.array([6, 7, 7, 12, 13, 13, 15, 16, 19, 22]) hallar manualmente el valor de Z-Score utilizando la fórmula $xi - \mu/\sigma$

- Con el siguiente conjunto de datos: data = np.array([6, 7, 8, 12, 13, 13, 15, 16, 21, 22]) normaliza manualmente los datos aplicando la técnica de min-max xi-xmin/xmax-xmin.

- Con el siguiente conjunto de datos: data = np.array([6, 7, 8, 12, 13, 13, 15, 16, 21, 22]) estandariza los datos aplicando la técnica de Z-score en Python.

3.10. Referencias bibliográficas

Amón Uribe, I. y Jiménez Ramírez, C. (2009). Hacia una metodología para la selección de técnicas de depuración de datos. *Revista Avances en Sistemas e Informática, 6*(1), 185-190. https://www.redalyc.org/articulo.oa?id=133112608019

Batista, G. y Monard, M., (2003). An analysis of four missing data treatment methods for supervised learning. *Applied Artificial Intelligence, 17*(5), 519-533. https://www.tandfonline.com/doi/abs/10.1080/713827181

Brownlee, J. (2020). *How to Handle Missing Data with Python*. Machine Learning Mastery. https://machinelearningmastery.com/handle-missing-data-python/

Chavez, R. (2020). *Pre-Procesamiento de datos en Python. Valores perdidos y filas duplicadas*. Finanzas Cuantitativas en español. https://ricovictor.com/index.php/2020/08/08/pre-procesamiento-de-datos-en-python-valores-perdidos-y-filas-duplicadas/

Dempster, A., Laird, N., Rubin, D., (1977). Maximum likelihood estimation from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, 39*(1), 1-38. http://www.jstor.org/stable/2984875

Dr. Deng. (2012). *Missingness Mechanism (MCAR, MAR, and MNAR) - A Great Explanation of These Terms*. On Biostatistics and Clinical Trials. https://onbiostatistics.blogspot.com/2012/10/missingness-mechanism-mcar-mar-and-mnar.html

DeShon, J. (2016). *Creating a Q-gram algorithm to determine the similarity of two character strings*. Boehringer-Ingelheim Vetmedica. https://support.sas.com/resources/papers/proceedings16/2080-2016.pdf

Díaz Monroy, L. G. (2007). *Estadística Multivariada: Inferencia y Métodos* (2ª ed.). Departamento de Estadística, Facultad de Ciencias, Universidad Nacional de Colombia. https://repositorio.unal.edu.co/handle/unal/79907

Flores, M. Á. (2020). *El coeficiente de correlación de Pearson (con ejemplo en Python)*. Medium. https://medium.com/@hdezfloresmiguelangel/el-coeficiente-de-correlaci%C3%B3n-de-pearson-con-ejemplo-en-python-6e8588f67e35

García, S., Luengo, J. y Herrera, F. (2015). *Data Preprocessing in Data Mining*, vol. 72. (1ª ed.). Springer International Publishing. https://link.springer.com/book/10.1007/978-3-319-10247-4

Gil González, W. J., Mora Flórez, J. J. y Pérez Londoño, S. M. (2014). Análisis del procesamiento de los datos de entrada para un localizador de fallas en sistemas de distribución. *Revista Tecnura, 18*(41). https://revistas.udistrital.edu.co/index.php/Tecnura/article/view/7023

Guevara Valdez, V. y Amón Uribe, I. (2016). *Métodos y herramientas para la detección de duplicados en paralelo*. Repositorio Institucional UPB. https://repository.upb.edu.co/handle/20.500.11912/2847

Hand, D. J., Adèr, H. J., Mellenbergh, G. J., (2008). *Advising on Research Methods: A Consultant's Companion* (pp. 305-332). Huizen, Netherlands. Editorial Johannes van Kessel.

Jain, D. (2023). *Redundancy and Correlation in Data Mining*. GeeksforGeeks. https://www.geeksforgeeks.org/redundancy-and-correlation-in-data-mining/

Kuhn, M. y Johnson, K. (2019). *Feature Engineering and Selection: A Practical Approach for Predictive Models* (1ª ed.). CRC Press. ISBN: 978-1138079229.

Little, R. J. A., Rubin, D. B., (1987). *Statistical Analysis with Missing Data* (1ª ed.). Serie Wiley en Probabilidad y Estadística.

Pyle, D. (1999). *Data Preparation for Data Mining*. Editorial Morgan Kaufmann Publishers.

Raschka, S. (2014). *About Feature Scaling and Normalization*. Sebastian Raschka. https://sebastianraschka.com/Articles/2014_about_feature_scaling.html

Raschka, S. (2015). Python machine learning. Packt publishing. https://www.packtpub.com/product/python-machine-learning/9781783555130

Sakkaf, Y. (2020). *Chi-Squared Test for Feature Selection with implementation in Python*. Medium. https://towardsdatascience.com/chi-squared-test-for-feature-selection-with-implementation-in-python-65b4ae7696db

Scheuren, F. (2005). Multiple imputation: how it began and continues. *The American Statistician, 59*(4), 315–319. http://www.jstor.org/stable/27643702

Cómo manejar datos perdidos en Python

González, L. (2018). *Librería Pandas de Python*. Aprende IA. https://aprendeia.com/libreria-pandas-de-python-tutorial/

En este recurso didáctico el estudiante podrá encontrar información complementaria a lo presentado en esta unidad. En esta página web puede encontrar formas de datos perdidos, cómo corregir datos perdidos, eliminar datos perdidos y reemplazar datos perdidos con Python.

Manejo de datos faltantes

Roy, B. (2020). *Todo Sobre El manejo De Datos Faltantes*. DataSource.ai. https://www.datasource.ai/es/data-science-articles/todo-sobre-el-manejo-de-datos-faltantes

En este recurso el alumno podrá ampliar más información respecto a datos faltantes, así como técnicas de imputación de datos.

Normalización de Columnas con Python

Sarathi, S. (2023). *Normalizar una columna en Pandas Dataframe*. Delft Stack. https://www.delftstack.com/es/howto/python-pandas/pandas-normalize/

En esta página web, el estudiante puede encontrar codificación en base a Python e indicaciones sobre cómo normalizar datos a partir de la biblioteca Pandas; con lo cual podrá profundizar más sobre la normalización de columnas a partir de los métodos más usados, como la media o z-score, y la normalización mínima-máxima.

Cambio de escala de datos para el aprendizaje automático en Python con Scikit-Learn

Brownlee, J. (2014). *Rescaling Data for Machine Learning in Python with Scikit-Learn*. Machine Learning Mastery. https://machinelearningmastery.com/rescaling-data-for-machine-learning-in-python-with-scikit-learn/

En esta página web, el estudiante puede encontrar codificación en base a Python e indicaciones sobre cómo normalizar y estandarizar datos a partir de la biblioteca scikit-learn con la base de datos Iris. Esta herramienta puede resultar un complemento a lo ya tratado en la unidad sobre las generalidades de normalización.

Test

1. Son parte de las técnicas para imputar datos ausentes:

    A. Enfoque probabilísticos.

    B. A, C y D son verdaderas.

    C. Imputación por eliminación.

    D. Imputación por media, mediana y moda.

2. La siguiente línea de código `imputer = SimpleImputer(missing_values=nan, strategy='mean')` representa:

    A. Imputación de valores mediante el enfoque probabilísticos.

    B. Imputación de valores mediante el algoritmo de esperanza-maximización.

    C. Imputación por eliminación.

    D. Imputación por media.

3. La siguiente línea de código `dataset.dropna(inplace=True)` representa:

    A. Imputación de valores mediante el enfoque probabilístico.

    B. Imputación de valores mediante el algoritmo de esperanza-maximización.

    C. Imputación por eliminación.

    D. Imputación por media.

4. ¿Por qué es importante intentar llenar los valores faltantes?

    A. No es agradable ver un conjunto de datos con valores faltantes.

    B. Porque la muestra no es representativa.

    C. Puede afectar el método de aprendizaje automático elegido.

    D. Todas son verdaderas.

5. Son técnicas para encontrar atributos redundantes:

A. Correlación.

B. Covarianza.

C. Prueba de datos nominales.

D. Todas las anteriores.

6. En el siguiente conjunto de datos: data = {'Id': [1, 2, 3, 4, 5, 6, 7, 7], 'Primer_nombre': ['Angela', 'Adrian', 'Theodoro', 'Angela', 'Adrian', 'Beatriz', 'Olivia', 'Olivia'], 'Primer_apellido': ['Castro', 'Guzman', 'Rivadeneira', 'Castillo', 'Casas', 'Perez', 'Apraez', 'Apraez'], 'Edad': [27, 31, 36, 27, 53, 48, 36, 36], 'Estatura': [1.67, 1.80, 1.61, 1.67, 1.88, 1.69, 1.62, 1.62]} , ¿cuántos registros duplicados existen?

A. 0.

B. 1.

C. 2.

D. 3

7. ¿Por qué aplicar normalización a los datos?

A. Porque se utiliza la distancia euclídea y necesita la normalización para poder calcularse.

B. Los atributos con magnitudes altas pesarán mucho más que los atributos con magnitudes bajas.

C. A y B son ciertas.

D. A y B son falsas.

8. Son técnicas de escalado:

A. Normalización.

B. Z-score.

C. A y B son ciertas.

D. A y B son falsas.

9. Es una característica del algoritmo Z-score:

A. Consiste en la estandarización de un conjunto de datos.

B. Transforma los datos en un espacio de distribución normal.

C. La media es cero y la desviación es 1.

D. Todas las anteriores son características del algoritmo Z-score.

10. Es una ventaja que se tiene al transformar una variable nominal a una numérica:

A. Las redes neuronales no pueden lidiar con atributos nominales.

B. Las máquinas de vector de soporte no trabajan correctamente con atributos nominales.

C. Varios algoritmos no pueden lidiar con valores nominales y cuando se transforman los datos se pueden ejecutar.

D. Ninguna de las anteriores respuestas es una ventaja.