María Fernanda Leandro Guillén

Universidad Castro Carazo

Portafolio de evidencias

Análisis de Datos

# Semana 6

**Resumen**

**Preprocesamiento de datos en Python**

El preprocesamiento de datos se refiere a los pasos aplicados para hacer los datos más adecuados para la minería de datos.

Los pasos utilizados para el preprocesamiento de datos suelen ser de dos categorías

Seleccionar los objetos de datos y los atributos para el análisis.

crear/modificar los atributos.

En este post voy a caminar a través de la implementación de los métodos de preprocesamiento de datos utilizando Python. Voy a cubrir lo siguiente, uno a la vez:

* La importación de las bibliotecas
* Importación del conjunto de datos
* Manejo de datos perdidos
* Manejo de datos categóricos
* Dividir el conjunto de datos en conjuntos de datos de entrenamiento y de prueba

**Importación de las bibliotecas**

# bibliotecas

import numpy as np # utilizada para manejar los números

import pandas as pd # utilizada para manejar el conjunto de datos

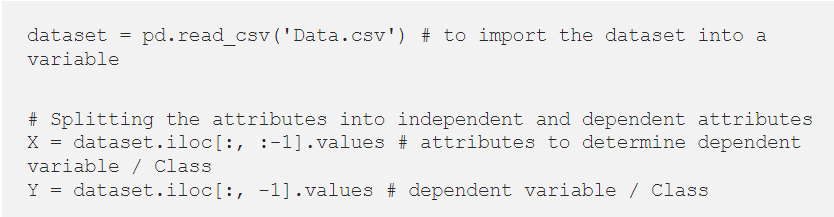
from sklearn.impute import SimpleImputer # utilizada para manejar los datos perdidos

from sklearn.preprocessing import LabelEncoder, OneHotEncoder # utilizados para codificar datos categóricos

from sklearn.model\_selection import train\_test\_split # utilizado para dividir los datos de entrenamiento y de prueba

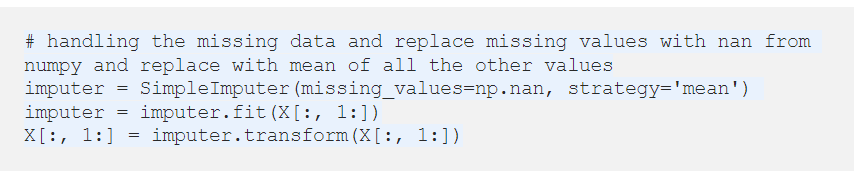
from sklearn.preprocessing import StandardScaler # utilizado para el escalado de características

**Importar el conjunto de datos**



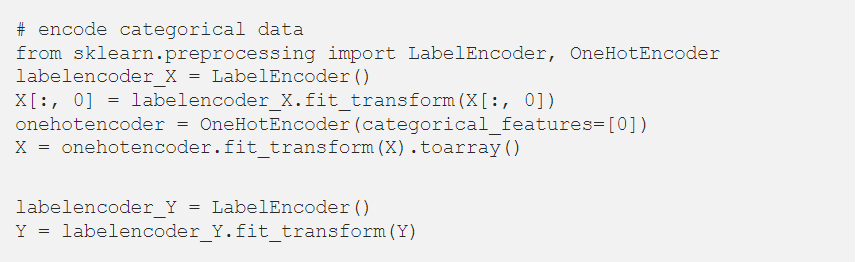
**Manejo de datos faltantes**

Bueno, la primera idea es eliminar las líneas en las observaciones donde hay algunos datos que faltan. Pero eso puede ser bastante peligroso porque imagine que este conjunto de datos contiene información crucial. Sería muy peligroso eliminar una observación. Así que tenemos que encontrar una mejor idea para manejar este problema. Y otra idea que es en realidad la idea más común para manejar los datos que faltan es tomar la media de las columnas.



**Manejo de datos categóricos**

Puedes adivinar que, dado que los modelos de machine learning se basan en ecuaciones matemáticas, puedes entender intuitivamente que causaría algún problema si mantenemos el texto aquí en las variables categóricas en las ecuaciones porque sólo querríamos números en las ecuaciones. Así que por eso necesitamos codificar las variables categóricas. Es decir, codificar el texto que tenemos aquí en números. Para hacer esto usamos el siguiente fragmento de código.



**Splitting the dataset into training and testing datasets**

Cualquier algoritmo de machine learning necesita ser probado para comprobar su precisión. Para ello, dividimos nuestro conjunto de datos en dos partes: conjunto de entrenamiento y conjunto de prueba. Como su propio nombre indica, utilizamos el conjunto de entrenamiento para que el algoritmo aprenda los comportamientos presentes en los datos y comprobamos la corrección del algoritmo probándolo en el conjunto de pruebas. En Python, lo hacemos de la siguiente manera

# dividiendo el conjunto de datos en conjunto de entrenamiento y conjunto de prueba

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=0)

En este caso, estamos tomando el conjunto de entrenamiento como el 80% del conjunto de datos original y el conjunto de prueba como el 20% del conjunto de datos original. Esta suele ser la proporción en la que se dividen. Sin embargo, a veces puede encontrarse con una proporción de 70-30% o 75-25%. Pero, no se quiere dividir al 50-50%. Esto puede llevar a un sobreajuste del modelo. Este tema es demasiado grande para ser cubierto en el mismo post. Lo cubriré en un futuro post. Por ahora, vamos a dividirlo en una proporción de 80-20%.

Usamos el escalado de características para convertir las diferentes escalas a una escala estándar para que sea más fácil para los algoritmos de Machine Learning. Hacemos esto en Python como sigue:

# feature scaling

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.transform(X\_test)

**El agrupamiento**

Agrupa los datos en contenedores (o depósitos), en el sentido de que reemplaza los valores contenidos en un intervalo pequeño con un único valor representativo para ese intervalo. A veces, el agrupamiento mejora la precisión en los modelos predictivos.

El agrupamiento de datos es un tipo de preprocesamiento de datos, un mecanismo que también incluye el manejo de [valores faltantes](https://towardsdatascience.com/data-preprocessing-with-python-pandas-part-1-missing-data-45e76b781993) , [formateo](https://towardsdatascience.com/data-processing-with-python-pandas-part-2-data-formatting-710c2eafa426) , [normalización](https://towardsdatascience.com/data-preprocessing-with-python-pandas-part-3-normalisation-5b5392d27673) y [estandarización](https://towardsdatascience.com/data-preprocessing-with-python-pandas-part-4-standardization-ccd5b1608f1c) .

El agrupamiento se puede aplicar para convertir valores numéricos en valores numéricos categóricos o de muestra (cuantificación).

Convertir numérico a categórico incluye agrupamiento por distancia y agrupamiento por frecuencia

reducir valores numéricos incluye cuantificación (o muestreo).

El binning es una técnica para el suavizado de datos. El suavizado de datos se emplea para eliminar el ruido de los datos. Tres técnicas para el suavizado de datos:

* agrupar
* regresión
* análisis de valores atípicos.

**Importación de datos**

Explotamos la pandas biblioteca para importar el conjunto de datos y lo transformamos en un marco de datos a través de la read\_csv()función.

importar pandas como pd

df = pd.read\_csv('cupcake.csv')

df.head(5)

**Clasificación por distancia**

En este caso definimos los bordes de cada bin. En Python pandas, el binning por distancia se logra mediante la cut()función.

Podemos calcular el rango de cada intervalo, es decir, el valor mínimo y máximo de cada intervalo. Como tenemos 3 grupos, necesitamos 4 aristas de intervalos (bins):

pequeño — (borde1, borde2)

medio — (borde2, borde3)

grande - (borde3, borde4)

Podemos usar la linspace()función del numpypaquete para calcular los 4 contenedores, igualmente distribuidos.

**Clasificación por frecuencia**

El agrupamiento por frecuencia calcula el tamaño de cada contenedor para que cada contenedor contenga (casi) el mismo número de observaciones, pero el rango del contenedor variará. Podemos usar la pandas qcut()función Python. Podemos establecer el precisionparámetro para definir el número de puntos decimales.

df['bin\_qcut'] = pd.qcut(df['Cupcake'], q=3, precisión=1, etiquetas=etiquetas)

**Muestreo**

El muestreo es otra técnica de agrupación de datos. Permite reducir el número de muestras, agrupando valores similares o valores contiguos. Hay tres enfoques para realizar el muestreo:

1. por medios bin: cada valor en un bin se reemplaza por el valor medio del bin.
2. por bin mediana: cada valor bin se reemplaza por su valor bin mediana.
3. por límite de bin: cada valor de bin se reemplaza por el valor de límite más plt.plot(df['sampled\_cupcake'], color='red', label='sampled')