ejercicio1-nombre1-apellido1

February 8, 2021

[ ]:
```
""" Cualquier librería adicional que necesiteis durante el ejercicio,␣
 ↪importadlo en esta sección """

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=2)
# warnings.filterwarnings("ignore")

from sklearn.preprocessing import MinMaxScaler, StandardScaler, Normalizer,␣
 ↪Binarizer, RobustScaler
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, PowerTransformer
from sklearn.impute import SimpleImputer, KNNImputer

from sklearn.feature_selection import SelectKBest, chi2, RFE
from sklearn.model_selection import train_test_split
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

from sklearn.model_selection import KFold, ShuffleSplit, LeaveOneOut,␣
 ↪StratifiedKFold
```

```
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

1 Ejercicio 1

Este ejercicio pretende poner en práctica la habilidad para crear modelos en **sklearn** mediante el uso de transformadores *ad hoc*.

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 adicional.

El estudiante tendrá siempre que introducir una semilla (seed) que generará acorde a su fecha de nacimiento (sin ser intrusivos en edad).

Finalmente, la entrega será un fichero .ipynb cambiando nombre y apellido al fichero.

[ ]:
```
""" El estudiante tendrá que utilizar la semilla proporcionada para todos los␣
↪procesos aleatorios """

seed = #dia-nacimiento-estudiante + 13 * mes-nacimiento-estudiante
```

1.0.1 Data cleansing

[ ]:
```
""" Leed el fichero con pandas y almacenarlo en una variable llamada data """

data = pd.read_csv('./data/titanic-2.csv')
```

[ ]:
```
""" Cread una variable adicional, llamada hasCabin,
    que tome valor 0 si la columna Cabin es nula, y 1 si no lo es """

data['hasCabin'] = # codigo-alumno
```

[ ]:
```
"""Eliminad las columnas PassengerId, Cabin, Ticket y Name de data (comando␣
↪drop) """

data = data.drop(# codigo-alumno
```

[ ]:
```
""" Modificad las variables Title, Parch y SibSp, donde Title tome los valores␣
↪Mr, Mrs, Miss y Otros.
    Y Parch y SibSp toman los valores 0, 1 o 2 (donde 2 incluye 2 o más) """

data['SibSp'] = # codigo-alumno
data['Parch'] = # codigo-alumno
data['Title'] = # codigo-alumno
```

[ ]:
```
""" Eliminad los dos registros cuyo valor Embarked es nulo
    (se recomienda comprobar que se hayan eliminado correctamente) """

# codigo-alumno
data.isnull().sum()
```

[ ]:
```
""" Eliminad registros duplicados en caso de que los haya """
# codigo-alumno
data.duplicated().sum()
```

1.0.2 Feature engineering

[ ]:
```
"""
Realizar un ColumnTransformer que lleve:
- Un KNN Imputer para 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare'
- Un PowerTransformer yeo-johnson para 'Fare'
- Un OneHotEncoder para las variables 'Sex', 'Parch', 'Embarked' y 'Title'
"""

col_transformer = # codigo-alumno

ctransformed = col_transformer.fit_transform(data)
ctransformed
```

1.0.3 Model Selection

[ ]:
```
""" Realizaremos un análisis de los siguientes modelos con las siguentes␣
 ↪features """

models = []
models.append(('LR', LogisticRegression(random_state=seed)))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DTC', DecisionTreeClassifier(random_state=seed)))
models.append(('NB', GaussianNB()))
models.append(('RFC', RandomForestClassifier(random_state=seed)))
models.append(('SVM', SVC()))

X = data[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked',␣
 ↪'Title', 'hasCabin']]
y = data['Survived']
```

[ ]:
```
""" Para plotear los resultados utilizaremos la siguiente función """

def boxplots_algorithms(results, names):

    plt.figure(figsize=(8,8))
```

```
    plt.boxplot(results)
    plt.xticks(range(1,len(names)+1), names)
    plt.show()
```

[ ]:
```
""" Realizad un bucle que calcule, para cada modelo,
    - Un pipeline que realice:
        1. El ColumnTransformer diseñado anteriormente
        2. Un RobustScaler a continuación y, finalmente,
        3. Evalue cada modelo
    - Una validación cruzada:
        1. Tipo KFolds, con 10 splits
        2. Aleatorio con semilla y,
        3. scoring='accuracy'
     Usaremos la función boxplots_algorithms para plotear los resultados """

results = []
names = []

for name, model in models:

    scaler = # codigo-alumno
    pipeline = # codigo-alumno

    cv_technique = # codigo-alumno
    cv_results = cross_val_score(# codigo-alumno

    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

    results.append(cv_results)
    names.append(name)

boxplots_algorithms(results, names)
```

[ ]:
```
""" Realizad el mismo estudio de selección de variables, pero con␣
 ↪StratifiedKFolds """

results = []
names = []

for name, model in models:

    scaler = # codigo-alumno
    pipeline = # codigo-alumno

    cv_technique = # codigo-alumno
    cv_results = cross_val_score(# codigo-alumno
```

```
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

    results.append(cv_results)
    names.append(name)

boxplots_algorithms(results, names)
```

1.0.4 Model Tuning

[ ]:
```
""" Realizad un Tuneado del RandomForestClassifier para estimar cuál es la␣
 ↪mejor configuración paramétrica
    Y comprobarlo con GridSearchCV """

param_grid = {
    'model__n_estimators': [10, 20],
    'model__max_features': ['auto', 'sqrt', 'log2'],
    'model__max_depth' : [4,5,6,7,8],
    'model__criterion' :['gini', 'entropy']
}
model = RandomForestClassifier(# codigo-alumno

pipeline = Pipeline([('transformacion_columna', col_transformer),
                     ('robust_scaler', scaler),
                     ('model', model)])

cv_technique = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
grid_model = # codigo-alumno
grid_model.fit(X, y)

print(grid_model.best_score_)
print(grid_model.best_estimator_)
```

[ ]:
```

```