

NOMBRE DE LA PRÁCTICA	Preparacion del datadet			NO.	1
ASIGNATURA:	SIMULACION	CARRERA:	INGENIERÍA EN SISTEMAS COMPUTACIONALES	DURACIÓN DE LA PRÁCTICA (HRS)	2 HORAS

**NOMBRE DEL ALUMNO:**

**GRUPO: 3502**

**I. Competencia(s) específica(s):**

- 

**II. Lugar de realización de la práctica (laboratorio, taller, aula u otro):**

- Salon de clase

**III. Material empleado:**

equipo de computo

## PRACTICA

### Preparacion del DataSet

En este notebook se muestran las tecnicas mas utilizadas para trasformar el DataSet.

#### DataSet

**Description**

NSL-KDD is a data set suggested to solve some of the inherent problems of the KDD'99 data set which are mentioned in [1]. Although, this new version of the KDD data set still suffers from some of the problems discussed by McHugh and may not be a perfect representative of existing real networks, because of the lack of public data sets for network-based IDSs, we believe it still can be applied as an effective benchmark data set to help researchers compare different intrusion detection methods.

Furthermore, the number of records in the NSL-KDD train and test sets are reasonable. This advantage makes it affordable to run the experiments on the complete set without the need to randomly select a small portion. Consequently, evaluation results of different research work will be consistent and comparable.

**Ficheros de Datos**

- KDDTrain+.ARFF: The full NSL-KDD train set with binary labels in ARFF format.
- KDDTrain+.TXT: The full NSL-KDD train set including attack-type labels and difficulty level in CSV format.
- KDDTrain+\_20Percent.ARFF: A 20% subset of the KDDTrain+.arff file. \* KDDTrain+\_20Percent.TXT: A 20% subset of the KDDTrain+.txt file.
- KDDTest+.ARFF: The full NSL-KDD test set with binary labels in format.

La "Preparación del DataSet" se refiere al proceso de acondicionamiento y transformación de un conjunto de datos antes de utilizarlo para entrenar un modelo de aprendizaje automático. Este proceso es esencial para garantizar que los datos sean adecuados y estén en la forma correcta

para que los algoritmos de machine learning puedan aprender patrones y realizar predicciones de manera efectiva.

## Ficheros de Datos

- KDDTrain+.ARFF: The full NSL-KDD train set with binary labels in ARFF format.
  - KDDTrain+.TXT: The full NSL-KDD train set including attack-type labels and difficulty level in CSV format.
  - KDDTrain+\_20Percent.ARFF: A 20% subset of the KDDTrain+.arff file. \* KDDTrain+\_20Percent.TXT: A 20% subset of the KDDTrain+.txt file.
  - KDDTest+.ARFF: The full NSL-KDD test set with binary labels in format.
  - KDDTest+.TXT: The full NSL-KDD test set including attack- labels and difficulty level in CSV format.
  - KDDTest-21.ARFF: A subset of the KDDTest+.arff file does not include records with difficulty level of 21 out of 21.
  - KDDTest-21.TXT: A subset of the KDDTest+.txt which does not include records with difficulty level of 21 out of 21.

## Descarga de los ficheros de datos.

<https://www.unb.ca/cic/datasets/index.html>

## Referencias adicionales sobre el conjunto de datos

M. Tavallaei, E. Bagheri, W. Lu, and A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set," Submitted to Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), 2009.

## Imports.

```
In [1]: 1 import arff
        2 import pandas as pd
        3 import numpy as np
        4 from sklearn.model_selection import train_test_split
```

Los ficheros de datos están relacionados con el conjunto de datos NSL-KDD, que se utiliza comúnmente en el ámbito de la detección de intrusiones en redes

## Imports.

```
In [1]: 1 import arff
2 import pandas as pd
3 import numpy as np
4 from sklearn.model_selection import train_test_split
```

## Funciones Auxiliares.

```
In [2]: 1 def load_kdd_dataset(data_path):
2     """Lectura del DataSet NSL-KDD."""
3     with open(data_path, 'r') as train_set:
4         dataset = arff.load(train_set)
5         attributes = [attr[0] for attr in dataset["attributes"]]
6         return pd.DataFrame(dataset["data"], columns=attributes)

In [3]: 1 # Construcción de una función que realice un particionado completo
2 def train_val_test_split(df, rstate=42, shuffle=True, stratify=None):
3     sstrat = df[stratify] if stratify else None
4     train_set, test_set = train_test_split(
5         df, test_size=0.4, random_state=rstate, shuffle=shuffle, stratify=sstrat)
6     strat=test_set[stratify] if stratify else None
7     val_set, test_set=train_test_split(
8         test_set, test_size = 0.5, random_state = rstate, shuffle=shuffle, stratify =strat)
9     return(train_set, val_set, test_set)
```

## 1.- Preparacion del DataSet

```
In [4]: 1 df = load_kdd_dataset("datasets/NSL-KDD/KDDTrain+.arff")
```

```
In [5]: 1 df
```

```
Out[5]:
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_f
0	0.0	tcp	ftp_data	SF	491.0	0.0	0	0.0	0.0	0.0	...	25.0	0.17	
1	0.0	udp	other	SF	146.0	0.0	0	0.0	0.0	0.0	...	1.0	0.00	
2	0.0	tcp	private	SO	0.0	0.0	0	0.0	0.0	0.0	...	26.0	0.10	
3	0.0	tcp	http	SF	232.0	8153.0	0	0.0	0.0	0.0	...	255.0	1.00	
4	0.0	tcp	http	SF	199.0	420.0	0	0.0	0.0	0.0	...	255.0	1.00	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
125968	0.0	tcp	private	SO	0.0	0.0	0	0.0	0.0	0.0	...	25.0	0.10	
125969	8.0	udp	private	SF	105.0	145.0	0	0.0	0.0	0.0	...	244.0	0.96	
125970	0.0	tcp	smtp	SF	2231.0	384.0	0	0.0	0.0	0.0	...	30.0	0.12	
125971	0.0	tcp	klogin	SO	0.0	0.0	0	0.0	0.0	0.0	...	8.0	0.03	
125972	0.0	tcp	ftp_data	SF	151.0	0.0	0	0.0	0.0	0.0	...	77.0	0.30	

125973 rows x 42 columns

función llamada `load_kdd_dataset` para cargar un conjunto de datos desde un archivo ARFF.

## Division del DataSet

```
In [6]: 1 train_set, val_set, test_set = train_val_test_split(df, stratify='protocol_type')
```

```
In [7]: 1 print("Longitud del training Set", len(train_set))
2 print("Longitud del validation Set", len(val_set))
3 print("Longitud del Test Set", len(test_set))
```

```
Longitud del training Set 75583
Longitud del validation Set 25195
Longitud del Test Set 25195
```

## 3.- Limpieza de los Datos.

Antes de comenzar a ingerir datos, es necesario recuperar el DataSet Raw y tambien es necesario separar las etiquetas del resto de los datos no necesariamente se quiere aplicar las mismas transformaciones en ambos conjuntos

```
In [8]: 1 # separar las características de entrada de las características de salida
2 x_train = train_set.drop("class", axis = 1 )
3 y_train = train_set["class"].copy()
```

```
In [9]: 1 # para ilustrar esta sección se añaden algunos valores nulos.
2 # a algunas características del DataSet.
3 x_train.loc[(x_train["src_bytes"] > 400) & (x_train["src_bytes"] < 800), "src_bytes"] = np.nan
4 x_train.loc[(x_train["dst_bytes"] > 500) & (x_train["dst_bytes"] < 2000), "dst_bytes"] = np.nan
```

```
In [10]: 1 x_train
```

Realiza la división de un conjunto de datos (df) en conjuntos de entrenamiento, validación y prueba utilizando la función `train_val_test_split`.

Asi mismo se separan las características de entrada (`x_train`) y las características de salida (`y_train`) del conjunto de entrenamiento (`train_set`).

```
In [11]: 1 # comprobar si existe algun atributo con valores nulos
2 x_train.isna().any()
```

```
Out[11]: duration          False
protocol_type            False
service                  False
flag                     False
src_bytes                 True
dst_bytes                 True
land                     False
wrong_fragment            False
urgent                    False
hot                       False
num_failed_logins         False
logged_in                 False
num_compromised           False
root_shell                False
su_attempted              False
num_root                  False
num_file_creations        False
num_shells                 False
num_access_files          False
num_outbound_cmds         False
is_host_login             False
is_guest_login            False
count                     False
srv_count                 False
error_rate                False
srv_error_rate            False
error_rate                False
```

Se verifica si hay algún atributo (columna) en el conjunto de características de entrada `x_train` que contiene valores nulos.

```
dtype: bool

In [12]: 1 # Selecciona la fila que contienen valores nulos.
          2 filas_valores_nulos = [x_train.isnull().any(axis =1)]
          3 filas_valores_nulos

Out[12]: [113467    True
          31899    False
          108116    True
           89913    False
          106319    False
           ...
          64559    False
          67272    True
          32452    False
          112657    False
          99030    False
          Length: 75583, dtype: bool]

Opcion 1: Eliminar las filas con valores nulos.

In [13]: 1 # Copiar el DataSet para no alterar el original
          2 x_train_copy = x_train.copy()

In [14]: 1 # Eliminar las filas con valores nulos
          2 x_train_copy.dropna(subset=["src_bytes", "dst_bytes"], inplace = True)
          3 x_train_copy

Out[14]: duration  protocol_type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_count  dst_host_srv_count  dst_host_sai
```

seleccionar las filas en el DataFrame `x_train` que contienen al menos un valor nulo así mismo creamos una copia del DataFrame `x_train` y asignándolo a la variable `x_train_copy`. Esto se hace para asegurarse de que cualquier modificación realizada en `x_train_copy` no afecte al DataFrame original `x_train`

utilizamos el método `dropna` para eliminar las filas que contienen valores nulos en las columnas "`src_bytes`" y "`dst_bytes`" de `x_train_copy`



```
In [15]: 1 # Contar el numero de filas eliminadas
2 print(" El numero de filas eliminadas es ", len(x_train)-len(x_train_copy))
```

El numero de filas eliminadas es 9886

Opcion 2 : eliminar los atributos con valores nulos.

```
In [16]: 1 # Copiar el DataSet para no alterar el original
2 x_train_copy = x_train.copy()
```

```
In [17]: 1 # Eliminar los atributos con valores nulos
2 x_train_copy.drop(["src_bytes", "dst_bytes"], axis = 1, inplace = True)
3 x_train_copy
```

```
Out[17]:
```

	duration	protocol_type	service	flag	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	...	dst_host_count	dst_host_srv_count	dst_h
113467	0.0	tcp	http	SF	0	0.0	0.0	0.0	0.0	1	...	9.0	255.0	
31899	0.0	tcp	private	S0	0	0.0	0.0	0.0	0.0	0	...	255.0	4.0	
108116	0.0	tcp	http	SF	0	0.0	0.0	0.0	0.0	1	...	39.0	255.0	
89913	0.0	tcp	private	S0	0	0.0	0.0	0.0	0.0	0	...	255.0	15.0	
106319	0.0	icmp	eco_i	SF	0	0.0	0.0	0.0	0.0	0	...	2.0	7.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
64559	0.0	tcp	systat	S0	0	0.0	0.0	0.0	0.0	0	...	255.0	20.0	
67272	0.0	tcp	http	SF	0	0.0	0.0	0.0	0.0	1	...	119.0	255.0	
32452	3.0	tcp	smtp	SF	0	0.0	0.0	0.0	0.0	1	...	111.0	155.0	
112657	0.0	tcp	http	SF	0	0.0	0.0	0.0	0.0	1	...	255.0	255.0	

calculando el número de filas eliminadas al restar la longitud del DataFrame original `x_train` menos la longitud del DataFrame después de la eliminación `x_train_copy`

sto crea una copia del DataFrame `x_train` y asigna esa copia a la variable `x_train_copy`. Puedes realizar modificaciones y experimentos en `x_train_copy` sin afectar al conjunto de datos original `x_train` se eliminan las columnas "src\_bytes" y "dst\_bytes" del DataFrame `x_train_copy` utilizando el método `drop`

## Opcion 3: Rellenar los valores nulos con un valor determinado.

```
In [19]: 1 # Copiar el DataSet para no alterar el original
2 x_train_copy = x_train.copy()
```

```
In [20]: 1
2 # Rellenar los valores nulos con la media de los valores del atributo
3 media_srcbytes = x_train_copy["src_bytes"].mean()
4 media_dstbytes = x_train_copy["dst_bytes"].mean()
5
6 x_train_copy["src_bytes"].fillna(media_srcbytes, inplace = True)
7 x_train_copy["dst_bytes"].fillna(media_dstbytes, inplace = True)
8
9 x_train_copy
```

```
Out[20]:
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_count	dst_host_srv_count	dst_host...
113467	0.0	tcp	http	SF	66914.530762	53508.000000	0	0.0	0.0	0.0	...	9.0	255.0	
31899	0.0	tcp	private	S0	0.000000	0.000000	0	0.0	0.0	0.0	...	255.0	4.0	
108116	0.0	tcp	http	SF	304.000000	9181.334754	0	0.0	0.0	0.0	...	39.0	255.0	
89913	0.0	tcp	private	S0	0.000000	0.000000	0	0.0	0.0	0.0	...	255.0	15.0	
106319	0.0	icmp	eco_i	SF	8.000000	0.000000	0	0.0	0.0	0.0	...	2.0	7.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
64559	0.0	tcp	systat	S0	0.000000	0.000000	0	0.0	0.0	0.0	...	255.0	20.0	
67272	0.0	tcp	http	SF	210.000000	9181.334754	0	0.0	0.0	0.0	...	119.0	255.0	

utilizando un imputador (como el SimpleImputer de scikit-learn) para rellenar los valores nulos en tu conjunto de datos x\_train\_copy

```
In [21]: 1 # Copiar el DataSet para no alterar el original
2 x_train_copy = x_train.copy()
```

```
In [22]: 1 # Un valor muy alto en el atributo puede disparar la media.
2 # es necesario rellenar los valores con la mediana.
3 # Rellenar los valores nulos con la media de los valores del atributo
4 mediana_srcbytes = x_train_copy["src_bytes"].median()
5 mediana_dstbytes = x_train_copy["dst_bytes"].median()
6
7 x_train_copy["src_bytes"].fillna(mediana_srcbytes, inplace = True)
8 x_train_copy["dst_bytes"].fillna(mediana_dstbytes, inplace = True)
9
10 x_train_copy
```

```
Out[22]:
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_count	dst_host_srv_count	dst_host_sam...
113467	0.0	tcp	http	SF	43.0	53508.0	0	0.0	0.0	0.0	...	9.0	255.0	
31899	0.0	tcp	private	S0	0.0	0.0	0	0.0	0.0	0.0	...	255.0	4.0	
108116	0.0	tcp	http	SF	304.0	0.0	0	0.0	0.0	0.0	...	39.0	255.0	
89913	0.0	tcp	private	S0	0.0	0.0	0	0.0	0.0	0.0	...	255.0	15.0	
106319	0.0	icmp	eco_i	SF	8.0	0.0	0	0.0	0.0	0.0	...	2.0	7.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
64559	0.0	tcp	systat	S0	0.0	0.0	0	0.0	0.0	0.0	...	255.0	20.0	
67272	0.0	tcp	http	SF	210.0	0.0	0	0.0	0.0	0.0	...	119.0	255.0	
32452	3.0	tcp	smtp	SF	889.0	328.0	0	0.0	0.0	0.0	...	111.0	155.0	
112657	0.0	tcp	http	SF	284.0	444.0	0	0.0	0.0	0.0	...	255.0	255.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

llenando los valores nulos en las columnas "src\_bytes" y "dst\_bytes" con las medianas respectivas de esas columnas.

## Existen otra alternativa para la opcion 3 que consiste en usar la clase imputer de Sklearn

```
In [23]: 1 # Copiar el DataSet para no alterar el original
          2 x_train_copy = x_train.copy()

In [24]: 1 from sklearn.impute import SimpleImputer
          2
          3 imputer = SimpleImputer(strategy = "median")

In [25]: 1 # La clase imputer no admite valores categoricos eliminar los atributos categoricos
          2 x_train_copy_num = x_train_copy.select_dtypes(exclude = ['object'])
          3 x_train_copy_num.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 75583 entries, 113467 to 99030
Data columns (total 34 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   duration              75583 non-null  float64
1   src_bytes              73696 non-null  float64
2   dst_bytes              67572 non-null  float64
3   wrong_fragment         75583 non-null  float64
4   urgent                 75583 non-null  float64
5   hot                    75583 non-null  float64
6   num_failed_logins      75583 non-null  float64
7   num_compromised        75583 non-null  float64
8   root_shell             75583 non-null  float64
9   su_attempted           75583 non-null  float64
```

- `from sklearn.impute import SimpleImputer`: Importa la clase `SimpleImputer` desde el módulo `impute` de `scikit-learn`. Esta clase se utiliza para imputar valores nulos en un conjunto de datos.
- `imputer = SimpleImputer(strategy="median")`: Crea una instancia de `SimpleImputer` con la estrategia de imputación establecida como "median". La estrategia "median" significa que los valores nulos se rellenarán con las medianas de las columnas correspondientes.

Se seleccionan las columnas numéricas del conjunto de datos `x_train_copy` y eliminando aquellas que tienen tipos de datos categóricos ('object').



```
In [26]: 1 # Se le tiene que proporcionar los atributos numericos para que calcule los valores
        2 imputer.fit(x_train_copy_num)
```

Out[26]: SimpleImputer(strategy='median')

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.  
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [27]: 1 #rellenar los valores nulos
        2 x_train_copy_num_nonan = imputer.transform(x_train_copy_num)
```

```
In [28]: 1 # Transformar el resultado a un DataFrame de pandas
        2 x_train_copy = pd.DataFrame(x_train_copy_num_nonan, columns = x_train_copy_num.columns)
```

```
In [29]: 1 x_train_copy.head(10)
```

```
Out[29]:
```

	duration	src_bytes	dst_bytes	wrong_fragment	urgent	hot	num_failed_logins	num_compromised	root_shell	su_attempted	...	dst_host_count	dst_host_
0	0.0	43.0	53508.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	9.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	255.0	
2	0.0	304.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	39.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	255.0	
4	0.0	8.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	2.0	
5	0.0	46.0	139.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	255.0	
6	0.0	1790.0	363.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	141.0	

método `fit` del imputador (`SimpleImputer`) para calcular los valores necesarios basándote en las columnas numéricas del conjunto de datos `x_train_copy_num`. Estamos utilizando `x_train_copy_imputed`, que es el resultado de la transformación del imputador

## APIs de Sklearn

Antes de continuar es necesario hacer una pequeña reseña sobre como funcionan las APIs de sklearn:

- **Estimators:** Cualquier objeto que pueda estimar cualquier parametro
  - El propio estimator se forma mediante el metodo `fit()`, que siempre toma un data set como argumento
  - cualquier otro parametro de este metodo es un hipermetro
- **Transformers:** Son estimadores capaces de transformar el DataSet (como imputer)
  - La transformacion se realiza mediante el metodo `transform()`.
  - Reciben un DataSet como parametro de entrada
- **Predictors** Son estimadores capaces de realizar predicciones
  - La prediccion se realiza por el metodo `predict()`.
  - Reciben un DataSet como entrada
  - Retornan un DataSet con las predicciones
  - Tienen un metodo `score()` para evaluar el resultado de la prediccion.

## 4.- Transformacion de atributos categoricos a numericos.

Antes de comenzar, a recuperar el DataSet el DataSet limpio es necesario separar las etiquetas del resto de los datos, no necesariamente se quiere aplicar las mismas transformaciones en ambos conjuntos.

```
In [30]: 1 # Copiar el DataSet para no alterar el original
        2 x_train = train_set.drop("class", axis = 1)
        3 y_train = train_set["class"].copy()
```

APIs de scikit-learn" se hace referencia a las interfaces de programación de aplicaciones (API, por sus siglas en inglés) proporcionadas por la biblioteca scikit-learn (sklearn). Scikit-learn es una biblioteca de machine learning en Python que ofrece herramientas simples y eficientes para análisis predictivo de datos.

```
In [32]: 1 protocol_type = x_train['protocol_type']
          2 protocol_type_encoded, categorias = protocol_type.factorize()

In [33]: 1 #Muestra por pantalla como sehan codificado
          2 for i in range(10):
          3     print(protocol_type.iloc[i], "=", protocol_type_encoded[i])

tcp = 0
tcp = 0
tcp = 0
tcp = 0
tcp = 1
tcp = 2
tcp = 0
tcp = 0
tcp = 0
tcp = 0
```

```
In [34]: 1 print(categorias)

Index(['tcp', 'icmp', 'udp'], dtype='object')
```

Se esta utilizando la función `factorize` de pandas para convertir las etiquetas categóricas en valores numéricos asi mismo se esta mostrando cómo se han codificado las primeras 10 filas de la columna 'protocol\_type'.

## Ordinal Encoding.

Realiza la misma codificación que el metodo `factorize` de pandas

```
In [35]: 1 from sklearn.preprocessing import OrdinalEncoder
          2
          3 protocol_type = x_train[['protocol_type']]
          4
          5 ordinal_encoder = OrdinalEncoder()
          6 protocol_type_encoded = ordinal_encoder.fit_transform(protocol_type)

In [36]: 1 # Mostrar por pantalla como se han modificado
          2 for i in range(10):
          3     print( protocol_type['protocol_type'].iloc[i], "=", protocol_type_encoded[i])

tcp = [1.]
tcp = [1.]
tcp = [1.]
tcp = [1.]
icmp = [0.]
udp = [2.]
tcp = [1.]
tcp = [1.]
tcp = [1.]
tcp = [1.]
```

```
In [37]: 1 print(ordinal_encoder.categories_)  
[array(['icmp', 'tcp', 'udp'], dtype=object)]
```

El problema de este tipo de codificación radica en que ciertos algoritmos de ML que funcionan midiendo la similitud de dos puntos por distancia, van a considerar que 1 está más cerca del 2, que del 3, y en este caso (para estos valores categóricos), no tiene sentido por ello se utilizan otros métodos de categorización como por ejemplo, OneHot Encoding

## OneHot Encoding

Genera para cada categoría del atributo categórico una matriz binaria que representa el valor

```
In [38]: 1 #Espace la matriz, almacenando la posición de los valores que no son '0' para ahorrar memoria.  
2 from sklearn.preprocessing import OneHotEncoder  
3  
4 protocol_type = x_train[['protocol_type']]  
5  
6 oh_encoder = OneHotEncoder()  
7 protocol_type_oh = oh_encoder.fit_transform(protocol_type)  
8 protocol_type_oh
```

```
Out[38]: <75583x3 sparse matrix of type '<class 'numpy.float64'>' with 75583 stored elements in Compressed Sparse Row format>
```

```
In [39]: 1 # Convertir Espase Matrix a un array de Numpy  
2 protocol_type_oh.toarray()
```

el `OrdinalEncoder` de `scikit-learn` para codificar las variables categóricas en tu conjunto de datos `x_train`

```
In [39]: 1 # Convertir Espase Matrix a un array de Numpy  
2 protocol_type_oh.toarray()
```

```
Out[39]: array([[0., 1., 0.],  
 [0., 1., 0.],  
 [0., 1., 0.],  
 ...,  
 [0., 1., 0.],  
 [0., 1., 0.],  
 [0., 1., 0.]])
```

```
In [40]: 1 # Mostrar en pantalla como se ha codificado  
2 for i in range(10):  
3     print(protocol_type["protocol_type"].iloc[i], "=", protocol_type_oh.toarray()[i])
```

```
tcp = [0. 1. 0.]  
tcp = [0. 1. 0.]  
tcp = [0. 1. 0.]  
tcp = [0. 1. 0.]  
icmp = [1. 0. 0.]  
udp = [0. 0. 1.]  
tcp = [0. 1. 0.]  
tcp = [0. 1. 0.]  
tcp = [0. 1. 0.]  
tcp = [0. 1. 0.]
```

```
In [41]: 1 print(ordinal_encoder.categories_)
```

el método `toarray()` para convertir la matriz dispersa (sparse matrix) `protocol_type_oh` en un array de NumPy. Así mismo muestra cómo se ha codificado la columna 'protocol\_type' en las primeras 10 filas después de aplicar la codificación one-hot



```
In [41]: 1 print (ordinal_encoder.categories_)  
[array(['icmp', 'tcp', 'udp'], dtype=object)]
```

En muchas ocasiones al particionar el DataSet o al realizar una prediccion con nuevos ejemplos aparecen nuevos valores para determinadas categorias que produzcan un error en la funcion **transfor()**. La clase OneHotEncoding proporciona el parametro **handle\_unknown** ya sea para generar un error o ignorar una caracteristica categorica desconocida que este presente durante la transformacion (el valor predeterminado es lanzar un error). Cuando este parametro se establece en "ignorar" y se encuentra una categoria desconocida durante la transformacion, las columnas codificadas resultantes para esta caracteristica seran todos ceros en la transformacion inversa una categoria desconocida se denotara como None.

```
In [42]: 1 oh_encoder = OneHotEncoder(handle_unknown = 'ignore')
```

## Get Dummies

Get Dummies es un metodo sencillo de utilizar que permite aplicar One-HOT Encoding a un data frame de pandas

```
In [43]: 1 pd.get_dummies(x_train['protocol_type'])
```

```
Out[43]:
```

	icmp	tcp	udp
113467	0	1	0
31899	0	1	0

OrdinalEncoder, puedes acceder a las categorías originales utilizando el atributo `categories`  
OneHotEncoder con el parámetro `handle_unknown='ignore'`, estás especificando cómo manejar categorías desconocidas durante la transformación

## 5.- Escalado del DataSet.

Antes de comenzar, vamos a recuperar el DataSet limpio y separar las etiquetas del resto de los datos no necesariamente se quiere aplicar las mismas transformaciones en ambos conjuntos.

```
In [44]: 1 x_train = train_set.drop("class", axis = 1)  
2 y_train = train_set["class"].copy()
```

Por norma general los algoritmos de ML no se comportan adecuadamente si los valores de las características que reciben como entrada se encuentran en rangos muy dispares por ello se utilizan distintas tecnicas de escalado. **Importante:** Tener en cuenta que estos mecanismos de escalado no deben aplicarse sobre etiquetas.

**NORMALIZACION :** Los valores del atributo se escalan para adquirir un valor entre 0 y 1. **ESTANDARIZACION:** Los valores del atributo se escalan y reciben un valor similar pero no se encuentra dentro de un rango

Es importante que para probar estos valores se realizen las transformaciones solo sobre el DataSet de entrenamiento. Despues se aplicaran , sobre el DataSet para testear.

```
In [47]: 1 from sklearn.preprocessing import RobustScaler  
2 scale_attrs = x_train[['src_bytes', 'dst_bytes']]  
3  
4 robust_scaler = RobustScaler()  
5 x_train_scaled = robust_scaler.fit_transform(scale_attrs)  
6  
7 x_train_scaled = pd.DataFrame(x_train_scaled, columns = ['src_bytes', 'dst_bytes'])
```

vamos a recuperar el DataSet limpio y separar las etiquetas del resto de los datos no necesariamente se quiere aplicar las mismas transformaciones en ambos conjuntos.

RobustScaler de scikit-learn para realizar una transformación robusta de escalado a las columnas 'src\_bytes' y 'dst\_bytes' de tu conjunto de datos `x_train`

```
In [50]: 1 x_train_scaled.head(20)
```

```
Out[50]:
```

	src_bytes	dst_bytes
0	1.324818	101.920000
1	-0.160584	0.000000
2	0.948905	1.211429
3	-0.160584	0.000000
4	-0.131387	0.000000
5	0.007299	0.264762
6	6.372263	0.691429
7	2.500000	0.626667
8	0.591241	2.841905
9	1.058394	0.000000
10	37.755474	0.000000
11	-0.160584	0.000000
12	-0.160584	0.000000
13	1.065693	8.158095
14	-0.160584	0.000000
15	0.007299	0.142857
16	3.638686	0.628571

se muestran las primeras 20 filas del DataFrame `x_train_scaled`, utilizando el método `head()` de pandas

## V. Conclusiones

la preparación del conjunto de datos es una fase crítica en el proceso de análisis y modelado de datos. El notebook proporciona una visión detallada de las técnicas más utilizadas para transformar el conjunto de datos, destacando la importancia de abordar problemas como la limpieza, la normalización de características. Al explorar diversas técnicas, desde el manejo de valores atípicos hasta la codificación de variables categóricas, se evidencia la versatilidad y la necesidad de adaptar las estrategias según las características específicas del conjunto de datos. La comprensión profunda de estas técnicas permite mejorar la calidad de los datos, lo que a su vez contribuye a la construcción de modelos más precisos y robustos