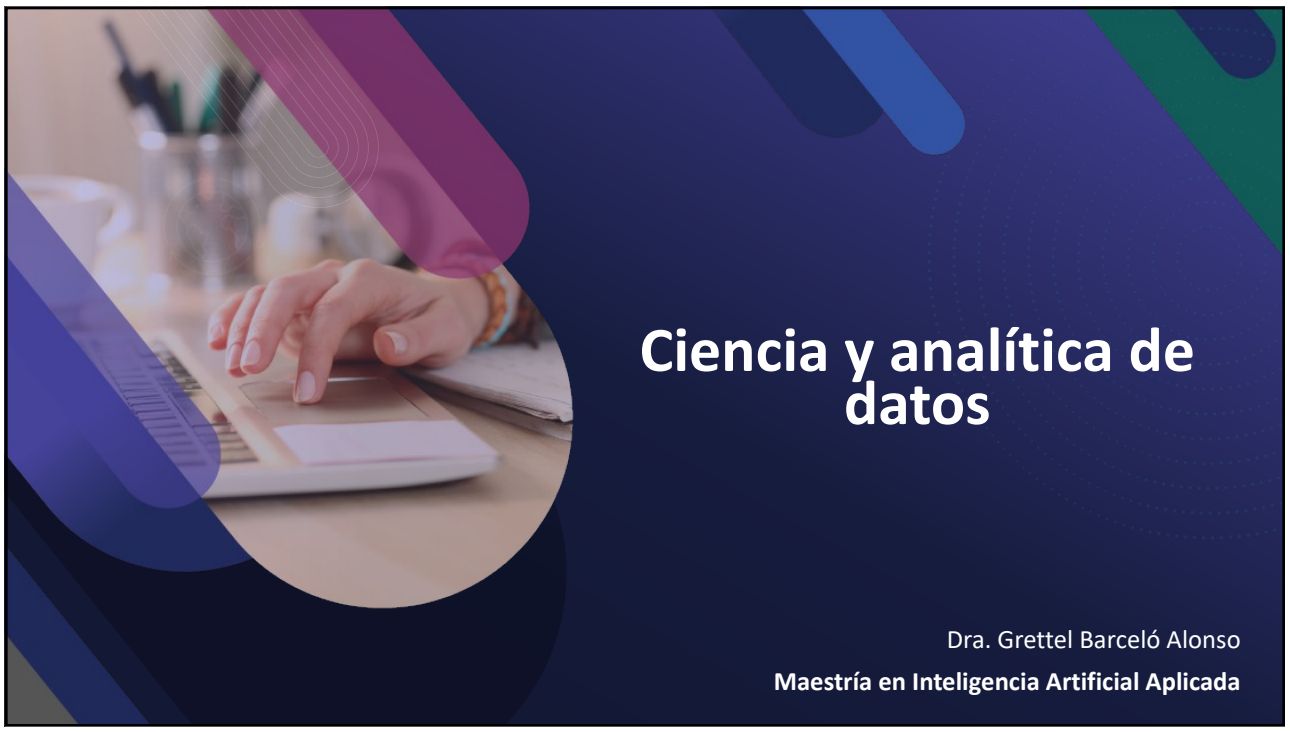

Ciencia y analítica de datos
Dra. Grettel Barceló Alonso
Maestría en Inteligencia Artificial Aplicada

1


Active class
Sesión 3

2

3

Problemas detectados con EDA

Preprocesamiento

- Alto porcentaje de valores faltantes
  - `isna()`
- Valores atípicos
  - `boxplot, método IQR`
- Alta cardinalidad de atributos categóricos
  - `nunique()`

- Distribución sesgada de atributos numéricos
  - `skew(), histogramas`
- Alta correlación entre características (redundancia)
  - `corr(), diagramas de dispersión, mapas de calor`

Ingeniería y selección de características

4

Manejo valores faltantes

Estrategia	Definición	Pros	Contras	Implementación con pandas
Preservar	Mantener los valores faltantes.	El conjunto se conserva su estado original.	Sólo algunas herramientas de análisis de datos lo permiten	-
Eliminación por lista	Excluir todos los casos (en lista) que tienen valores faltantes. Pueden ser filas o columnas*	Preserva la distribución si MCAR.	1. Puede descartar demasiados datos y dañar el modelo. 2. Puede generar estimaciones sesgadas si no es MCAR (ya que mantenemos una submuestra especial de la población)	`dropna()`
Imputación media/mediana /moda	Reemplazar el NaN por la media/mediana/moda (para características categóricas) de esa variable**	Buena práctica si MCAR.	1. Puede distorsionar la distribución. 2. Puede distorsionar la relación con otras variables.	`fillna()`

* Cuando la cantidad de valores faltantes en una variable es lo suficientemente grande (aproximadamente más del 25 %), eliminar el atributo es mejor que estimar los valores faltantes.

** Cuando la variable tiene una distribución normal, usar la media. Si está sesgada, usar la mediana.

5

Valores faltantes

- Es una biblioteca para aprendizaje automático de software libre para Python.
- Incluye varios algoritmos de clasificación, regresión y análisis de grupos.
- Está diseñada para interoperar con las bibliotecas numéricas y científicas *NumPy* y *SciPy*.

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='median')
no_missing_data = imputer.fit_transform(data)
```

`strategy` = {mean, median, most_frequent, constant}

Otros métodos

`IterativeImputer()` - Imputador multivariado que estima cada característica a partir de todas las demás.

`KNNImputer()` - Imputación para completar valores faltantes utilizando k-vecinos más cercanos.

6

Valores atípicos

- Un valor atípico es un punto de datos que es significativamente diferente de los datos restantes.
- Pueden afectar el rendimiento de algunos modelos de aprendizaje automático.

Métodos para la detección de atípicos

Límite arbitrario	Identificar valores atípicos basados en límites arbitrarios (requiere entendimiento del negocio).
Media y desviación estándar	`lower_limit = df[variable].mean() - 3 * df[variable].std()` `upper_limit = df[variable].mean() + 3 * df[variable].std()`

Método IQR	`IQR = df[variable].quantile(0.75) - df[variable].quantile(0.25)`
	`lower_limit = df[variable].quantile(0.25) - (IQR * 1.5)`
	`upper_limit = df[variable].quantile(0.75) + (IQR * 1.5)`

7

Estrategias manejo valores atípicos

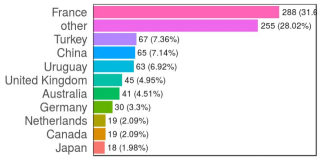
Estrategia	Definición	Pros	Contras	Implementación
Imputación media/mediana/moda	Reemplazar el valor atípico por la media/mediana/moda de esa variable.	Preservar la distribución.	Se pierde información de valores atípicos si hay uno.	`df[variable].mask( df[variable] > upper_limit, df[variable].median(), inplace = True)`
Límites	Limitar el máximo y mínimo de una distribución en un valor establecido.	Evita el sobreajuste del modelo.	Distorsiona la distribución.	`df[variable].mask( df[variable] > upper_limit, upper_limit, inplace = True)`
Descarte	Eliminar todas las observaciones que son valores atípicos.	.	Se pierde información de valores atípicos si hay uno.	`df.drop( outliers.index, inplace = True)`

* Cuando la cantidad de outliers es relativamente grande (aunque deberían estar alrededor del 5%), se debe investigar el origen para tomar mejores decisiones.

** Se recomienda hacer varios modelos y comparar resultados.

8

Alta cardinalidad



- El número de etiquetas dentro de una variable categórica se conoce como cardinalidad.
- Un alto número de etiquetas dentro de una variable (cientos de valores únicos) se conoce como alta cardinalidad.

Problemas

- Las variables con demasiadas etiquetas tienden a dominar sobre aquellas con solo unas pocas etiquetas.
- Una gran cantidad de etiquetas dentro de una variable puede introducir ruido con poca o ninguna información, lo que hace que los modelos de aprendizaje automático sean propensos a sobreajustarse.

Estrategias

- Agrupación de categorías con conocimiento empresarial.
- Agrupación de categorías con poca ocurrencia en una categoría única.

¿Con qué método de *pandas*?

`groupby()`

9

Características

Una característica es un atributo de datos que es significativo para el proceso de aprendizaje automático.
También conocida como:

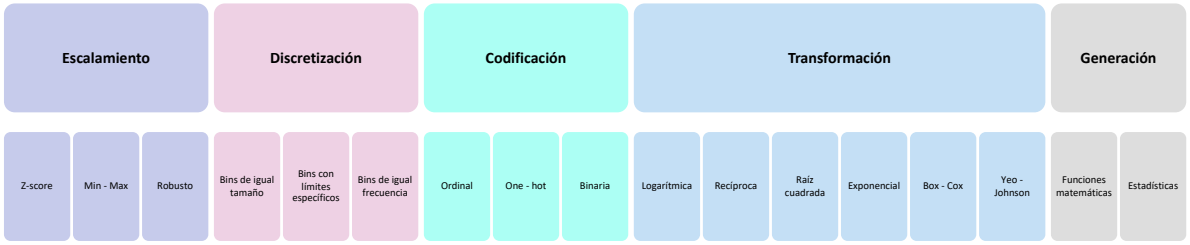
- variable independiente
- predictor
- variable de entrada

El objetivo, será la variable que se predice en el aprendizaje supervisado. También conocido como:

- variable dependiente
- variable de respuesta
- variable de salida

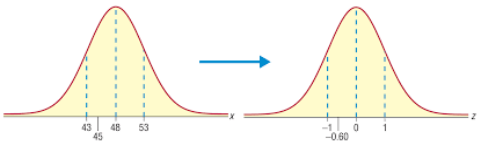
10

Ingeniería de características



11

Escalamiento



Se estandariza el rango de las variables independientes o características para tener escalas similares. También se conoce como normalización.

Aplicar a: Variables numéricas continuas. Aunque es usual aplicar también estas transformaciones a variables numéricas discretas.

¿Por qué es necesario?

- Para que todas las variables sean, en principio, igualmente competitivas en cuanto a su relevancia en la construcción del modelo.
- Ayuda a que los métodos de minimización del error como el gradiente descendente, no oscilen demasiado alrededor del valor mínimo buscado.
- Los algoritmos que implican el cálculo de distancias también se ven afectados por la magnitud de la característica.

12

Escalamiento

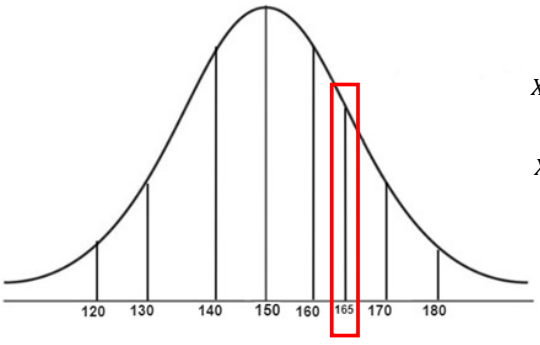
Estrategia	Definición	Pros	Contras
Z-score / estandarización / transformación gaussiana	Resta la media y escala los datos a la varianza unitaria. $X_{scaled} = \frac{X - X.mean}{X.std}$	La característica se reescala para tener media 0 y desviación estándar 1.	El uso de la media no permite aminorar el efecto negativo de los outliers.
Min-Max	Transforma características escalando cada característica a un rango dado. Predeterminado a [0,1]. $X_{scaled} = \frac{X - X.min}{X.\max - X.min}$	Es la que menos distorsiona los datos originales y hace lo mínimo para que sean competitvas las variables entre sí.	Comprime las observaciones en el rango estrecho si la variable está sesgada o tiene valores atípicos, lo que perjudica el poder predictivo.
Robusto	Elimina la mediana y escala los datos de acuerdo con el rango de cuantiles (el valor predeterminado es IQR) $X_{scaled} = \frac{X - X.median}{IQR}$ $IQR = Q3 - Q1$	El uso de la mediana y el rango intercuartil ayuda a reducir el efecto de outliers.	

13

Z - score

Alturas de los estudiantes de una clase:

- media 150 cm
- desviación estándar 10 cm



$$X_{scaled} = \frac{X - X.mean}{X.std}$$

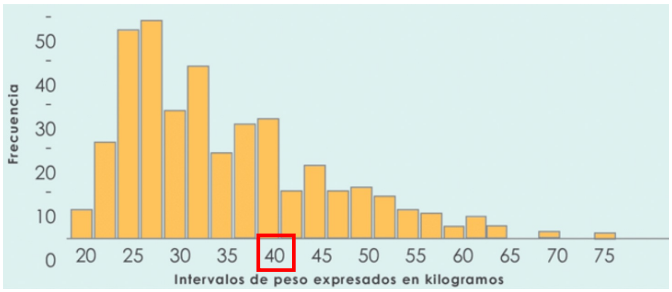
$$X_{scaled} = \frac{165-150}{10} = 1.5$$

14

Min - Max

Pesos de niñas entre 5 y 11 años:

- mínimo 17.5 Kg
- máximo 74.4 Kg



G. Avilés, G. Chávez, y M. Almendarez Hernández, *Análisis de antropometría y de factores determinantes de la prevalencia de obesidad y sobrepeso infantil*, pp. 80-195, Nov. 2017. ISBN 978-607-7777-83-0.

$$X_{scaled} = \frac{X - X.min}{X.\max - X.min}$$

$$X_{scaled} = \frac{40 - 17.5}{74.4 - 17.5} = 0.395$$

15

Escalamiento

Estandarización

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
```

MinMax

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)
```

Robusto

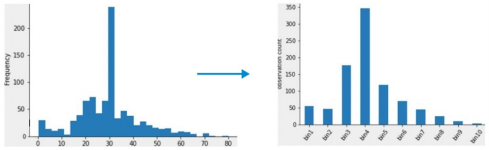
```
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
scaled_data = scaler.fit_transform(data)
```

Nota. Como la salida la devuelve en formato array, para convertirla a dataframe:

```
scaled_df = pd.DataFrame(
                scaled_data,
                columns=scaler.get_feature_names_out())
```

16

Discretización (*binning*)



Se transforman las variables continuas en discretas mediante la creación de un conjunto de intervalos contiguos que abarca el rango de valores de la variable.

Aplicar a: Variables numéricas continuas

¿Por qué es necesario?

- Ayuda a mejorar el rendimiento del modelo mediante la agrupación de atributos similares.
- Mejora la interpretabilidad con valores agrupados.
- Minimizar el impacto de los valores extremos.
- Evitar el sobreajuste posible con variables numéricas.

17

Discretización (*binning*)

Estrategia	Definición	Pros	Contras	Implementación con pandas
Bins de igual tamaño	Divide los valores de la variable en N contenedores del mismo ancho.		Sensible a la distribución sesgada.	`cut()` `bins=N`
Bins con límites específicos	Divide los valores de la variable en contenedores definidos por los números especificados.	Puede ayudar a mejorar el rendimiento del algoritmo.	Se necesita conocimiento de dominio para establecer los límites de cada contenedor.	`cut()` `bins=[0,4,8]`
Bins de igual frecuencia	Divide los valores de la variable en N contenedores, donde cada contenedor contiene la misma cantidad de observaciones.	Puede ayudar a mejorar el rendimiento del algoritmo.	Este agrupamiento arbitrario puede interrumpir la relación con el objetivo.	`qcut()` `bins=N`

data = [2, 4, 7, 9, 10, 11]

- bins = 3 (ancho de cada bin)
- bins = [0, 3, 9, 12] (límites de los bins)
- bins = 3 (cantidad de bins)

18

Discretización (*binning*)

```
from sklearn.preprocessing import KBinsDiscretizer
grouper = KBinsDiscretizer(n_bins=3, encode='ordinal',
                           strategy='uniform')
grouped_data = grouper.fit_transform(data)
```

`encode`= {onehot, onehot-dense, ordinal}
`strategy` = {uniform, quantile, kmeans}

19

Codificación

fuel
gas
diesel
gas
gas

→

gas	diesel
1	0
0	1
1	0
1	0

Se transforman las variables categóricas a números para que puedan ser procesadas por algoritmos de aprendizaje automático y otras técnicas estadísticas.

Aplicar a: Variables categóricas

¿Por qué es necesario?

- Para que los algoritmos puedan manejar esos valores.
- Incluso si ve que un algoritmo puede tomar entradas categóricas, lo más probable es que el algoritmo incorpore el proceso de codificación en su interior.

20

Codificación

Estrategia	Definición	Pros	Contras	Implementación en pandas
One - hot	Crear nuevas variables binarias para indicar si cierta etiqueta es verdadera o no para esa observación. A estas variables también se les conocen como variables *dummies*. blue ⟶ [1 0 0] ⟶ [0 0] green ⟶ [0 1 0] ⟶ [1 0] ← `drop_first` red ⟶ [0 0 1] ⟶ [0 1]	Mantiene toda la información de esa variable.	Expande el espacio de características dramáticamente si hay demasiadas etiquetas en esa variable.	`get_dummies()` `drop_first=True`
Ordinal	Reemplazar las etiquetas por algún número ordinal si el orden es significativo. Talla S ⟶ 0 Talla M ⟶ 1 Talla L ⟶ 2	Es una transformación muy sencilla.	Puede ingresar sesgo al modelo.	`factorize()`
Binaria	Reemplazar las etiquetas por un código binario que representa las diferentes categorías de la variable. blue ⟶ [0 1] green ⟶ [1 0] red ⟶ [1 1]	Codifica los datos en menos dimensiones que one-hot (log2 # de categorías)	Expande el espacio de características.	

21

Codificación

One - hot

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(drop='first')
encoded_data = encoder.fit_transform(data)
```

Ordinal

```
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder(categories=[['bajo','medio','alto']])
encoded_data = encoder.fit_transform(data)
```

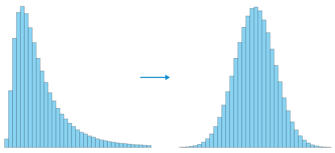
Binaria

```
!pip install category_encoders
from category_encoders.binary import BinaryEncoder
encoder = BinaryEncoder()
encoded_data = encoder.fit_transform(data)
```

`* category_encoders` Un conjunto de transformadores de estilo *scikit-learn* para codificar variables categóricas en numéricas con diferentes técnicas.

22

Transformación



Se reemplazan los valores originales de las variables con una función matemática de esa variable. Las transformaciones intentan llevar la distribución de la variable a una forma más simétrica, es decir, normal o gaussiana.

Aplicar a: Variables numéricas continuas. Aunque es usual aplicar también estas transformaciones a variables numéricas discretas.

¿Por qué es necesario?

- En la regresión lineal y logística se asume normalidad, que significa que cada variable X debe seguir una distribución normal.
- Los modelos restantes, incluidas las redes neuronales, SVM, los métodos basados en árboles y PCA, no hacen ninguna suposición sobre la distribución de las variables. Sin embargo, en muchas ocasiones el rendimiento del modelo puede beneficiarse de una distribución normal.
- Como se modifica la distribución de los datos de entrada, ayuda a manejar mejor los datos de la cola, los cuales se estarían comportando como valores extremos (outliers)

23

Transformación

Estrategia	Definición	Implementación con numPy
Logarítmica	$X_{transf} = \log(X)$	`transformed_data = np.log(data)`
Recíproca	$X_{transf} = \frac{1}{X}$	`transformed_data = np.reciprocal(data)`
Raíz cuadrada	$X_{transf} = \sqrt{X}$	`transformed_data = np.sqrt(data)`
Exponencial	$X_{transf} = X^m$	`transformed_data = np.power(data, exponent)`
Box - cox	$X_{transf} = \begin{cases} \frac{X^\lambda - 1}{\lambda} & si\ \lambda > 0\ y\ X > 0 \\ log(X) & si\ \lambda = 0\ y\ X > 0 \end{cases}$	
Yeo - Johnson	$X_{transf} = \begin{cases} \frac{(X+1)^\lambda - 1}{\lambda} & si\ \lambda \neq 0\ y\ X \geq 0 \\ \log(X+1) & si\ \lambda = 0\ y\ X \geq 0 \\ -\frac{(-X+1)^{2-\lambda} - 1}{2-\lambda} & si\ \lambda \neq 2\ y\ X < 0 \\ -\log(-X+1) & si\ \lambda = 2\ y\ X < 0 \end{cases}$	

24


scikit
learn
Nota. standarize = true por defecto. Así, después de la transformación, escalaría con z-score
Transformación
from sklearn.preprocessing import FunctionTransformer
Logarítmica
transformer = FunctionTransformer(func=np.log)
Recíproca
transformer = FunctionTransformer(func=np.reciprocal)
Raíz cuadrada
transformer = FunctionTransformer(func=np.sqrt)
Exponencial
transformer = FunctionTransformer(lambda x:
np.power(x, 0.3))
from sklearn.preprocessing import PowerTransformer
Box - cox
transformer = PowerTransformer(method='box-cox',
standardize=False)
Yeo - Johnson
transformer = PowerTransformer(method='yeo-johnson',
standardize=False)
transformed_data = transformer.fit_transform(data)

25


Tecnológico de Monterrey
D.R.© Tecnológico de Monterrey, México, 2022.
Prohibida la reproducción total o parcial
de esta obra sin expresa autorización del
Tecnológico de Monterrey.

26