Feature Engineering

Preprocesado de datos

Concepto

Los datos recolectados suelen presentar ausencia o inconsistencia de algunos datos. Esto puede ocasionar errores o resultados no deseados en los modelos de ML. De ahí su importancia

En todos los algoritmos que vamos a usar existen las funciones fit() y transform() que comparten muchas clases que implementan algoritmos de ML dentro de Scikit-learn. El modo de operar será:

Instanciar objeto de alguna clase de interés con argumentos

Invocar el método fit() que se ajusta con los datos de entrenamiento y estima valores para los parámetros internos

Llamar a transform() para escalar, imputar valores ausentes, etc.

Existe fit _transform para hacerlo en un solo paso

* fit() se usa además de para preprocesamiento de datos para algoritmos de ML, como Regresión Logística.

- Algoritmos supervisados X e Y
- No supervisados X
- X matruz con los atributos descriptivos
- Y array con etiquetas de clase.

Creación de entrenamiento y pruebas

Con la biblioteca Scikit-learn esto puede ser logrado a través de la función train_test_split() del submódulo model_selection.

```
import pandas as pd
from sklearn.model_selection import train_test_split
df = pd.read_csv("precios_casas.csv")
X = df.iloc[:,1:].values
y = df.iloc[:,0].values
print(X)
entX, pruX, enty, pruy = train_test_split(X, y, test_size=0.2, random_state=100)
```

Manejo de datos ausentes

- Eliminar las filas donde hay valores ausentes, así:
  df.dropna(axis=0), esta es la opción por defecto.

- Eliminar las columnas donde hay valores ausentes en alguna fila, así:
  df.dropna(axis=1)

  Aquí, axis=0 se refiere a filas y axis=1 a columnas.

```
df.isnull().sum()
```

Siguiendo con el ejemplo, el dataframe que usaremos será el que sigue:

```
df = pd.DataFrame([
    ['1', 1, 30],
    ['2', 1, 32],
    ['3', 0]],
)
df.columns = ['codigo','credito','edad']
```

Manejo de datos ausentes

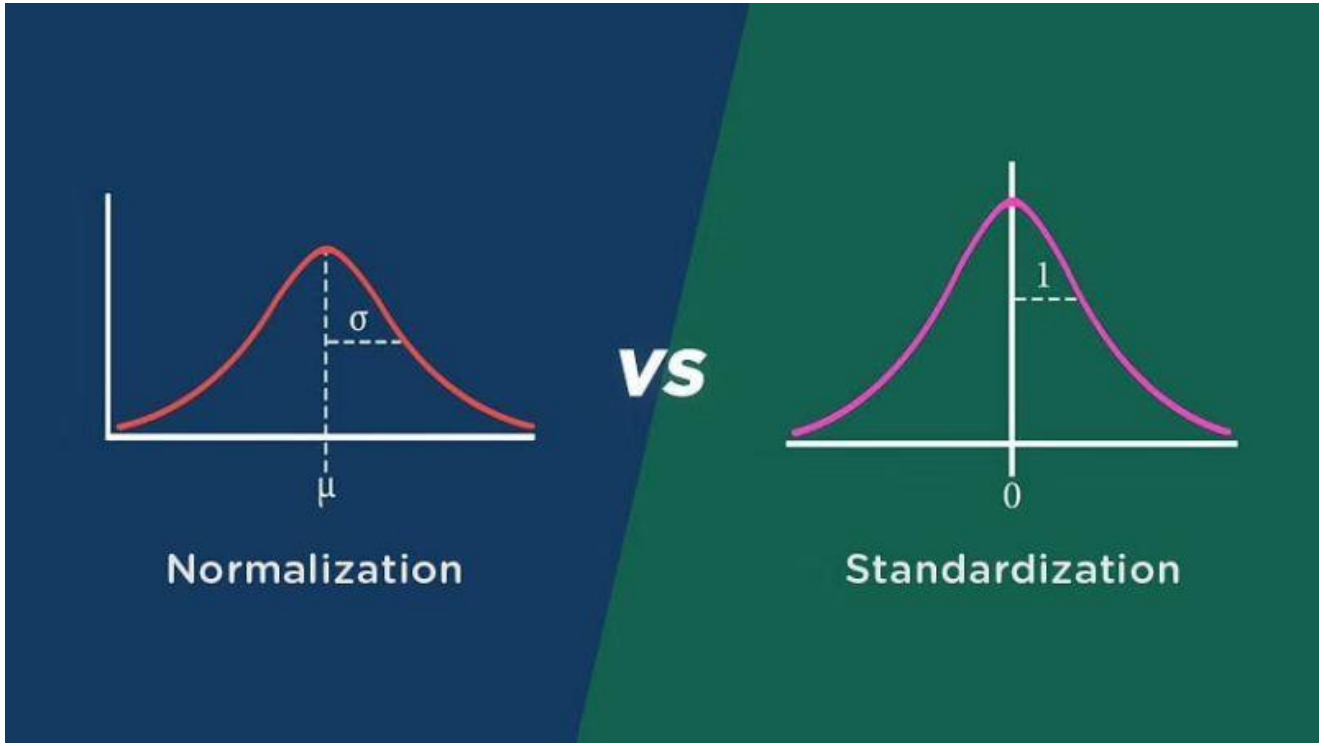
```
df.replace('?', np.nan, inplace=True)
```

Una vez se haya realizado un reemplazo como el anterior se realiza la imputación utilizando el siguiente fragmento de código:

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
imp = imp.fit(df.values)
imp_datos = imp.transform(df.values)
```

```
[[ 1.  1. 30.]
 [ 2.  1. 32.]
 [ 3.  0. 31.]]
```

Escalamiento de características



Normalización

CONCEPTO MUY IMPORTANTE→ <u>Lo que hago es pensar en el histograma de cada variable</u>. Voy a coger cada columna numérica del DataFrame y aplicarle una **transformación REVERSIBLE a toda la columna**.

* **Estandarizar**→ Calculo la media de toda la columna, M. Calculo la desviación típica de toda la columna, S.
Ahora dato por dato, a cada uno le resto M y eso lo divido entre S

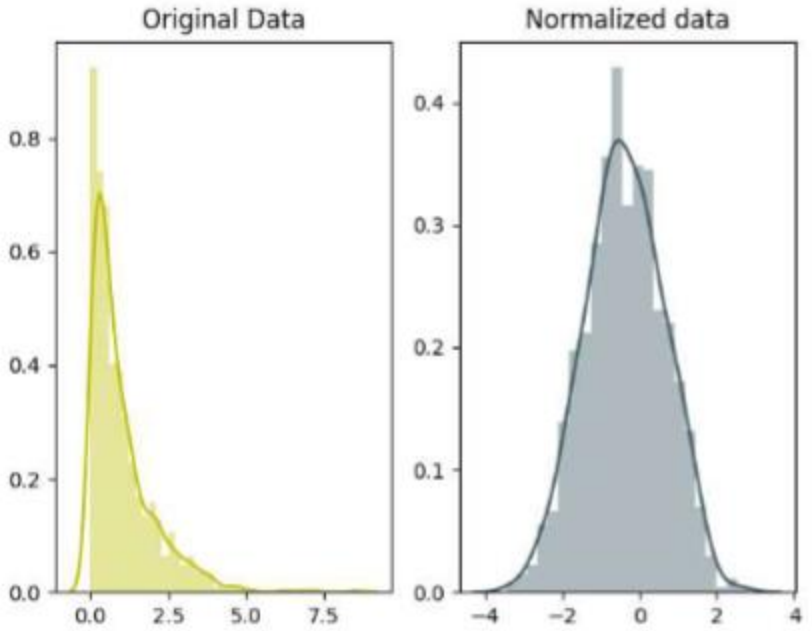
¿Qué busco? → Que los datos tengan media cero y desviación unitaria
Media cero → Ahora los datos están alrededor de 0 (positivos y negativos, pero en media valen entre todos cero)
Desviación típica unitaria→ Además de estar alrededor de 0, están cerca de 0 (varianza es 1 que es poco)
Conclusión: tengo datos pegados a 0 por arriba y por abajo

* **Escalar**→ Calculo el mínimo de toda la columna, Mín. Calculo el máximo de toda la columna, Máx. Calculo el rango de la columna, R = Máx - Mín
Ahora dato por dato, a cada uno le resto Mín y eso lo divido entre R

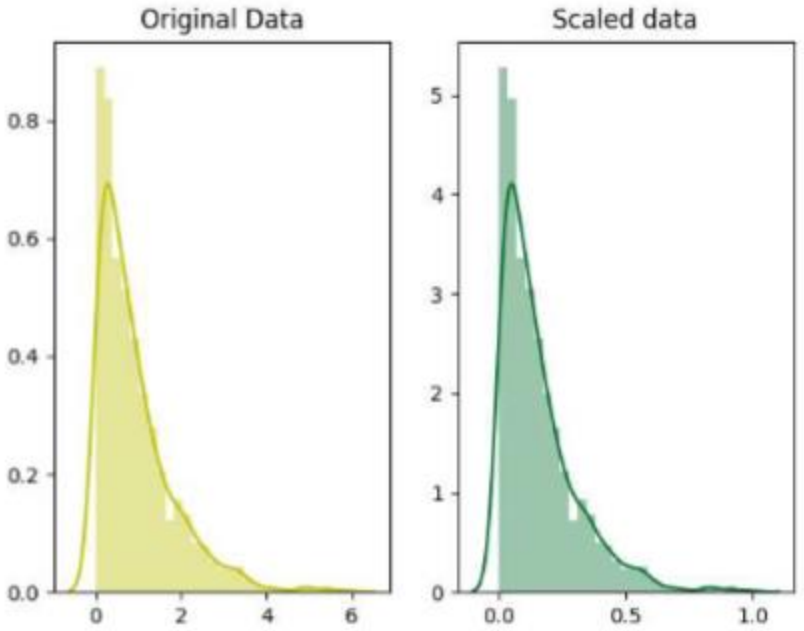
¿Qué busco? → Que los datos estén entre 0 y 1
Conclusión: tengo datos solo dentro del intervalo [0,1]

Estandarización

Estandarizar



Escalar



Manejo de datos categóricos

Se recomienda convertir a números todos los datos categóricos

Es necesario codificar la etiqueta de clase del conjunto de datos a un valor numérico. (LabelEncoder)

```
df = pd.DataFrame([
      ['M', 30, 'Amarillo',"Clase 1'],
      ['P', 28 , 'Azul',"Clase 2'],
      ['J', 21, 'Rojo', 'Clase 1']],
)
df.columns = ['nombre',"edad","color', 'etiqueta']
dfAux = df
```

Convertiremos ahora los valores de la columna *etiqueta* a su representación numérica:

```
from sklearn.preprocessing import LabelEncoder
le_clase = LabelEncoder()
y = le_clase.fit_transform(df.etiqueta)
print(y)
```

Manejo de datos categóricos

Si tuviesemos tres categorías, como ingeniero arquitecto y contados, se podría entender que uno es superior a otro. Para solucionarlo tenemos el One Hot Encoding

```
from sklearn.preprocessing import OneHotEncoder
le_color = LabelEncoder()
ohe_color = OneHotEncoder(categories='auto')
df['color_cod'] = le_color.fit_transform(df.color)
print(df)
```

	nombre	edad	color	etiqueta	color_cod
0	M	30	Amarillo	Clase 1	0
1	P	28	Azul	Clase 2	1
2	J	21	Rojo	Clase 1	2

Manejo de datos categóricos

Ahora transformamos las característica color mediante el objeto ohe_color, cuyo fit_transform espera un array de 2D

```
datos_ohe = ohe_color.fit_transform(df.color_cod.values.reshape(-1,1)).
toarray()
dfOneHot = pd.DataFrame(datos_ohe, columns = ["Color_"+str(int(i)) for i in
range(len(df.color))])
df = pd.concat([df, dfOneHot], axis=1)
print(df)
```

	nombre	edad	color	etiqueta	color_cod	Color_0	Color_1	Color_2
0	M	30	Amarillo	Clase 1	0	1.0	0.0	0.0
1	P	28	Azul	Clase 2	1	0.0	1.0	0.0
2	J	21	Rojo	Clase 1	2	0.0	0.0	1.0

```
df= df.drop(['color','color_cod'], axis=1)
```

Manejo de datos categóricos

También podemos usar get_dummies de PANDAS

```
pd.get_dummies(data=dfAux, columns=["color"], drop_first=True)
df= df.drop(['color_cod'], axis=1)
```

	nombre	edad	color	etiqueta	color_cod
0	M	30	Amarillo	Clase 1	0
1	P	28	Azul	Clase 2	1
2	J	21	Rojo	Clase 1	2