Machine learning y Deep learning con Python: Lectura 6

Ing. Pedro Rotta

Universidad de Piura - Vida Universitaria

Enero-2022

Procesamiento de datos

Hasta ahora, hemos trabajado algoritmos de machine learning con datos crudos. Sin embargo, algunas veces, es mejor procesar los datos, ya que la data real puede no estar limpia. El procesamiento de datos ayuda a mejorar el rendimiento de nuestros algoritmos y será importante analizar si es mejor tratar con data cruda o con data procesada.

Procesamiento de datos

Hasta ahora, hemos trabajado algoritmos de machine learning con datos crudos. Sin embargo, algunas veces, es mejor procesar los datos, ya que la data real puede no estar limpia. El procesamiento de datos ayuda a mejorar el rendimiento de nuestros algoritmos y será importante analizar si es mejor tratar con data cruda o con data procesada.

El procesamiento de datos cubre los siguientes items:

- Escalamiento de datos
- Estandarización
- Normalización
- Binarización
- Hot Encoding
- Ingresar datos faltantes
- Generar características polinómicas.

Escalamiento de datos

Es común encontrar conjuntos de datos, con diferentes escalas. Por ejemplo, características que tienen en la primera columna rangos entre 1000 a 9000, mientras que en otra rangos de 1-5.

Estas diferencias en el conjunto de datos, puede causar efectos adversos para ciertos modelos de machine learning, especialmente para los modelos lineales(OLS,Lasso,Ridge,etc).

Cuando se escala la data, se convierte la data al rango de [0,1].

Computarización

Para escalar datos, usamos de la librería sklearn **MinMaxScaler**:

```
from sklearn.preprocessing import MinMaxScaler
X = data
scaler1 = MinMaxScaler(feature_range = (0,1))
x_rescaled = scaler.fit_transform(X)
plotting
fig,ax = plt.subplots(1,2)
sns.displot(X,ax = ax[0])
ax[0].set_title("Original")
sns.displot(X_rescaled,ax = ax[1])
ax[0].set_title("Scaled data")
```

Ver problema 1.

Estandarización

Algunos algoritmos de machine learning asumen que el conjunto de datos a analizar está normalmente distribuido. (Logistic regression o Linear Regression).

Un conjunto de datos normalmente distribuido tiene su media en 0 y su desviación estándar en 1.

Aplicando una estandarización, el conjunto de datos pasa a transformar a las características o features del conjunto de datos, tal que tengan una distribución Gaussiana o normal con una media de0 y una desviación estándar de 1.

Computarización

Para estandarizar un conjunto de datos, usamos de la librería sklearn **StandardScaler**:

```
from sklearn.preprocessing import StandarScaler
X = data
scaler1 = StandarScaler().fit(X)
x_standarize = scaler1.transform(X)
plotting
fig,ax = plt.subplots(1,2)
sns.displot(X,ax = ax[0])
ax[0].set_title("Original")
sns.displot(X_rescaled,ax = ax[1])
ax[0].set_title("Standarized data")
```

Ver problema 2

Normalización

La normalización de datos implica transformar la data para que las observaciones en el conjunto de datos tengan una norma unitaria, es decir magnitud o tamaño de 1.

La norma de un vector es la raiz cuadrada de la suma de los cuadrados de los elementos de un vector. Normalizar el conjunto de datos, es útil para casos donde la data está dispersa(presencia de zeros) y el conjunto de datos tiene diferentes escalas.

Computarización

```
from sklearn.preprocessing import Normalizer
X = data
scaler2 = Normalizer().fit(X)
x_normalize = scaler2.transform(X)
plotting
fig,ax = plt.subplots(1,2)
sns.displot(X,ax = ax[0])
ax[0].set_title("Original")
sns.displot(X_normalize,ax = ax[1])
ax[0].set_title("Normalized data")
```

Ver problema 3.

Binarización

La binarización es una técnica de transformación que convierte un conjunto de datos en valores binarios mediante el establecimiento de un umbral o **threshold**.

Todos los valores por encima del umbral son fijados como 1, mientras que los valores por debajo del umbral, son fijados como 0. Esta técnica es útil para convertir algunas probabilidades en enteros. O transformar cierto feature en categoría binaria.

Computarización

```
from sklearn.preprocessing import Normalizer
X = data
scaler3 = Binarizer(threshold = m).fit(X)
x_binarized = scaler3.transform(X)
print(x_binarized)
```

Ver problema 4.

Hot Encoding

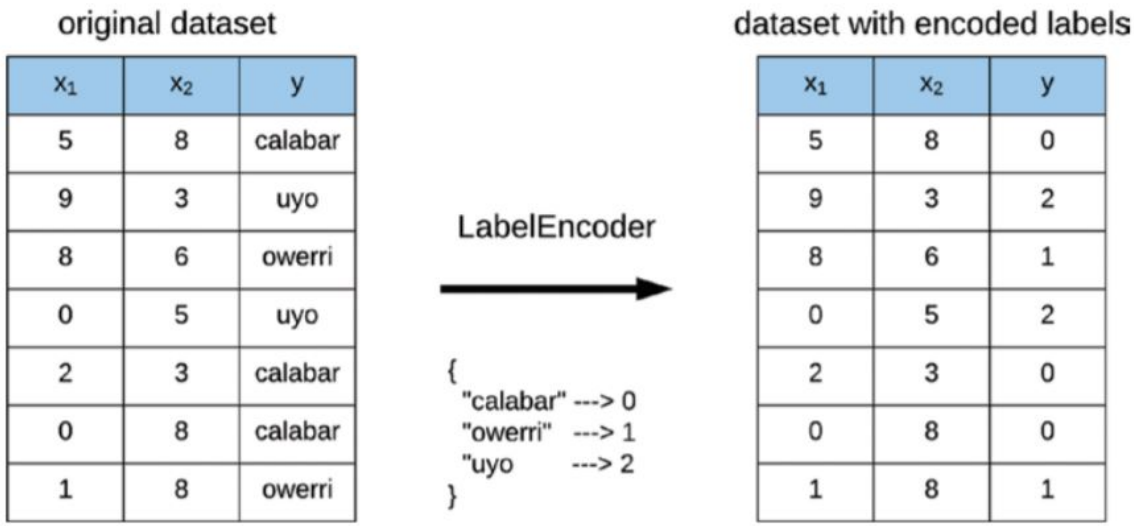
Si tienes un problema de clasificación, tendrás varias etiquetas y algunas veces, los conjuntos de datos, no traen etiquetas numéricas, sino etiquetas escritas. También en el caso de los features, cuando son datos categóricos, muchas veces no son datos numéricos.

El método de codificar datos, se refiere a transformar las variables categóricas textuales, en variables categóricas numéricas, para que la data sea más óptima para aplicar el algoritmo de machine learning.

Existen 2 métodos de codificación. Uno es para etiquetas **LabelEncoder** y otro para features **OneHotEncoder**.

LabelEncoder

Este método es típicamente usado cuando el objetivo de una variable se busca transformar a un vector de categorías numéricas. Este método transforma las categorías no numéricas, en numéricas, para un rango desde 0 hasta $n - 1$.



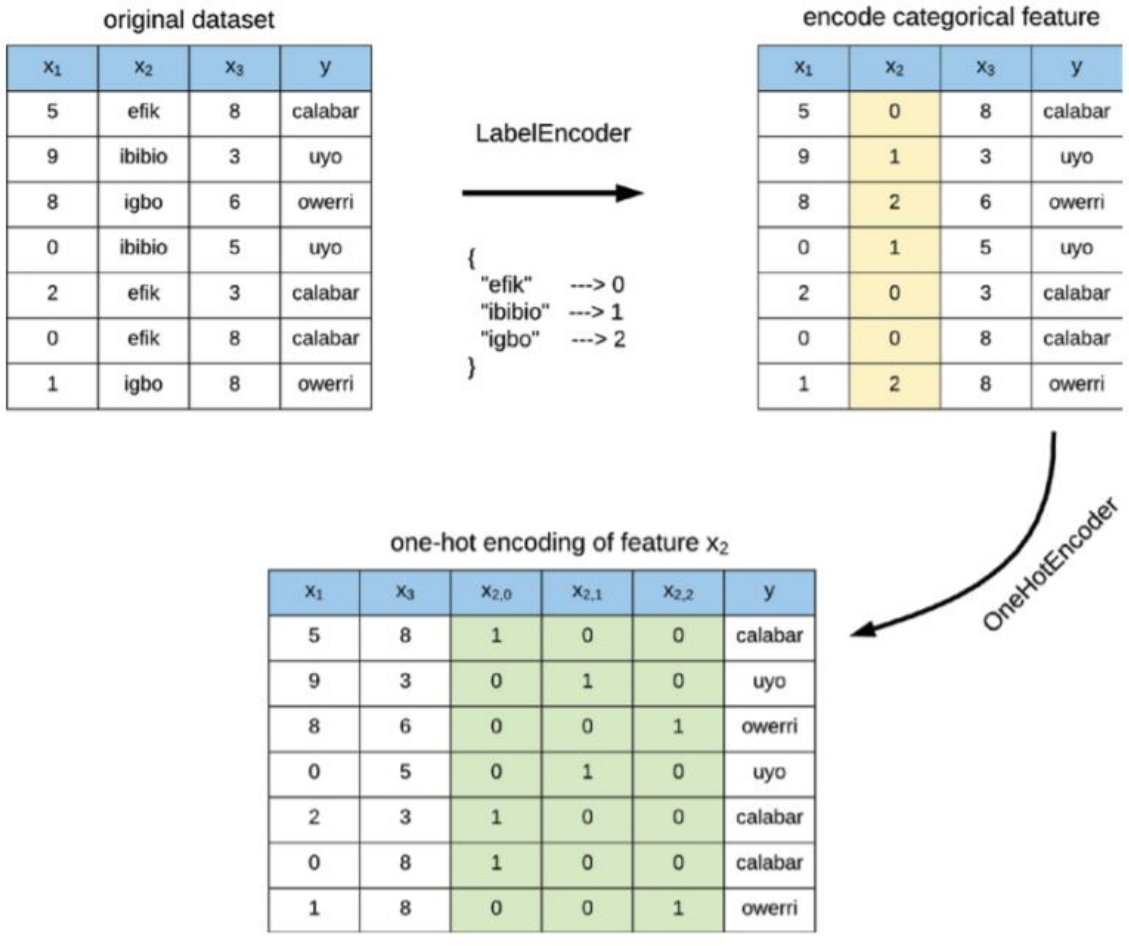
Computarización

```
from sklearn.preprocessing import LabelEncoder
y = data
encoder = LabelEncoder()
encode_y = encoder.fit_transform(y)
print(encode_y)
```

Ver problema 5.

One Hot encoder

One Hot encoder es usada para transformar datos categóricos en una matriz de enteros. Esta nueva matriz tiene en cada fila una sola combinación que representa el valor de la categoría.



x_1	x_2	x_3	y
5	efik	8	calabar
9	ibibio	3	uyo
8	igbo	6	owerri
0	ibibio	5	uyo
2	efik	3	calabar
0	efik	8	calabar
1	igbo	8	owerri

x_1	x_2	x_3	y
5	0	8	calabar
9	1	3	uyo
8	2	6	owerri
0	1	5	uyo
2	0	3	calabar
0	0	8	calabar
1	2	8	owerri

x_1	x_3	$x_{2,0}$	$x_{2,1}$	$x_{2,2}$	y
5	8	1	0	0	calabar
9	3	0	1	0	uyo
8	6	0	0	1	owerri
0	5	0	1	0	uyo
2	3	1	0	0	calabar
0	8	1	0	0	calabar
1	8	0	0	1	owerri

Computarización

```
from sklearn.preprocessing import OneHotEncoder
X = data.x
y = data.y
onehot = OneHotEncoder(handle_unknown = 'ignore')
encode_X = one_hot_encoder.fit_transform(x_encode)
encode_Xtotal = np.append(X,encode_X,axis = 1)
print(encode_Xtotal)
```

Ver problema 6.

Ingresando data faltante

A veces existen datos faltantes, estos datos pueden ser ingresados por medio del método **Imputer**. Para computarizarlo usamos:

```
from sklearn.impute import SimpleImputer
X = data.x
y = data.y
imputer = SimpleImputer(missing_values = nan,
strategy = 'mean')
data = imputer.fit_transform(X)
print(data)
```