Preprocesamiento de datos en Python

El preprocesamiento de datos se refiere a los pasos aplicados para hacer que los datos sean más adecuados para la minería de datos. Los pasos utilizados para el preprocesamiento de datos generalmente se dividen en dos categorías:

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

2. Creando/cambiando los atributos.

Importación de las bibliotecas

```
# bibliotecas

importar numpy como np # usado para manejar números
importar pandas como pd # usado para manejar el conjunto de datos

from sklearn.impute import SimpleImputer # usado para manejar datos
faltantes

de sklearn.preprocessing import LabelEncoder, OneHotEncoder #
utilizado para codificar datos categóricos

from sklearn.model_selection import train_test_split # utilizado
para dividir datos de entrenamiento y prueba

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

Importación del conjunto de datos

Utilizando pandas se aplica de la siguiente manera:

```
dataset = pd.read_csv('Data.csv') # para importar el conjunto de
datos a una
variable


# Dividir los atributos en atributos independientes y dependientes
X = conjunto de datos.iloc[:, :-1].valores # atributos para
determinar la variable dependiente / Clase
Y = conjunto de datos.iloc[:, -1].valores # variable dependiente /
Clase
```

Manejo de datos faltantes

Se debe tener cuidado a la hora de manipular los datos ya que se pueden eliminar datos importantes para la investigación, por lo tanto, se requiere mucha cautela

```
# manejar los datos faltantes y reemplazar los valores faltantes
con nan de numpy y reemplazar con la media de todos los demás
valores
imputer = SimpleImputer(missing_values=np.nan, estrategia='mean')
imputer = imputer.fit(X[:, 1: ])
X[:, 1:] = imputador.transformar(X[:, 1:])
```

Manejo de Datos Categóricos

```
# codificar datos categóricos
de sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
onehotencoder = OneHotEncoder(categorical_features=[0])
X = onehotencoder.fit_transform (X).toarray()


labelencoder_Y = LabelEncoder()
Y = labelencoder_Y.fit_transform(Y)
```

Dividir el conjunto de datos en conjuntos de datos de entrenamiento y prueba

Cualquier algoritmo de aprendizaje automático debe probarse para verificar su precisión. Para hacer eso, dividimos nuestro conjunto de datos en dos partes: conjunto de **entrenamiento** y conjunto de **prueba.** Como sugiere el propio nombre, usamos el conjunto de entrenamiento para hacer que el

algoritmo aprenda los comportamientos presentes en los datos y verifique la corrección del algoritmo mediante la prueba en el conjunto de prueba.

```
# dividir 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)
```

Escalado de características

Utilizamos el escalado de funciones para convertir diferentes escalas en una escala estándar para facilitar los algoritmos de aprendizaje automático.

```
# escalado de funciones

sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Preprocesamiento de datos con Python Pandas (Binning)

El agrupamiento (o depósito) de datos 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.

El agrupamiento de datos es un tipo de preprocesamiento de datos, un mecanismo que también incluye el manejo de valores faltantes , formateo , normalización y estandarización .

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:

- Agrupamiento
- Regresión
- Análisis de valores atípicos.

Importación de datos

```
importar pandas como pd
df = pd.read_csv('cupcake.csv')
df.head(5)
```

Clasificación por distancia

En Python pandas, el binning por distancia se logra mediante la **cut() función.**

Calculamos el rango del intervalo como la diferencia entre el valor máximo y mínimo, con las funciones min()y max().

```
min_value = df['Cupcake'].min()
max_value = df['Cupcake'].max()
print(min_value)
print(max_value)
```

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

```
importar numpy como np
bins = np.linspace(min_value,max_value,4)
bins
```

Podemos graficar la distribución de valores usando la `hist()` función del `matplotlib` paquete.

```
importar matplotlib.pyplot como plt

plt.hist(df['contenedores'], contenedores = 3)
```

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.

```
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.

```
de scipy.stats import binned_statistic
x_data = np.arange(0, len(df))
y_data = df['Cupcake']
x_bins,bin_edges, misc = binned_statistic(y_data,x_data,
statistic="median", bins=2)
```

```
bin_intervals = pd.IntervalIndex.from_arrays(bin_edges[:-1],
bin_edges[1:])
```

```
def set_to_median(x, bin_intervals):
    para intervalo en bin_intervals:
        si x en intervalo:
            return intervalo.mid
```

```
df['sampled_cupcake'] = df['Cupcake'].apply(lambda x:
set_to_median(x, bin_intervals))
```

```
plt.plot(df['Cupcake'], label='original')
plt.plot(df['sampled_cupcake'], color='red', label='sampled')
plt.legend()
plt.show()
```