ejercicio2-nombre1-apellido1

June 27, 2022

1 Ejercicio 2

Este ejercicio pretende poner en práctica la habilidad de limpiar datos y visualizar plots para crear finalmente modelos en **sklearn**.

El estudiante tendrá que repasar los comandos realizados en clase y lidiar con posibles errores durante el desarrollo.

Para facilitar y agilizar el desarrollo, el estudiante tendrá que rellenar los huecos marcados como # **codigo-alumno**. No obstante, si además el estudiante necesita ejecutar código adicional, siempre podrá utilizar cualquier celda intermedia.

Las celdas con el título **""" No alterar """**, no deben ser modificadas por el estudiante. Sin embargo sí que se pueden ejecutar, pues representan controles intermedios para asegurar que no se cometen errores importantes que desvirtuen el desarrollo esperado del ejercicio.

Finalmente, la entrega será un fichero .ipynb cambiando nombre y apellido al fichero. No hace falta entregarlo en html/pdf ni comprimirlo.

2 Fase inicial: Preparativos del ejercicio

Estableceremos una semilla que nos permita generar números aleatorios bajo control.

Importante: Todos los comandos (incluido algoritmos) generen números aleatorios deberás ser inicializados con esta semilla.

[303]:
```
seed = 99
```

Además, cargaremos todos los comandos vistos en el curso. Si el estudiante considera utilizar alguna librería adicional, puede hacerlo en esta fase.

[304]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
sns.set_style('darkgrid')
np.set_printoptions(precision=3)
warnings.filterwarnings("ignore")
```

```
from sklearn.model_selection import train_test_split, KFold, ShuffleSplit,␣
 ↪LeaveOneOut, StratifiedKFold, cross_val_score, cross_val_predict,␣
 ↪GridSearchCV, RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler, StandardScaler, Normalizer,␣
 ↪Binarizer, RobustScaler, OneHotEncoder, LabelEncoder, PowerTransformer
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.feature_selection import SelectKBest, chi2, RFE
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report, f1_score
```

Trabajaremos con el dataframe de sklearn llamado *fetch_kddcup99* y lo almacenacenaremos en una variable. Además, no trabajaremos con todas las variables, sino con las features seleccionadas en *key_colums* y la variable a predecir *target*.

Nota: Si tuvieramos problemas en esta celda, lo más probable es que se deba a que tengamos una versión inferior a 0.24 de sklearn

[305]:
```
from sklearn.datasets import fetch_kddcup99

data = fetch_kddcup99(as_frame=True)
df = data.frame

key_columns = ['duration', 'protocol_type', 'service', 'flag', 'logged_in',␣
 ↪'count', 'srv_count', 'serror_rate', 'dst_host_srv_count',␣
 ↪'dst_host_srv_serror_rate']
target = 'labels'

df = df[key_columns + [target]]
df.head()
```

[305]:

	duration	protocol_type	service	flag	logged_in	count	srv_count	\
0	0	b'tcp'	b'http'	b'SF'	1	8	8	
1	0	b'tcp'	b'http'	b'SF'	1	8	8	
2	0	b'tcp'	b'http'	b'SF'	1	8	8	
3	0	b'tcp'	b'http'	b'SF'	1	6	6	
4	0	b'tcp'	b'http'	b'SF'	1	6	6	

	serror_rate	dst_host_srv_count	dst_host_srv_serror_rate	labels

```
0         0.0                9                    0.0  b'normal.'
1         0.0               19                    0.0  b'normal.'
2         0.0               29                    0.0  b'normal.'
3         0.0               39                    0.0  b'normal.'
4         0.0               49                    0.0  b'normal.'
```

3 Fase exploración, limpieza y transformación

En la siguiente celda, comprobad que no haya nulos

[306]:
```
# codigo-alumno
```

En la siguiente celda, eliminad los registros duplicados

[308]:
```
# codigo-alumno
```

[310]:
```
""" No alterar """

try:
    assert df.shape == (54165, 11)
except:
    print('Algo falla')
```

En la siguiente celda, mostrad un barplot para la variable objetivo (labels)

Nota: Un barplot para variables categóricas, no histograma

[311]:
```
# codigo-alumno
```

A continuación, solo trabajaremos con las labels de mayor frecuencia. Por tanto, en la siguiente celda, filtrad el dataframe para quedarnos con las más frecuentes.

[313]:
```
# codigo-alumno
```

[315]:
```
""" No alterar """

try:
    assert df.shape == (50177, 11)
except:
    print('Algo falla')
```

En la siguiente celda, volved a mostra un barplot para la variable objetivo (labels) del dataframe ya filtrado

[316]:
```
# codigo-alumno
```

En la siguiente celda, exploraremos transformaciones sobre las variables numéricas. Para ello, se pide mostrar 3 histogramas por cada feature numérica: * Uno con el valor de la variable * Uno con

el valor de la variable transformada por Box-Cox (si es viable) * Uno con el valor de la variable transformada por Yeo-Johnson

[318]:
```
# codigo-alumno
```

Por la forma de los histogramas, podría ser un buen estudio convertir las variables numéricas a variables dummy, y es lo que hareis en este apartado. En concreto, en la siguiente celda, realizad una binarización de estas features tomando el criterio que considereis más apropiado.

Nota: No siempre la media o la mediana es la mejor de los umbrales.

[320]:
```
# codigo-alumno
```

[322]:
```
""" No alterar """

try:
    assert df.shape == (50177, 11)
    assert 0 < df['duration'].sum() < 50177
    assert 0 < df['count'].sum() < 50177
    assert 0 < df['srv_count'].sum() < 50177
    assert 0 < df['serror_rate'].sum() < 50177
    assert 0 < df['dst_host_srv_count'].sum() < 50177
    assert 0 < df['dst_host_srv_serror_rate'].sum() < 50177
    assert 0 < df['logged_in'].sum() < 50177
except:
    print('Algo falla')
```

Para finalizar con estas features, en la siguiente celda, se pide mostrar un baplot (no histograma) por cada una de estas variables binarias estratificado por la variable objetivo (labels)

[ ]:
```
# codigo-alumno
```

En la siguiente celda, exploraremos las variables categóricas. Para ello, se pide mostrar un barplot por cada feature categórica:

[324]:
```
# codigo-alumno
```

Para lo que resta de ejercicio, localizaremos las 2 categorías más frecuentes de las features *flag* y *service*. En la siguiente celda, se pide transformar estas features para que ponga *resto* a todos aquellos registros de estas features que no están entre las frecuentes, dejando así 3 catgorías en total.

[326]:
```
# codigo-alumno
```

[328]:
```
""" No alterar """

try:
    assert df.shape == (50177, 11)
    assert len(df[df['service'] == 'resto']) == 20395
```

```
    assert len(df[df['flag'] == 'resto']) == 5778
except:
    print('Algo falla')
```

Para finalizar con estas features, en la siguiente celda, se pide mostrar un baplot (no histograma) por cada una de estas features categóricas estratificado por la variable objetivo (labels)

[329]:
```
# codigo-alumno
```

Antes de empezar con la modelización, eliminad los duplicados en la siguiente celda:

[331]:
```
# codigo-alumno
```

[333]:
```
""" No alterar """

try:
    assert 100 < df.shape[0] < 1000
    assert df.shape[1] == 11
except:
    print('Algo falla')
```

4 Fase de modelos

4.1 Model 1

Realizad un ajuste de machine learning con las siguientes características: * probad 5 algoritmos en bucle y mostrar un boxplot con los resultados * usad la técnica de validación cruzada KFolds (5 folds) * entrenad solo con variables numericas

[334]:
```
# codigo-alumno
```

4.2 Model 2

Realizad un ajuste de machine learning con las siguientes características: * probad 5 algoritmos en bucle * usad la técnica de validación KFolds (5 folds) * usad un pipeline que encadene * One-hot-encoder con las variables tipo string * algortimo * entrenad con todas las variables * mostrad la matriz de confusión en cada caso

[335]:
```
# codigo-alumno
```

4.3 Model 3

Realizad un ajuste de machine learning en bucle con las siguientes características: * probad 3 PCA dentro del pipeline y en bucle para n_components=3,4,5 * probad 5 algoritmos en bucle * usad la técnica de validación StratifiedKFolds (5 folds) * usad un pipeline que encadene * One-hot-encoder con las variables tipo string, especificando eliminar la primera columna si es binaria * PCA * algortimo * mostrad la matriz de confusión en cada caso

[336]:
```
# codigo-alumno
```

4.4 Model 4

Con el estudio previo que hemos realizado de validación cruzada, ya sabremos qué algoritmos son más robustos. En esta parte, vamos a separar primero el dataset en *train* y *test*. En concreto, separaremos el 20% del dataset en el *test-set* con el que validaremos la calidad real del algortimo. Tras ello, solo con el *train test*, se pide realizar un ajuste de machine learning con las siguientes características: * usad el mejor algoritmo a vuestro juicio (pero se pide justificarlo en un comentario del código) * usad un pipeline que encadene * One-hot-encoder con las variables tipo string, especificando eliminar la primera columna si es binaria * algortimo * Realizad la predicción de *train* y *test* para poder mostrar ambas matrices de confusión (justificar si existe overfitting).

[337]:
```
# codigo-alumno
```

4.5 Model 5

Con el estudio previo que hemos realizado de validación cruzada, ya sabremos qué algoritmos son más robustos. En esta parte, vamos a separar primero el dataset en train y test. En concreto, separaremos el 20% del dataset en el test-set con el que validaremos la calidad real del algortimo. Tras ello, solo con el train test, se pide realizar un ajuste de machine learning con las siguientes características: * usad el mismo algoritmo que en el apartado anterior * usad la técnica de validación cruzada KFolds (5 folds) * usad un pipeline que encadene: * One-hot-encoder si son tipo variables tipo string, especificando eliminar la primera columna si es binaria * StandardScaler * algortimo * realizar un tuneado del modelo con grid-search

Con el mejor de los modelos tuneados, realizad la predicción de *train* y *test* para poder mostrar ambas matrices de confusión (justificar si existe overfitting).

[338]:
```
# codigo-alumno
```

[ ]:
```

```