intro

June 9, 2021

1 Análisis exploratorio de datos

El conjunto de datos con el que trabajaremos es una recopilación de situaciones de ajedrez en las cuales el rey y la torre blanca intentan dar mate al rey negro en un determinado número de jugadas. METER LINK

El objetivo es determinar, a partir de las posiciones del rey y la torre negra, el número de movimientos necesario para dar mate al rey negro. Es posible que la configuración de piezas no dé lugar a un mate y acabe en tablas. Además, todos los datos están generados asumiendo un estilo de juego óptimo usando el estimador de teoría de juegos *Minimax*

Nuestro conjunto de datos está formado por 7 columnas, de las cuales las 6 primeras serán las variables predictoras y la última la variable a predecir:

1. Columna del rey blanco (wkc)
2. Fila del rey blanco (wkr)
3. Columna de la torre blanca (wrc)
4. Fila de la torre blanca (wrr)
5. Columna del rey negro (bkc)
6. Fila del rey negor (bkr)
7. Número de movimientos óptimos para que ganen las blancas. Varían del 0 al 16 más la posibilidad de empate.

Nuestro trabajo será construir modelos que sean capaces de predecir el número de movimientos óptimos para ganar a partir de las posiciones de las tres piezas.

[11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from IPython.display import set_matplotlib_formats
set_matplotlib_formats('png', 'pdf')
```

Importamos los datos usando pandas:

[12]:
```
data = pd.read_csv("krkopt.data", header=None)
data.columns = ["wkc", "wkr", "wrc", "wrr", "bkc", "bkr", "opt rank" ]
```

Veamos la estructura del conjunto:

[13]:
```
data
```

[13]:
```
       wkc  wkr wrc  wrr bkc  bkr opt rank
0        a    1   b    3   c    2     draw
1        a    1   c    1   c    2     draw
2        a    1   c    1   d    1     draw
3        a    1   c    1   d    2     draw
4        a    1   c    2   c    1     draw
…     ..  …   ..  …   ..  …       …
28051    b    1   g    7   e    5  sixteen
28052    b    1   g    7   e    6  sixteen
28053    b    1   g    7   e    7  sixteen
28054    b    1   g    7   f    5  sixteen
28055    b    1   g    7   g    5  sixteen

[28056 rows x 7 columns]
```

Existe un total de 28056 casos para 7 columnas.

Comprobamos si existen valores perdidos:

[14]:
```
np.sum(data.isnull())
```

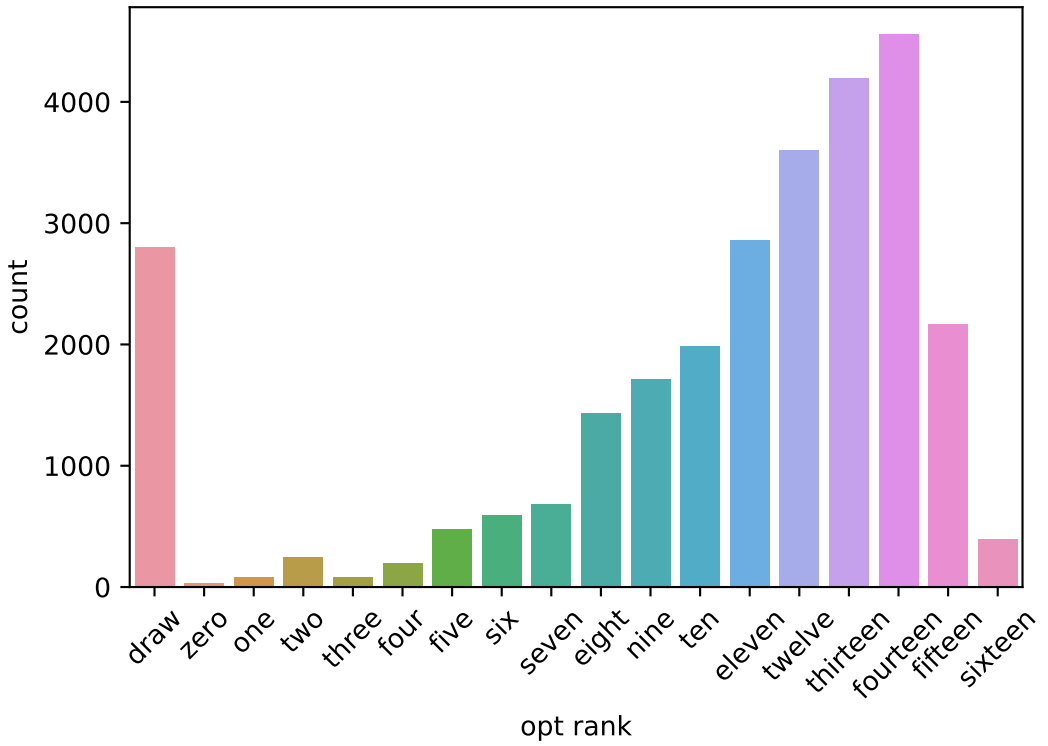
[14]:
```
wkc         0
wkr         0
wrc         0
wrr         0
bkc         0
bkr         0
opt rank    0
dtype: int64
```

No existen valores omitidos, por lo que podemos usar todos los casos sin ningún problema.

Estamos interesados en predecir los valores de *otp rank*, nos preguntamos cuál es la distribución de los casos para esta columna:

[15]:
```
plt.xticks(rotation=45)
sns.countplot(x='opt rank',
              data=data)
```

[15]:
```
<AxesSubplot:xlabel='opt rank', ylabel='count'>
```



[16]:
```
from collections import Counter
counter = Counter(data['opt rank'])
for k,v in counter.items():
        per = v / len(data['opt rank']) * 100
        print('Class=%s, n=%d (%.3f%%)' % (k, v, per))
```

```
Class=draw, n=2796 (9.966%)
Class=zero, n=27 (0.096%)
Class=one, n=78 (0.278%)
Class=two, n=246 (0.877%)
Class=three, n=81 (0.289%)
Class=four, n=198 (0.706%)
Class=five, n=471 (1.679%)
Class=six, n=592 (2.110%)
Class=seven, n=683 (2.434%)
Class=eight, n=1433 (5.108%)
Class=nine, n=1712 (6.102%)
Class=ten, n=1985 (7.075%)
Class=eleven, n=2854 (10.173%)
Class=twelve, n=3597 (12.821%)
Class=thirteen, n=4194 (14.949%)
Class=fourteen, n=4553 (16.228%)
Class=fifteen, n=2166 (7.720%)
```

```
Class=sixteen, n=390 (1.390%)
```

Tal y como vemos, se trata de un problema de clasificación desbalanceada. Existe una gran tendencia en la distribución a necesitar un número de entre diez y catorce pasos para acabar la partida. Destacamos, de igual manera, la importante cantidad de veces que acaba en tablas. Hay muy poca representanción de partidas que puedan acabar en el rango de uno a siete movimientos.

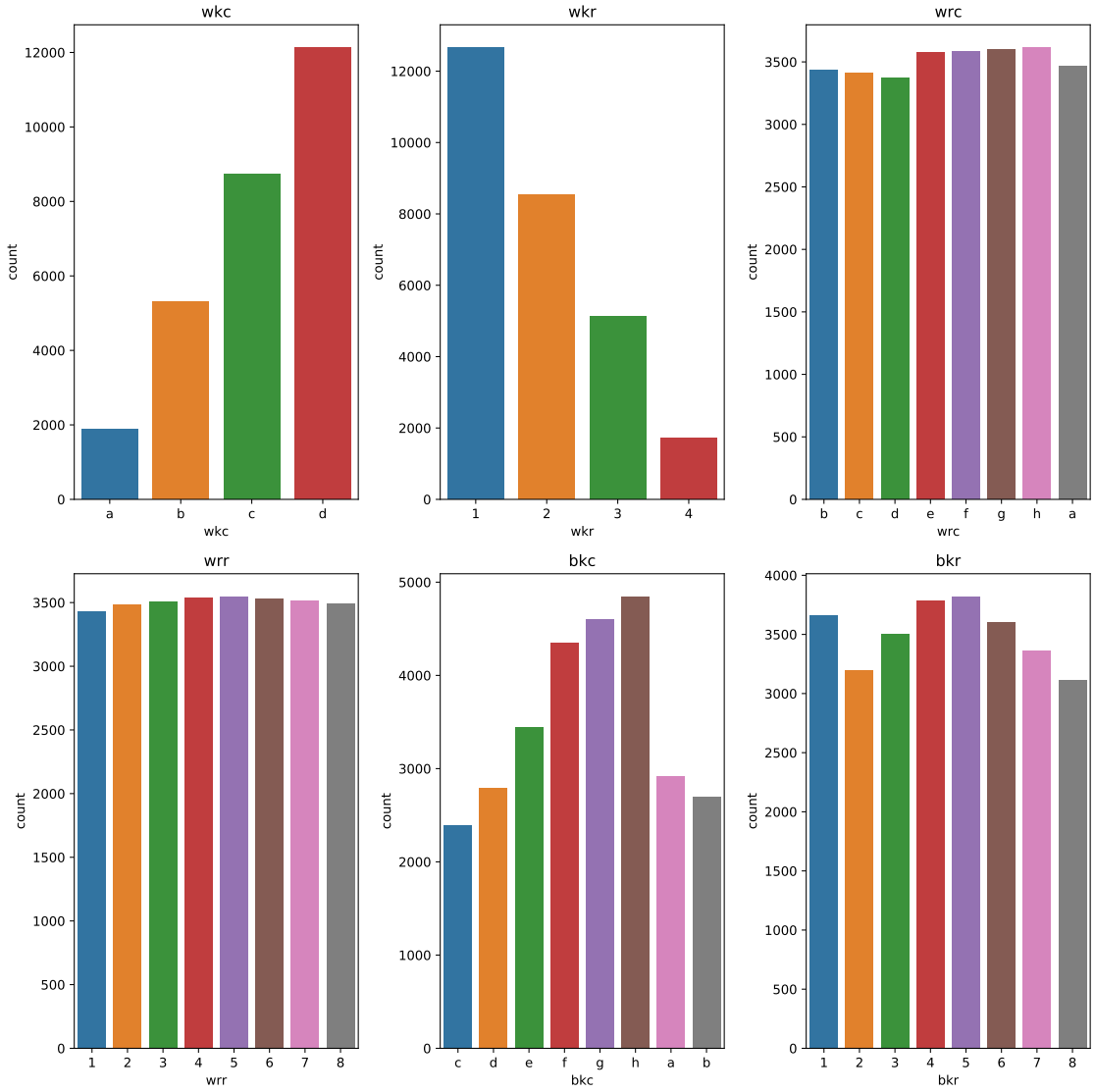
Este tipo de conjuntos de datos son problemáticos ya que el gran desbalanceo existente entre las clases impide que los modelos construidos consigan buenas métricas en la clasificación. Por ello, usaremo, además de los datos *raw*, la herramiento SMOTE, que realiza un sobremuestreo de los datos. Su funcionamiento es parecido a la interpolación, pero mucho más complejo.

Continuamos con la distribución de las demás clases en cada variable predictora:

[18]:
```
f, ax = plt.subplots(2,3, figsize=(12,12) )
sns.countplot(ax= ax[0,0], x='wkc',
              data=data)
sns.countplot(ax= ax[0,1], x='wkr',
              data=data)
sns.countplot(ax= ax[0,2], x='wrc',
              data=data)
sns.countplot(ax= ax[1,0], x='wrr',
              data=data)
sns.countplot(ax= ax[1,1], x='bkc',
              data=data)
sns.countplot(ax= ax[1,2], x='bkr',
              data=data)


ax[0,0].set_title("wkc")
ax[0,1].set_title("wkr")
ax[0,2].set_title("wrc")
ax[1,0].set_title("wrr")
ax[1,1].set_title("bkc")
ax[1,2].set_title("bkr")

plt.tight_layout()
plt.savefig('foo.pdf', transparent=True)
```



Existe una mayor homogeneidad en las variables predictoras correspondientes a la posición de las torres.

Es curioso que el rey blanco suele estar en la columna d en la primera fila, mientras que el rey negro presenta una distribución mucho más variada.

Todavía no hemos realizado ningúna conversión de los tipos ya que preferimos aplazarlo hasta la aplicación de determinados algoritmos, que nos exijan la codificación de las columnas en un deterimnado tipo.

1.1 Importancia de variables

Podemos utilizar algoritmos basados en árboles de decisión como RandomForest para estimar qué variables predictoras importan más a la hora de determinar la variable a predecir.

Para ello utilizaremos el paquete Scikit-learn. En primer lugar codificaremos numéricamente las variables categóricas.

[19]:
```
data_aux = data.copy()
data_aux[['wkc', 'wrc', 'bkc']] = data_aux[['wkc', 'wrc', 'bkc']].
 ↪astype('category')

data_aux['wkc'] = data_aux['wkc'].cat.codes
data_aux['wrc'] = data_aux['wrc'].cat.codes
data_aux['bkc'] = data_aux['bkc'].cat.codes
```

Entrenamos un modelo de RandomForest

[20]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data_aux.drop("opt rank",␣
 ↪axis=1), data_aux['opt rank']
                                                    ,test_size=0.2)
feature_names = [f'feature {i}' for i in range(data_aux.drop('opt rank',␣
 ↪axis=1).shape[1])]

forest = RandomForestClassifier(random_state=0)
forest.fit(X_train, y_train)
```

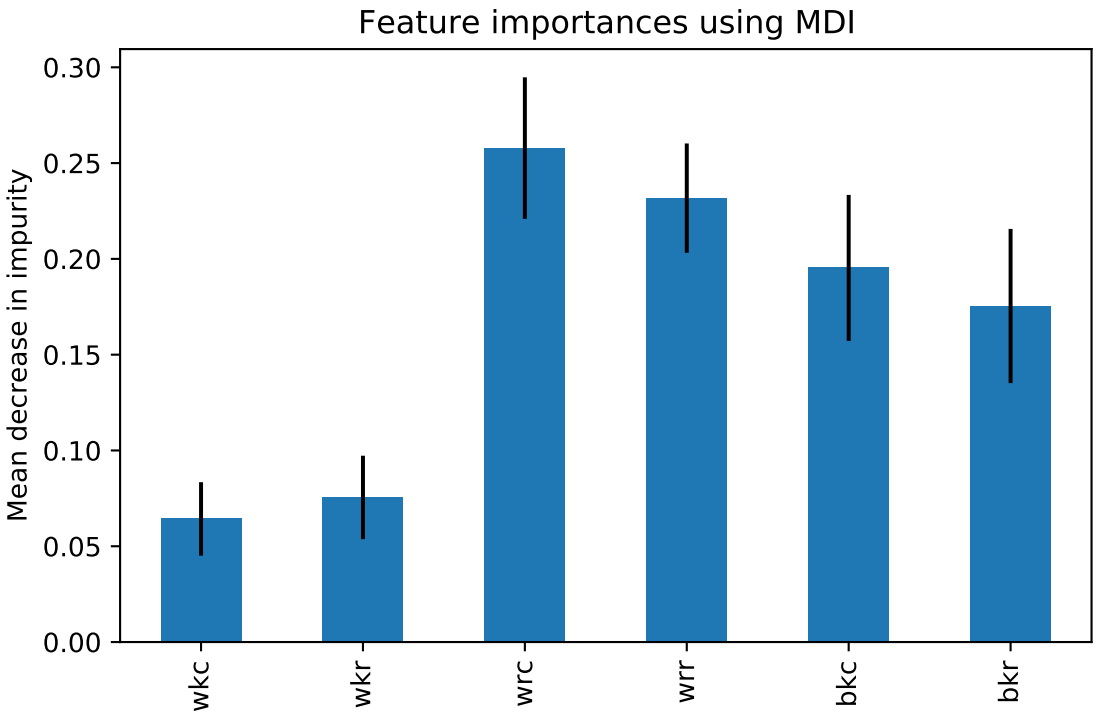
[20]:
```
RandomForestClassifier(random_state=0)
```

[22]:
```
importances = forest.feature_importances_
std = np.std([
    tree.feature_importances_ for tree in forest.estimators_], axis=0)

forest_importances = pd.Series(importances, index=feature_names)

fig, ax = plt.subplots()
forest_importances.plot.bar(yerr=std, ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
ax.set_xticks([0,1,2,3,4,5]) # values
labels = list(data_aux.columns[:-1])
ax.set_xticklabels(labels) # labels
positions = (1, 2, 3,4, 5)


fig.tight_layout()
```



Las variables más importantes a la hora de determinar el número de pasos son las coordenadas de la torre junto a las coordenadas del rey negro. Parece que la posición del rey blanco no es tan importante como el papel de la torre a la hora de ir *acorralando* al rey negro.

Podemos dar por finalizado el análisis exploratorio de datos, ya que no necesitamos hacer ningún test estadístico o análisis continuo de los datos debido a la naturaleza categórica de los mismos. Además, la distribución de clases por columna no sigue ninguna distribución normal, atendiendo a las figuras obtenidas.

2 Creación de modelos con redes neuronales con TensorFlow

De acuerdo a lo aprendido en la asignatura, utilizaremos TensorFlow en su versión 2.5 para crear distintos modelos de redes neuronales para abordar este problema de clasificación. Los datos que usaremos son los originales y los balanceados usando SMOTE.

Para cada conjunto crearemos distintos modelos de perceptrón multicapa en los cuales variaremos la profundidad del mismo (número de capas) así como la cantidad de neuronas por capa.

Haremos uso de técnicas como dropout y callbacks. La técnica del dropout consiste en desactivar ciertas neuronas por capa de forma aleatoria en cada época con el fin de evitar que la red neuronal sobreaprenda. Con esta idea, se debe poder conseguir redes neuronales de mayor tamaño que permitan aprender más características del conjunto de datos aún sin sobreaprender.

Los callbacks son utilidades que incluye Tensorflow que facilitan el entrenamiento de los modelos. Podemos guardar en disco el modelo que mejor resultado da en función de una métrica de control. En nuestro caso, estamos interesados en la función *EarlyStopping* que permite acabar el

entrenamiento de forma automática si, tras un número determinado a elegir de épocas, la métrica que elijamos no ha mejorado. Esta ventaja es fundamental ya que nos ahorra tener que incluir el número de épocas dentro de los parámetros tuneables durante el entrenamiento.

Para realizar el entrenamiento, estamos usando Linux junto a una tarjeta gráfica Nvidia 1070 max-q que nos ayuda a acelerar el trabajo de forma drástica. El entorno de Linux permite la configuración de las librerías CUDA de forma sencilla.

De entre los modelos vistos en clase, nos hemos decantado por el perceptrón multicapa ya que es el que más fácilmente se adapta a este problema. Por un lado, no podemos usar las redes convolucionales, joya de la corona del deep learning actualmente, ya que se usan en reconomiento de imágenes. No sabemos cómo utilizar las capas convolucionales en este tipo de datos. Muchas funcionalidades de TensorFlow no son usadas, como *DataAugmentation*, que permite variar el conjunto de entrenamiento para que la red neuronal tenga más variedad durante el mismo. Por otro lado, los mapas autogenerativos así como las redes neuronales con funciones de base radial no tienen suficiente documentación en internet para este tipo de problemas. Las redes neuronales recurrentes no se suelen usar en estas situaciones.

2.1 Datos *raw* con *smote* y *dropout*

Importamos los datos y separamos en variables predictoras: X y variable a predecir: y.

[4]:
```
import pandas as pd
import numpy as np
data = pd.read_csv("krkopt.data", header=None)
data.columns = ["wkc", "wkr", "wrc", "wrr", "bkc", "bkr", "opt rank" ]
X = data.iloc[:, 0:6]
y = data['opt rank']
X
```

[4]:

	wkc	wkr	wrc	wrr	bkc	bkr
0	a	1	b	3	c	2
1	a	1	c	1	c	2
2	a	1	c	1	d	1
3	a	1	c	1	d	2
4	a	1	c	2	c	1
…	..	…	..	…	..	…
28051	b	1	g	7	e	5
28052	b	1	g	7	e	6
28053	b	1	g	7	e	7
28054	b	1	g	7	f	5
28055	b	1	g	7	g	5

[28056 rows x 6 columns]

Codificamos los valores “a”, “b”, “c” de las variables categóricas a enteros para que puedan ser utilizadas por las redes neuronales

[5]:
```
X["wkc"]=X["wkc"].astype('category')
X["wrc"]=X["wrc"].astype('category')
X["bkc"]=X["bkc"].astype('category')
X["wkc"]=X["wkc"].cat.codes
X["wrc"]=X["wrc"].cat.codes
X["bkc"]=X["bkc"].cat.codes
y = y.astype('category')
y = y.cat.codes
X
```

[5]:

	wkc	wkr	wrc	wrr	bkc	bkr
0	0	1	1	3	2	2
1	0	1	2	1	2	2
2	0	1	2	1	3	1
3	0	1	2	1	3	2
4	0	1	2	2	2	1
…	…	…	…	…	…	…
28051	1	1	6	7	4	5
28052	1	1	6	7	4	6
28053	1	1	6	7	4	7
28054	1	1	6	7	5	5
28055	1	1	6	7	6	5

[28056 rows x 6 columns]

[6]:
```
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow import keras
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
```

Creamos los conjuntos de entrenamiento y test con una proporción 80-20 usando herramientas de *sklearn* y utilizamos oversampling con *smote*

[7]:
```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y, test_size=0.2,
                                                    random_state = 1)


from imblearn.over_sampling import SMOTE
oversample = SMOTE()



X_smote, y_smote = oversample.fit_resample(X, y)

X_train_smote, X_test_smote, y_train_smote, y_test_smote =␣
↪train_test_split(X_smote,
                                                    y_smote, test_size=0.2,
```

```
                                                    random_state = 1)



y_train = to_categorical(y_train)
y_test  = to_categorical(y_test)
y_train_smote = to_categorical(y_train_smote)
y_test_smote = to_categorical(y_test_smote)
```

Como vemos, *Smote* permite utilizar datos categóricos aunque no sea lo más recomendable

[9]:
```
X_smote
```

[9]:

	wkc	wkr	wrc	wrr	bkc	bkr
0	0	1	1	3	2	2
1	0	1	2	1	2	2
2	0	1	2	1	3	1
3	0	1	2	1	3	2
4	0	1	2	2	2	1
…	…	…	…	…	…	…
81949	2	3	0	1	2	1
81950	2	2	0	7	0	2
81951	2	1	0	8	0	1
81952	2	1	0	8	0	1
81953	2	3	6	1	2	1

[81954 rows x 6 columns]

Estandarizamos los datos de entrada usando *zscore*.

[10]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

scaler = StandardScaler().fit(X_train_smote)
X_train_smote = scaler.transform(X_train_smote)
X_test_smote = scaler.transform(X_test_smote)
```

2.1.1 Definición de funciones

Definimos tres funciones que nos serán útiles para automatizar el proceso: - *make_my_model_multi* permite crear un perceptrón multicapa proporcionándole la estructura de la forma $[n_1, n_2, ..., n_i, ..., n_N]$ donde n_i es el número de neuronas de la capa oculta i. Además de otros parámetros como la forma de entrada, salida y la función de activación que queramos usar. - *compile_fit_multiclass* entrena un modelo de entrada con el conjunto de entrenamiento usando un conjunto de validación 80-20 y produce predicciones con un conjunto test. Utilizamos la her-

ramienta *ModelCheckpoint* para guardar el mejor modelo durante el entrenamiento y *EarlyStopping* para parar el entrenamiento si el valor de *loss* en el conjunto de validación no mejora en 10 épocas. Así evitamos el sobreaprendizaje.

- *compute_metrics_multiclass* calcula las métricas precision, recall, F1 y Kappa a partir de las predicciones y los valores exactos de test.
- *make_my_model_multi_dropout* es una modificación que permite crear un modelo introduciendo capas internas de dropout. La estructura de la red se introduce de la forma $[n_1, n_2, ..., n_i, ..., n_N]$ donde n_i es el número de neuronas de la capa i si escribimos un número entero o una capa dropout con un valor de desactivación n_i si introducimos un elemento de tipo carácter. Por ejemplo, [50, "0.2", 50, "0.2"] creará unas capas ocultas de la forma: capa con 50 neuronas, dropout 0.2, capa con 50 neuronas y dropout de 0.2.

[11]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


checkpoint_filepath = '/tmp/checkpoint'
from sklearn.metrics import confusion_matrix, precision_score, \
f1_score, cohen_kappa_score, recall_score

def make_my_model_multi( units_per_layer, input_s, output_s,␣
↪activation_='relu'):
    model = Sequential()
    depth = len(units_per_layer)
    model.add(Dense(units_per_layer[0], activation=activation_,␣
↪input_shape=(input_s,)))
    for i in range(1, depth):
        model.add(Dense(units_per_layer[i], activation=activation_))
    model.add(Dense(output_s, activation = 'softmax'))

    return model




def compile_fit_multiclass(modelo, X_train, X_test, y_train, batch, epochs,␣
↪verbose=0):
    modelo.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',␣
↪patience=10, verbose=True)

    model_checkpoint = tf.keras.callbacks.
↪ModelCheckpoint(filepath=checkpoint_filepath,
```

```
                                                    save_weights_only=True,
                                                    monitor='val_loss',
                                                    mode='min',
                                                    save_best_only=True,
                                                    verbose=False)

    modelo.fit(X_train, y_train, epochs=epochs, batch_size=batch,␣
 ↪verbose=verbose, validation_split=0.2, callbacks = [early_stopping,␣
 ↪model_checkpoint])
    model.load_weights(checkpoint_filepath)
    predictions = modelo.predict(X_test)
    return predictions

def compute_metrics_multiclass(y_test, y_pred):
    results=[]
    results.append(precision_score(y_test, np.round(y_pred), average="micro"))
    results.append(recall_score(y_test, np.round(y_pred), average="micro"))
    results.append(f1_score(y_test, np.round(y_pred), average="micro"))
    results.append(cohen_kappa_score(y_test, np.round(y_pred)))
    return results


from tensorflow.keras.layers import Dropout

def make_my_model_multi_dropout( units_per_layer, input_s, output_s,␣
 ↪activation_='relu'):
    model = Sequential()
    depth = len(units_per_layer)
    model.add(Dense(units_per_layer[0], activation=activation_,␣
 ↪input_shape=(input_s,)))
    for i in range(1, depth):
        if isinstance(units_per_layer[i], str):
            a = units_per_layer[i]
            dropout_r = float(a)
            model.add(Dropout(dropout_r))
        else:
            model.add(Dense(units_per_layer[i], activation=activation_))
    model.add(Dense(output_s, activation = 'softmax'))

    return model
```

Comprobamos que los datos tienen las dimensiones correctas

[22]:
```
X_train.shape, X_train_smote.shape, y_train.shape, y_train_smote.shape, X_test.
 ↪shape, y_test.shape
```

[22]: ((22444, 6), (65563, 6), (22444, 18), (65563, 18), (5612, 6), (5612, 18))

2.1.2 Pruebas

Datos *raw* Creamos un total de treinta experimentos en las que crearemos redes neuronales de la misma cantidad de neuronas por capas con distinto número de neuronas y distinto número de capas. El número de neuronas será: [50, 100, 150, 200, 250] y el tamaño variará de entre una y seis capas ocultas. Guardamos las predicciones junto a las métricas y la matriz de confusión. Escribimos en disco el objeto mediante *joblib* para su posterior análisis.

[15]:
```
results = []
seed = 1
from sklearn.metrics import confusion_matrix
```

[54]:
```
size_config = [50, 100, 150, 200, 250]
for size in size_config:
    layer_config = [[size], [size]*2, [size]*3, [size]*4, [size]*5, [size]*6]
    for layers in layer_config:
        np.random.seed(seed)
        tf.random.set_seed(seed)
        print(layers)
        model = make_my_model_multi(layers, 6, 18, activation_='relu' )
        preds = compile_fit_multiclass(model, X_train, X_test, y_train, 256,␣
↪300, verbose=0)
        metrics = compute_metrics_multiclass(np.argmax(preds, axis = 1), np.
↪argmax(y_test, axis = 1))
        confusion = confusion_matrix(np.argmax(preds, axis = 1), np.
↪argmax(y_test, axis = 1))
        aux = { "layer config" : layers,
                #"Model": model,
                "Predictions" : preds,
                "Metrics" : metrics,
                "Confusion" : confusion

        }
        print(metrics)
        results.append(aux)
```

```
[50]
[0.5563079116179616, 0.5563079116179616, 0.5563079116179616, 0.5014778486804247]
[50, 50]
Epoch 00278: early stopping
[0.6776550249465432, 0.6776550249465432, 0.6776550249465432, 0.6394872591777784]
[50, 50, 50]
Epoch 00189: early stopping
[0.7004632929436921, 0.7004632929436921, 0.7004632929436921, 0.66527050745813]
[50, 50, 50, 50]
Epoch 00135: early stopping
[0.7033143264433357, 0.7033143264433357, 0.7033143264433357, 0.6683596780441357]
```

```
[50, 50, 50, 50, 50]
Epoch 00083: early stopping
[0.7140057020669993, 0.7140057020669993, 0.7140057020669993, 0.6802859773047576]
[50, 50, 50, 50, 50, 50]
Epoch 00101: early stopping
[0.7325374198146828, 0.7325374198146828, 0.7325374198146829, 0.7014741703016625]
[100]
[0.5841054882394868, 0.5841054882394868, 0.5841054882394868, 0.532926380750063]
[100, 100]
Epoch 00226: early stopping
[0.7145402708481825, 0.7145402708481825, 0.7145402708481825, 0.6810448690167336]
[100, 100, 100]
Epoch 00145: early stopping
[0.7808267997148967, 0.7808267997148967, 0.7808267997148967, 0.7552060507593273]
[100, 100, 100, 100]
Epoch 00117: early stopping
[0.7770848182466144, 0.7770848182466144, 0.7770848182466144, 0.7511752665391203]
[100, 100, 100, 100, 100]
Epoch 00083: early stopping
[0.7854597291518175, 0.7854597291518175, 0.7854597291518175, 0.7603159232769513]
[100, 100, 100, 100, 100, 100]
Epoch 00089: early stopping
[0.7861724875267284, 0.7861724875267284, 0.7861724875267283, 0.7611696219845376]
[150]
[0.6051318602993585, 0.6051318602993585, 0.6051318602993585, 0.5571977645455048]
[150, 150]
Epoch 00196: early stopping
[0.7462580185317177, 0.7462580185317177, 0.7462580185317177, 0.7163812931399909]
[150, 150, 150]
Epoch 00117: early stopping
[0.7813613684960798, 0.7813613684960798, 0.7813613684960798, 0.7561423259186348]
[150, 150, 150, 150]
Epoch 00087: early stopping
[0.7957947255880257, 0.7957947255880257, 0.7957947255880257, 0.7718799039500605]
[150, 150, 150, 150, 150]
Epoch 00094: early stopping
[0.8209194583036351, 0.8209194583036351, 0.8209194583036351, 0.799991822447679]
[150, 150, 150, 150, 150, 150]
Epoch 00083: early stopping
[0.8184248039914469, 0.8184248039914469, 0.8184248039914469, 0.7973288336155344]
[200]
[0.6286528866714184, 0.6286528866714184, 0.6286528866714184, 0.5841443006982328]
[200, 200]
Epoch 00160: early stopping
[0.7530292230933714, 0.7530292230933714, 0.7530292230933714, 0.7243038312259522]
[200, 200, 200]
Epoch 00094: early stopping
[0.7794012829650748, 0.7794012829650748, 0.7794012829650748, 0.7533991727673046]
```

```
[200, 200, 200, 200]
Epoch 00072: early stopping
[0.8029223093371347, 0.8029223093371347, 0.8029223093371347, 0.7797459386027457]
[200, 200, 200, 200, 200]
Epoch 00064: early stopping
[0.8048823948681397, 0.8048823948681397, 0.8048823948681397, 0.7819079964441699]
[200, 200, 200, 200, 200, 200]
Epoch 00061: early stopping
[0.8191375623663578, 0.8191375623663578, 0.8191375623663577, 0.7979856944628764]
[250]
[0.6416607270135424, 0.6416607270135424, 0.6416607270135424, 0.5990285359827496]
[250, 250]
Epoch 00145: early stopping
[0.7528510334996437, 0.7528510334996437, 0.7528510334996438, 0.7238185938600421]
[250, 250, 250]
Epoch 00065: early stopping
[0.7694226657163221, 0.7694226657163221, 0.7694226657163221, 0.7424900002897106]
[250, 250, 250, 250]
Epoch 00072: early stopping
[0.8161083392729864, 0.8161083392729864, 0.8161083392729864, 0.7947878390730547]
[250, 250, 250, 250, 250]
Epoch 00060: early stopping
[0.8251960085531005, 0.8251960085531005, 0.8251960085531005, 0.8045311760051372]
[250, 250, 250, 250, 250, 250]
Epoch 00057: early stopping
[0.8218104062722738, 0.8218104062722738, 0.8218104062722738, 0.80115129573033]
```

[65]:
```
import joblib

joblib.dump(results, 'results_1_joblib')
```

[65]:
```
['results_1_joblib']
```

Datos con *smote* Repetimos el mismo esquema con los datos con *smote*.

[30]:
```
results_smote = []
seed = 1
```

[31]:
```
size_config = [50, 100, 150, 200, 250]
for size in size_config:
    layer_config = [[size], [size]*2, [size]*3, [size]*4, [size]*5, [size]*6]
    for layers in layer_config:
        np.random.seed(seed)
        tf.random.set_seed(seed)
        print(layers)
        model = make_my_model_multi(layers, 6, 18, activation_='relu' )
```

```
        preds = compile_fit_multiclass(model, X_train_smote, X_test,␣
↪y_train_smote, 256, 300, verbose=0)
        metrics = compute_metrics_multiclass(np.argmax(preds, axis = 1), np.
↪argmax(y_test, axis = 1))
        confusion = confusion_matrix(np.argmax(preds, axis = 1), np.
↪argmax(y_test, axis = 1))
        aux = { "layer config" : layers,
                #"Model": model,
                "Predictions" : preds,
                "Metrics" : metrics,
                "Confusion" : confusion

        }
        print(metrics)
        results_smote.append(aux)
```

```
[50]
[0.2735210263720599, 0.2735210263720599, 0.2735210263720599,
0.21056283287164057]
[50, 50]
Epoch 00195: early stopping
[0.3016749821810406, 0.3016749821810406, 0.3016749821810406,
0.23858740268705947]
[50, 50, 50]
Epoch 00149: early stopping
[0.319672131147541, 0.319672131147541, 0.319672131147541, 0.2573189896196104]
[50, 50, 50, 50]
Epoch 00150: early stopping
[0.3257305773342837, 0.3257305773342837, 0.3257305773342837,
0.26329646382574623]
[50, 50, 50, 50, 50]
Epoch 00105: early stopping
[0.32216678545972915, 0.32216678545972915, 0.32216678545972915,
0.26004197522733685]
[50, 50, 50, 50, 50, 50]
Epoch 00170: early stopping
[0.3310762651461155, 0.3310762651461155, 0.3310762651461155, 0.2689915712533639]
[100]
[0.29971489665003564, 0.29971489665003564, 0.29971489665003564,
0.23722592685369615]
[100, 100]
Epoch 00174: early stopping
[0.32323592302209553, 0.32323592302209553, 0.32323592302209553,
0.26046775820865786]
[100, 100, 100]
Epoch 00133: early stopping
```

```
[0.3341054882394868, 0.3341054882394868, 0.3341054882394868,
0.27209407806805874]
[100, 100, 100, 100]
Epoch 00065: early stopping
[0.34052031361368496, 0.34052031361368496, 0.34052031361368496,
0.2793803604511679]
[100, 100, 100, 100, 100]
Epoch 00065: early stopping
[0.33856022808268, 0.33856022808268, 0.33856022808268, 0.2775702647152132]
[100, 100, 100, 100, 100, 100]
Epoch 00078: early stopping
[0.3447968638631504, 0.3447968638631504, 0.3447968638631504, 0.2820060064899931]
[150]
[0.30024946543121883, 0.30024946543121883, 0.30024946543121883,
0.23828899508645218]
[150, 150]
Epoch 00145: early stopping
[0.32555238774055595, 0.32555238774055595, 0.32555238774055595,
0.26301755677369276]
[150, 150, 150]
Epoch 00095: early stopping
[0.323592302209551, 0.323592302209551, 0.323592302209551, 0.26068327259513024]
[150, 150, 150, 150]
Epoch 00065: early stopping
[0.3469351389878831, 0.3469351389878831, 0.3469351389878831,
0.28626249332475373]
[150, 150, 150, 150, 150]
Epoch 00076: early stopping
[0.3551318602993585, 0.3551318602993585, 0.3551318602993585,
0.29535007953313264]
[150, 150, 150, 150, 150, 150]
Epoch 00089: early stopping
[0.3544191019244476, 0.3544191019244476, 0.3544191019244476, 0.2932453533302928]
[200]
[0.31272273699215963, 0.31272273699215963, 0.31272273699215963,
0.2508736929301466]
[200, 200]
Epoch 00155: early stopping
[0.3362437633642195, 0.3362437633642195, 0.3362437633642195, 0.2741266338548447]
[200, 200, 200]
Epoch 00072: early stopping
[0.33481824661439774, 0.33481824661439774, 0.33481824661439774,
0.2737471663026608]
[200, 200, 200, 200]
Epoch 00059: early stopping
[0.34337134711332856, 0.34337134711332856, 0.34337134711332856,
0.28165430613441933]
[200, 200, 200, 200, 200]
```

```
Epoch 00065: early stopping
[0.34746970776906627, 0.34746970776906627, 0.34746970776906627,
0.2864768326195607]
[200, 200, 200, 200, 200, 200]
Epoch 00078: early stopping
[0.35655737704918034, 0.35655737704918034, 0.3565573770491803,
0.2961375972429202]
[250]
[0.32020669992872414, 0.32020669992872414, 0.32020669992872414,
0.2589827289284935]
[250, 250]
Epoch 00116: early stopping
[0.3271560940841055, 0.3271560940841055, 0.3271560940841055,
0.26503000969397217]
[250, 250, 250]
Epoch 00084: early stopping
[0.3394511760513186, 0.3394511760513186, 0.3394511760513186, 0.2769489283726123]
[250, 250, 250, 250]
Epoch 00068: early stopping
[0.35477548111190305, 0.35477548111190305, 0.3547754811119031,
0.29475622872925544]
[250, 250, 250, 250, 250]
Epoch 00045: early stopping
[0.34978617248752675, 0.34978617248752675, 0.34978617248752675,
0.2885785810340571]
[250, 250, 250, 250, 250, 250]
Epoch 00057: early stopping
[0.3617248752672844, 0.3617248752672844, 0.36172487526728436,
0.3008943441645957]
```

[33]:
```
joblib.dump(results_smote, 'results_smote_joblib')
```

[33]:
```
['results_smote_joblib']
```

Datos raw con dropout A la hora de realizar experimentos con dropout, creamos el mismo esquema que en casos anteriores intercalando capas dropout de 3 posibles valores: 0.1, 0.2 y 0.3.

[33]:
```
results_dropout = []
seed = 1
```

Mostramos la configuración de los experimentos. En total, realizaremos noventa casos.

[61]:
```
size_config = [50, 100, 150, 200, 250]
dropout_rate = ["0.1", "0.2", "0.3"]

for size in size_config:
    for size_d in (dropout_rate):
```

```
        layer_config_dense = [[size], [size]*2, [size]*3, [size]*4, [size]*5,␣
↪[size]*6]
        layer_config_dropout = [[size_d], [size_d]*2, [size_d]*3, [size_d]*4,␣
↪[size_d]*5, [size_d]*6]
        for layers_dense, layers_dropout in zip(layer_config_dense,␣
↪layer_config_dropout):
            final_design = [None]*(len(layers_dense)+len(layers_dropout))
            final_design[::2] = layers_dense
            final_design[1::2] = layers_dropout
            print(final_design)
```

```
[50, '0.1']
[50, '0.1', 50, '0.1']
[50, '0.1', 50, '0.1', 50, '0.1']
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
[50, '0.2']
[50, '0.2', 50, '0.2']
[50, '0.2', 50, '0.2', 50, '0.2']
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
[50, '0.3']
[50, '0.3', 50, '0.3']
[50, '0.3', 50, '0.3', 50, '0.3']
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
[100, '0.1']
[100, '0.1', 100, '0.1']
[100, '0.1', 100, '0.1', 100, '0.1']
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
[100, '0.2']
[100, '0.2', 100, '0.2']
[100, '0.2', 100, '0.2', 100, '0.2']
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
[100, '0.3']
[100, '0.3', 100, '0.3']
[100, '0.3', 100, '0.3', 100, '0.3']
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
```

```
[150, '0.1']
[150, '0.1', 150, '0.1']
[150, '0.1', 150, '0.1', 150, '0.1']
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
[150, '0.2']
[150, '0.2', 150, '0.2']
[150, '0.2', 150, '0.2', 150, '0.2']
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
[150, '0.3']
[150, '0.3', 150, '0.3']
[150, '0.3', 150, '0.3', 150, '0.3']
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
[200, '0.1']
[200, '0.1', 200, '0.1']
[200, '0.1', 200, '0.1', 200, '0.1']
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
[200, '0.2']
[200, '0.2', 200, '0.2']
[200, '0.2', 200, '0.2', 200, '0.2']
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
[200, '0.3']
[200, '0.3', 200, '0.3']
[200, '0.3', 200, '0.3', 200, '0.3']
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
[250, '0.1']
[250, '0.1', 250, '0.1']
[250, '0.1', 250, '0.1', 250, '0.1']
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
[250, '0.2']
[250, '0.2', 250, '0.2']
[250, '0.2', 250, '0.2', 250, '0.2']
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
```

```
[250, '0.3']
[250, '0.3', 250, '0.3']
[250, '0.3', 250, '0.3', 250, '0.3']
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
```

[62]:
```
size_config = [50, 100, 150, 200, 250]
dropout_rate = ["0.1", "0.2", "0.3"]

for size in size_config:
    for size_d in (dropout_rate):
        layer_config_dense = [[size], [size]*2, [size]*3, [size]*4, [size]*5,␣
↪[size]*6]
        layer_config_dropout = [[size_d], [size_d]*2, [size_d]*3, [size_d]*4,␣
↪[size_d]*5, [size_d]*6]
        for layers_dense, layers_dropout in zip(layer_config_dense,␣
↪layer_config_dropout):
            final_design = [None]*(len(layers_dense)+len(layers_dropout))
            final_design[::2] = layers_dense
            final_design[1::2] = layers_dropout
            np.random.seed(seed)
            tf.random.set_seed(seed)
            print(final_design)
            model = make_my_model_multi_dropout(final_design, 6, 18,␣
↪activation_='relu' )
            preds = compile_fit_multiclass(model, X_train, X_test, y_train,␣
↪256, 300, verbose=0)
            metrics = compute_metrics_multiclass(np.argmax(preds, axis = 1), np.
↪argmax(y_test, axis = 1))
            confusion = confusion_matrix(np.argmax(preds, axis = 1), np.
↪argmax(y_test, axis = 1))
            aux = { "layer config" : final_design,
                   #"Model": model,
                   "Predictions" : preds,
                   "Metrics" : metrics,
                   "Confusion" : confusion

            }
            print(metrics)
            results_dropout.append(aux)
```

```
[50, '0.1']
[0.5306486101211689, 0.5306486101211689, 0.5306486101211689,
0.47299466324494865]
[50, '0.1', 50, '0.1']
```

```
[0.6582323592302209, 0.6582323592302209, 0.6582323592302209, 0.6168927965574147]
[50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00274: early stopping
[0.7109764789736279, 0.7109764789736279, 0.7109764789736278, 0.677058966107785]
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00187: early stopping
[0.7066999287241625, 0.7066999287241625, 0.7066999287241625, 0.6719743342215174]
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00152: early stopping
[0.6903064861012117, 0.6903064861012117, 0.6903064861012117, 0.6533659428663383]
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00176: early stopping
[0.6915538132573058, 0.6915538132573058, 0.6915538132573058, 0.6548578250956504]
[50, '0.2']
Epoch 00253: early stopping
[0.5172843905915895, 0.5172843905915895, 0.5172843905915895,
0.45668070032612573]
[50, '0.2', 50, '0.2']
[0.607448325017819, 0.607448325017819, 0.607448325017819, 0.5594133149504249]
[50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00155: early stopping
[0.6181397006414825, 0.6181397006414825, 0.6181397006414825, 0.5718332632332058]
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00221: early stopping
[0.642551674982181, 0.642551674982181, 0.642551674982181, 0.5987753202495634]
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00135: early stopping
[0.5915894511760513, 0.5915894511760513, 0.5915894511760513, 0.5418785789667531]
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00100: early stopping
[0.577512473271561, 0.577512473271561, 0.577512473271561, 0.5258937398977566]
[50, '0.3']
Epoch 00249: early stopping
[0.5110477548111191, 0.5110477548111191, 0.5110477548111191,
0.44936967418461227]
[50, '0.3', 50, '0.3']
Epoch 00214: early stopping
[0.5673556664290805, 0.5673556664290805, 0.5673556664290805, 0.5134787826514988]
[50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00235: early stopping
[0.5616535994297933, 0.5616535994297933, 0.5616535994297933, 0.5066628382810214]
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00124: early stopping
[0.5342124019957234, 0.5342124019957234, 0.5342124019957234, 0.4751719679727483]
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00127: early stopping
[0.5263720598717035, 0.5263720598717035, 0.5263720598717035, 0.4667612314609778]
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
```

```
Epoch 00108: early stopping
[0.5226300784034212, 0.5226300784034212, 0.5226300784034212,
0.46306875130107517]
[100, '0.1']
Epoch 00249: early stopping
[0.5461511047754811, 0.5461511047754811, 0.5461511047754811,
0.49062978889937336]
[100, '0.1', 100, '0.1']
[0.7521382751247327, 0.7521382751247327, 0.7521382751247327, 0.7227156116282598]
[100, '0.1', 100, '0.1', 100, '0.1']
[0.8257305773342837, 0.8257305773342837, 0.8257305773342837, 0.8054057526695507]
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
Epoch 00165: early stopping
[0.7993585174625801, 0.7993585174625801, 0.7993585174625801, 0.775965204492387]
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
Epoch 00221: early stopping
[0.8182466143977192, 0.8182466143977192, 0.8182466143977192, 0.7970575500307149]
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
Epoch 00199: early stopping
[0.8155737704918032, 0.8155737704918032, 0.8155737704918032, 0.7940122031851752]
[100, '0.2']
Epoch 00249: early stopping
[0.5386671418389166, 0.5386671418389166, 0.5386671418389166,
0.48189218191606287]
[100, '0.2', 100, '0.2']
[0.7248752672843906, 0.7248752672843906, 0.7248752672843906, 0.6919934078576849]
[100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00238: early stopping
[0.7674625801853172, 0.7674625801853172, 0.7674625801853173, 0.740127140644157]
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00191: early stopping
[0.7610477548111191, 0.7610477548111191, 0.7610477548111191, 0.7329919066201204]
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00158: early stopping
[0.7273699215965788, 0.7273699215965788, 0.7273699215965788, 0.6954064061016627]
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00089: early stopping
[0.6945830363506771, 0.6945830363506771, 0.6945830363506771, 0.6587686427382717]
[100, '0.3']
Epoch 00214: early stopping
[0.5340342124019958, 0.5340342124019958, 0.5340342124019958,
0.47652760627391544]
[100, '0.3', 100, '0.3']
Epoch 00252: early stopping
[0.6764076977904491, 0.6764076977904491, 0.6764076977904491, 0.637487713970418]
[100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00195: early stopping
[0.7033143264433357, 0.7033143264433357, 0.7033143264433357, 0.6684001878880355]
```

```
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00157: early stopping
[0.6847826086956522, 0.6847826086956522, 0.6847826086956522, 0.6471773664755471]
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00136: early stopping
[0.6781895937277262, 0.6781895937277262, 0.6781895937277262, 0.6394933393602311]
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00124: early stopping
[0.6370277975766215, 0.6370277975766215, 0.6370277975766215, 0.5935140862849402]
[150, '0.1']
Epoch 00272: early stopping
[0.5620099786172488, 0.5620099786172488, 0.5620099786172488, 0.508341442519633]
[150, '0.1', 150, '0.1']
Epoch 00266: early stopping
[0.7749465431218817, 0.7749465431218817, 0.7749465431218816, 0.7484450378452736]
[150, '0.1', 150, '0.1', 150, '0.1']
Epoch 00200: early stopping
[0.8355310049893087, 0.8355310049893087, 0.8355310049893087, 0.8162182870942731]
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
Epoch 00197: early stopping
[0.8597647897362795, 0.8597647897362795, 0.8597647897362795, 0.8433538331608326]
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
Epoch 00110: early stopping
[0.8241268709907341, 0.8241268709907341, 0.8241268709907341, 0.8035528526761304]
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
Epoch 00230: early stopping
[0.8624376336421953, 0.8624376336421953, 0.8624376336421952, 0.846293252489121]
[150, '0.2']
[0.5563079116179616, 0.5563079116179616, 0.5563079116179616, 0.5015463823385387]
[150, '0.2', 150, '0.2']
[0.7615823235923022, 0.7615823235923022, 0.7615823235923023, 0.7335657779063354]
[150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00204: early stopping
[0.8095153243050606, 0.8095153243050606, 0.8095153243050606, 0.7871495134285849]
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00264: early stopping
[0.8342836778332146, 0.8342836778332146, 0.8342836778332146, 0.8149513387287599]
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00148: early stopping
[0.8022095509622238, 0.8022095509622238, 0.8022095509622238, 0.778780229359274]
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00098: early stopping
[0.7696008553100498, 0.7696008553100498, 0.7696008553100498, 0.7424088375196287]
[150, '0.3']
Epoch 00249: early stopping
[0.5411617961511048, 0.5411617961511048, 0.5411617961511048, 0.4847149956069786]
[150, '0.3', 150, '0.3']
[0.7403777619387027, 0.7403777619387027, 0.7403777619387029, 0.7093364064692692]
```

```
[150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00248: early stopping
[0.785816108339273, 0.785816108339273, 0.785816108339273, 0.7605921518542224]
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00140: early stopping
[0.7521382751247327, 0.7521382751247327, 0.7521382751247327, 0.722902071589828]
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00187: early stopping
[0.7599786172487527, 0.7599786172487527, 0.7599786172487527, 0.731518863771794]
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00147: early stopping
[0.7337847469707769, 0.7337847469707769, 0.733784746970777, 0.7018631579115805]
[200, '0.1']
[0.5771560940841055, 0.5771560940841055, 0.5771560940841055, 0.5250361092757381]
[200, '0.1', 200, '0.1']
Epoch 00227: early stopping
[0.7902708481824662, 0.7902708481824662, 0.7902708481824661, 0.765865311916265]
[200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00246: early stopping
[0.8709907341411262, 0.8709907341411262, 0.8709907341411262, 0.8558632550430805]
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00150: early stopping
[0.8661796151104776, 0.8661796151104776, 0.8661796151104776, 0.8504775774300332]
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00171: early stopping
[0.8736635780470421, 0.8736635780470421, 0.8736635780470421, 0.858903886573265]
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00143: early stopping
[0.8647540983606558, 0.8647540983606558, 0.8647540983606559, 0.8489033371956115]
[200, '0.2']
[0.5718104062722738, 0.5718104062722738, 0.5718104062722738, 0.5193521921585287]
[200, '0.2', 200, '0.2']
[0.7906272273699216, 0.7906272273699216, 0.7906272273699216, 0.7660980388359336]
[200, '0.2', 200, '0.2', 200, '0.2']
[0.8549536707056308, 0.8549536707056308, 0.8549536707056308, 0.8381041473725426]
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
Epoch 00131: early stopping
[0.8232359230220955, 0.8232359230220955, 0.8232359230220955, 0.8025074171854785]
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
Epoch 00219: early stopping
[0.8558446186742694, 0.8558446186742694, 0.8558446186742694, 0.8391218900502941]
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
Epoch 00152: early stopping
[0.8330363506771205, 0.8330363506771205, 0.8330363506771205, 0.8132935419648581]
[200, '0.3']
Epoch 00253: early stopping
[0.5481111903064861, 0.5481111903064861, 0.5481111903064861, 0.4926703109101981]
[200, '0.3', 200, '0.3']
```

```
Epoch 00268: early stopping
[0.7617605131860299, 0.7617605131860299, 0.7617605131860298, 0.7338285623633978]
[200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00245: early stopping
[0.8200285103349965, 0.8200285103349965, 0.8200285103349965, 0.7989672711956026]
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00211: early stopping
[0.8134354953670706, 0.8134354953670706, 0.8134354953670706, 0.7917231289447606]
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00133: early stopping
[0.770848182466144, 0.770848182466144, 0.770848182466144, 0.7438576324656899]
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00159: early stopping
[0.7790449037776194, 0.7790449037776194, 0.7790449037776194, 0.7530774131803226]
[250, '0.1']
[0.5851746258018532, 0.5851746258018532, 0.5851746258018532, 0.5348236817948813]
[250, '0.1', 250, '0.1']
Epoch 00251: early stopping
[0.8057733428367784, 0.8057733428367784, 0.8057733428367784, 0.7828834924115987]
[250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00150: early stopping
[0.8490734141126158, 0.8490734141126158, 0.8490734141126158, 0.8314265685216257]
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00113: early stopping
[0.8610121168923734, 0.8610121168923734, 0.8610121168923734, 0.8447712258954628]
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00109: early stopping
[0.8740199572344975, 0.8740199572344975, 0.8740199572344975, 0.8592490994133825]
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00133: early stopping
[0.8713471133285816, 0.8713471133285816, 0.8713471133285816, 0.8562902115842986]
[250, '0.2']
Epoch 00249: early stopping
[0.5712758374910906, 0.5712758374910906, 0.5712758374910906, 0.5190729423874725]
[250, '0.2', 250, '0.2']
[0.8034568781183179, 0.8034568781183179, 0.8034568781183179, 0.7804932079709793]
[250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00192: early stopping
[0.8526372059871703, 0.8526372059871703, 0.8526372059871702, 0.8354667548029857]
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00129: early stopping
[0.8483606557377049, 0.8483606557377049, 0.8483606557377049, 0.8308142882391145]
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00179: early stopping
[0.8635067712045617, 0.8635067712045617, 0.8635067712045617, 0.8475984137698004]
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00136: early stopping
[0.8390947968638631, 0.8390947968638631, 0.8390947968638631, 0.8203757068695277]
```

```
[250, '0.3']
[0.5659301496792587, 0.5659301496792587, 0.5659301496792587, 0.5130749928933578]
[250, '0.3', 250, '0.3']
Epoch 00266: early stopping
[0.7783321454027085, 0.7783321454027085, 0.7783321454027085, 0.7522834553134814]
[250, '0.3', 250, '0.3', 250, '0.3']
[0.8531717747683535, 0.8531717747683535, 0.8531717747683535, 0.8360999258915827]
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00262: early stopping
[0.8465787598004276, 0.8465787598004276, 0.8465787598004277, 0.8288494408217892]
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00249: early stopping
[0.8406985032074127, 0.8406985032074127, 0.8406985032074127, 0.8221270376207139]
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00210: early stopping
[0.8127227369921597, 0.8127227369921597, 0.8127227369921597, 0.7909136938135756]
```

[64]:
```
joblib.dump(results_dropout, 'results_dropout')
```

[64]:
```
['results_dropout']
```

Dropout* usando *smote Repetimos el procedimiento con estos datos.

[25]:
```
results_dropout_smote = []
seed = 1
```

[26]:
```
size_config = [50, 100, 150, 200, 250]
dropout_rate = ["0.1", "0.2", "0.3"]

for size in size_config:
    for size_d in (dropout_rate):
        layer_config_dense = [[size], [size]*2, [size]*3, [size]*4, [size]*5,␣
↪[size]*6]
        layer_config_dropout = [[size_d], [size_d]*2, [size_d]*3, [size_d]*4,␣
↪[size_d]*5, [size_d]*6]
        for layers_dense, layers_dropout in zip(layer_config_dense,␣
↪layer_config_dropout):
            final_design = [None]*(len(layers_dense)+len(layers_dropout))
            final_design[::2] = layers_dense
            final_design[1::2] = layers_dropout
            np.random.seed(seed)
            tf.random.set_seed(seed)
            print(final_design)
            model = make_my_model_multi_dropout(final_design, 6, 18,␣
↪activation_='relu' )
            preds = compile_fit_multiclass(model, X_train_smote, X_test,␣
↪y_train_smote, 256, 300, verbose=0)
```

```
            metrics = compute_metrics_multiclass(np.argmax(preds, axis = 1), np.
↪argmax(y_test, axis = 1))
            confusion = confusion_matrix(np.argmax(preds, axis = 1), np.
↪argmax(y_test, axis = 1))
            aux = { "layer config" : final_design,
                   #"Model": model,
                   "Predictions" : preds,
                   "Metrics" : metrics,
                   "Confusion" : confusion

            }
            print(metrics)
            results_dropout_smote.append(aux)
```

```
[50, '0.1']
[0.26300784034212404, 0.26300784034212404, 0.26300784034212404,
0.20085519448743994]
[50, '0.1', 50, '0.1']
Epoch 00265: early stopping
[0.2998930862437634, 0.2998930862437634, 0.2998930862437634,
0.23713346452369544]
[50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00199: early stopping
[0.3205630791161796, 0.3205630791161796, 0.3205630791161796, 0.258903190675154]
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00162: early stopping
[0.30648610121168923, 0.30648610121168923, 0.30648610121168923,
0.2430873107926549]
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00142: early stopping
[0.3075552387740556, 0.3075552387740556, 0.3075552387740556,
0.24394819172656956]
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00099: early stopping
[0.28367783321454026, 0.28367783321454026, 0.28367783321454026,
0.22122127360587496]
[50, '0.2']
Epoch 00271: early stopping
[0.2494654312188168, 0.2494654312188168, 0.2494654312188168, 0.1875769065873798]
[50, '0.2', 50, '0.2']
Epoch 00175: early stopping
[0.28421240199572345, 0.28421240199572345, 0.28421240199572345,
0.22168654599358906]
[50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00163: early stopping
[0.27619387027797576, 0.27619387027797576, 0.27619387027797576,
```

```
0.2133303494383716]
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00111: early stopping
[0.2729864575908767, 0.2729864575908767, 0.2729864575908767,
0.21067733393453925]
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00222: early stopping
[0.2920527441197434, 0.2920527441197434, 0.2920527441197434, 0.2270944345574314]
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00159: early stopping
[0.2719173200285103, 0.2719173200285103, 0.2719173200285103,
0.20919106407082955]
[50, '0.3']
Epoch 00216: early stopping
[0.2459016393442623, 0.2459016393442623, 0.2459016393442623,
0.18529233154432556]
[50, '0.3', 50, '0.3']
Epoch 00154: early stopping
[0.2656806842480399, 0.2656806842480399, 0.2656806842480399,
0.20272738533056844]
[50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00121: early stopping
[0.26069137562366357, 0.26069137562366357, 0.26069137562366357,
0.19940871426469609]
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00122: early stopping
[0.24376336421952957, 0.24376336421952957, 0.24376336421952957,
0.17933947264543937]
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00177: early stopping
[0.2275481111903065, 0.2275481111903065, 0.2275481111903065, 0.1614652947632501]
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00160: early stopping
[0.21596578759800428, 0.21596578759800428, 0.21596578759800428,
0.1511632883484848]
[100, '0.1']
[0.2594440484675695, 0.2594440484675695, 0.2594440484675695, 0.1955059345370388]
[100, '0.1', 100, '0.1']
[0.3447968638631504, 0.3447968638631504, 0.3447968638631504,
0.28340547320298026]
[100, '0.1', 100, '0.1', 100, '0.1']
Epoch 00219: early stopping
[0.372416250890948, 0.372416250890948, 0.372416250890948, 0.31370181177426715]
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
Epoch 00185: early stopping
[0.35673556664290806, 0.35673556664290806, 0.35673556664290806,
0.2965402996283344]
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
```

```
Epoch 00164: early stopping
[0.36689237348538845, 0.36689237348538845, 0.3668923734853885,
0.3079995227352187]
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
Epoch 00100: early stopping
[0.35263720598717035, 0.35263720598717035, 0.35263720598717035,
0.29222509720780787]
[100, '0.2']
Epoch 00290: early stopping
[0.25659301496792586, 0.25659301496792586, 0.25659301496792586,
0.19235916152006827]
[100, '0.2', 100, '0.2']
Epoch 00251: early stopping
[0.3143264433357092, 0.3143264433357092, 0.3143264433357092,
0.25120042985546787]
[100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00129: early stopping
[0.3282252316464718, 0.3282252316464718, 0.3282252316464718,
0.26708859834267673]
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00105: early stopping
[0.315751960085531, 0.315751960085531, 0.315751960085531, 0.2545872547346073]
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00191: early stopping
[0.3321454027084818, 0.3321454027084818, 0.3321454027084818,
0.27051519260022494]
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00215: early stopping
[0.3225231646471846, 0.3225231646471846, 0.3225231646471846, 0.2600452708848978]
[100, '0.3']
[0.2553456878118318, 0.2553456878118318, 0.2553456878118318,
0.19117397326811747]
[100, '0.3', 100, '0.3']
Epoch 00267: early stopping
[0.3036350677120456, 0.3036350677120456, 0.3036350677120456,
0.24018452699497383]
[100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00131: early stopping
[0.32430506058446185, 0.32430506058446185, 0.32430506058446185,
0.2625623102105059]
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00178: early stopping
[0.3014967925873129, 0.3014967925873129, 0.3014967925873129,
0.23783108719859625]
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00093: early stopping
[0.2965074839629366, 0.2965074839629366, 0.2965074839629366,
0.23485587148830267]
```

```
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00111: early stopping
[0.33018531717747684, 0.33018531717747684, 0.33018531717747684,
0.26997768943588174]
[150, '0.1']
[0.27049180327868855, 0.27049180327868855, 0.27049180327868855,
0.2070689926269983]
[150, '0.1', 150, '0.1']
Epoch 00236: early stopping
[0.36617961511047753, 0.36617961511047753, 0.36617961511047753,
0.3065969957363909]
[150, '0.1', 150, '0.1', 150, '0.1']
Epoch 00179: early stopping
[0.3544191019244476, 0.3544191019244476, 0.3544191019244476,
0.29432310566920317]
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
Epoch 00151: early stopping
[0.38025659301496795, 0.38025659301496795, 0.38025659301496795,
0.3217359080760902]
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
Epoch 00199: early stopping
[0.38934426229508196, 0.38934426229508196, 0.38934426229508196,
0.3320630520189505]
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
Epoch 00179: early stopping
[0.39468995010691377, 0.39468995010691377, 0.39468995010691377,
0.33666606146261224]
[150, '0.2']
[0.2557020669992872, 0.2557020669992872, 0.2557020669992872,
0.19120997021688535]
[150, '0.2', 150, '0.2']
Epoch 00201: early stopping
[0.3499643620812545, 0.3499643620812545, 0.3499643620812545,
0.28962387662704503]
[150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00227: early stopping
[0.3643977191732003, 0.3643977191732003, 0.3643977191732003, 0.3047718177375881]
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00263: early stopping
[0.3832858161083393, 0.3832858161083393, 0.3832858161083393, 0.3251691504162134]
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00151: early stopping
[0.3891660727013542, 0.3891660727013542, 0.3891660727013542, 0.3318250780448826]
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00069: early stopping
[0.35548823948681396, 0.35548823948681396, 0.3554882394868139,
0.29658654328800893]
[150, '0.3']
```

```
Epoch 00300: early stopping
[0.2540983606557377, 0.2540983606557377, 0.2540983606557377,
0.18919799124540182]
[150, '0.3', 150, '0.3']
Epoch 00230: early stopping
[0.3396293656450463, 0.3396293656450463, 0.3396293656450463, 0.2786682676628811]
[150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00197: early stopping
[0.3677833214540271, 0.3677833214540271, 0.36778332145402703,
0.3088770225577919]
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00140: early stopping
[0.3462223806129722, 0.3462223806129722, 0.3462223806129722,
0.28653161153670004]
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00129: early stopping
[0.3579828937990021, 0.3579828937990021, 0.35798289379900206,
0.29920706564380195]
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00137: early stopping
[0.3549536707056308, 0.3549536707056308, 0.35495367070563083,
0.2956493499086582]
[200, '0.1']
[0.2758374910905203, 0.2758374910905203, 0.2758374910905203,
0.21182638080300087]
[200, '0.1', 200, '0.1']
Epoch 00203: early stopping
[0.36065573770491804, 0.36065573770491804, 0.3606557377049181,
0.30122411585270636]
[200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00187: early stopping
[0.37829650748396293, 0.37829650748396293, 0.37829650748396293,
0.319156284144949]
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00136: early stopping
[0.39522451888809695, 0.39522451888809695, 0.395224518888097,
0.3373918663980583]
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00082: early stopping
[0.40021382751247325, 0.40021382751247325, 0.40021382751247325,
0.34222347641272977]
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00109: early stopping
[0.38649322879543835, 0.38649322879543835, 0.38649322879543835,
0.32731143396239426]
[200, '0.2']
[0.268888096935139, 0.268888096935139, 0.268888096935139, 0.20499839988310808]
[200, '0.2', 200, '0.2']
```

```
Epoch 00167: early stopping
[0.35655737704918034, 0.35655737704918034, 0.3565573770491803,
0.2967274796130971]
[200, '0.2', 200, '0.2', 200, '0.2']
Epoch 00232: early stopping
[0.40324305060584464, 0.40324305060584464, 0.40324305060584464,
0.3467882041439525]
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
Epoch 00161: early stopping
[0.3955808980755524, 0.3955808980755524, 0.3955808980755524, 0.3376063898548457]
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
Epoch 00241: early stopping
[0.3907697790449038, 0.3907697790449038, 0.3907697790449038, 0.3324561672984019]
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
Epoch 00141: early stopping
[0.3679615110477548, 0.3679615110477548, 0.36796151104775476,
0.3083622483336965]
[200, '0.3']
Epoch 00153: early stopping
[0.26924447612259444, 0.26924447612259444, 0.26924447612259444,
0.2058282571547384]
[200, '0.3', 200, '0.3']
Epoch 00175: early stopping
[0.3556664290805417, 0.3556664290805417, 0.3556664290805417, 0.2950894640652668]
[200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00232: early stopping
[0.39326443335709194, 0.39326443335709194, 0.39326443335709194,
0.33580720228153926]
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00232: early stopping
[0.37954383464005703, 0.37954383464005703, 0.37954383464005703,
0.3216607699137475]
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00140: early stopping
[0.3638631503920171, 0.3638631503920171, 0.3638631503920171,
0.30498056948130214]
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00129: early stopping
[0.36689237348538845, 0.36689237348538845, 0.3668923734853885,
0.3069702792714195]
[250, '0.1']
Epoch 00285: early stopping
[0.2753029223093371, 0.2753029223093371, 0.2753029223093371,
0.21098176024808302]
[250, '0.1', 250, '0.1']
Epoch 00271: early stopping
[0.36350677120456165, 0.36350677120456165, 0.36350677120456165,
0.3031787683148758]
```

```
[250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00117: early stopping
[0.39861012116892375, 0.39861012116892375, 0.39861012116892375,
0.3411751823555649]
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00116: early stopping
[0.3923734853884533, 0.3923734853884533, 0.3923734853884533, 0.3343374374058219]
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00126: early stopping
[0.3907697790449038, 0.3907697790449038, 0.3907697790449038,
0.33151015653713056]
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00116: early stopping
[0.4000356379187455, 0.4000356379187455, 0.4000356379187456,
0.34147287960435413]
[250, '0.2']
Epoch 00244: early stopping
[0.2662152530292231, 0.2662152530292231, 0.2662152530292231,
0.20255805666422488]
[250, '0.2', 250, '0.2']
Epoch 00252: early stopping
[0.3699215965787598, 0.3699215965787598, 0.3699215965787598, 0.312080858448489]
[250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00234: early stopping
[0.3995010691375624, 0.3995010691375624, 0.3995010691375624, 0.3421950134241337]
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00159: early stopping
[0.3923734853884533, 0.3923734853884533, 0.3923734853884533, 0.3339574861445036]
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00117: early stopping
[0.3914825374198147, 0.3914825374198147, 0.3914825374198147,
0.33271842857445766]
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00132: early stopping
[0.3995010691375624, 0.3995010691375624, 0.3995010691375624,
0.34119409276086266]
[250, '0.3']
Epoch 00285: early stopping
[0.2690662865288667, 0.2690662865288667, 0.2690662865288667, 0.2052302756253448]
[250, '0.3', 250, '0.3']
Epoch 00242: early stopping
[0.3602993585174626, 0.3602993585174626, 0.3602993585174626, 0.3011976641288431]
[250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00216: early stopping
[0.39611546685673554, 0.39611546685673554, 0.39611546685673554,
0.3388907927897745]
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00160: early stopping
```

```
[0.3923734853884533, 0.3923734853884533, 0.3923734853884533,
0.33487742605489634]
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00174: early stopping
[0.3766928011404134, 0.3766928011404134, 0.3766928011404134, 0.317371421227289]
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00160: early stopping
[0.387384176764077, 0.387384176764077, 0.387384176764077, 0.3296291053670306]
```

[27]:
```
import joblib
joblib.dump(results_dropout_smote, 'results_dropout_smote')
```

[27]:
```
['results_dropout_smote']
```

2.2 Datos codificados con *one_hot* con *smote* y *dropout*

Repetimos los mismos pasos que en el apartado anterior la diferencia de que codificamos los datos de las variables predictoras usando *dummy variables* o codificación *one-hot*. Así, compararemos el rendimiento de los modelos.

[6]:
```
import numpy as np
import pandas as pd
data = pd.read_csv("krkopt.data", header=None)
data.columns = ["wkc", "wkr", "wrc", "wrr", "bkc", "bkr", "opt rank" ]
X = data.iloc[:, 0:6]
y = data['opt rank']
X["wkc"]=X["wkc"].astype('category')
X["wrc"]=X["wrc"].astype('category')
X["bkc"]=X["bkc"].astype('category')
X["wkc"]=X["wkc"].cat.codes
X["wrc"]=X["wrc"].cat.codes
X["bkc"]=X["bkc"].cat.codes
y = y.astype('category')
y = y.cat.codes

from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split

cat_cols = list(X.columns)
```

Codificamos usando *get_dummies* de pandas. Veremos que ahora tenemos 40 variables predictoras.

[7]:
```
X = pd.get_dummies(X,columns=cat_cols)
X
```

[7]:

	wkc_0	wkc_1	wkc_2	wkc_3	wkr_1	wkr_2	wkr_3	wkr_4	wrc_0	wrc_1	\
0	1	0	0	0	1	0	0	0	0	1	
1	1	0	0	0	1	0	0	0	0	0	

```
2          1      0      0      0      1      0      0      0      0      0
3          1      0      0      0      1      0      0      0      0      0
4          1      0      0      0      1      0      0      0      0      0
…      …    …    …    …    …    …    …    …    …    …
28051      0      1      0      0      1      0      0      0      0      0
28052      0      1      0      0      1      0      0      0      0      0
28053      0      1      0      0      1      0      0      0      0      0
28054      0      1      0      0      1      0      0      0      0      0
28055      0      1      0      0      1      0      0      0      0      0

       …  bkc_6  bkc_7  bkr_1  bkr_2  bkr_3  bkr_4  bkr_5  bkr_6  bkr_7  \
0      …      0      0      0      1      0      0      0      0      0
1      …      0      0      0      1      0      0      0      0      0
2      …      0      0      1      0      0      0      0      0      0
3      …      0      0      0      1      0      0      0      0      0
4      …      0      0      1      0      0      0      0      0      0
…    …    …    …    …    …    …    …    …    …    …
28051  …      0      0      0      0      0      0      1      0      0
28052  …      0      0      0      0      0      0      0      1      0
28053  …      0      0      0      0      0      0      0      0      1
28054  …      0      0      0      0      0      0      1      0      0
28055  …      1      0      0      0      0      0      1      0      0

       bkr_8
0          0
1          0
2          0
3          0
4          0
…      …
28051      0
28052      0
28053      0
28054      0
28055      0

[28056 rows x 40 columns]
```

[15]:
```
from tensorflow import keras
import tensorflow as tf
from tensorflow.keras.utils import to_categorical

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y, test_size=0.2,
                                                    random_state = 1)

from imblearn.over_sampling import SMOTE
```

```
oversample = SMOTE()


X_smote, y_smote = oversample.fit_resample(X, y)

X_train_smote, X_test_smote, y_train_smote, y_test_smote =␣
 ↪train_test_split(X_smote,
                                                    y_smote, test_size=0.2,
                                                    random_state = 1)



y_train = to_categorical(y_train)
y_test  = to_categorical(y_test)
y_train_smote = to_categorical(y_train_smote)
y_test_smote = to_categorical(y_test_smote)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

scaler = StandardScaler().fit(X_train_smote)
X_train_smote = scaler.transform(X_train_smote)
X_test_smote = scaler.transform(X_test_smote)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


checkpoint_filepath = '/tmp/checkpoint'

from sklearn.metrics import confusion_matrix, precision_score, \
f1_score, cohen_kappa_score, recall_score
```

2.2.1 Pruebas

Todos los casos son los mismos que en los datos con la anterior codificación. Guardaremos en archivos todos los objetos de cada conjunto de experimentos.

Datos *one_hot*

[48]:
```
results = []
seed = 1
```

[49]:
```
size_config = [50, 100, 150, 200, 250]
for size in size_config:
    layer_config = [[size], [size]*2, [size]*3, [size]*4, [size]*5, [size]*6]
    for layers in layer_config:
        np.random.seed(seed)
        tf.random.set_seed(seed)
        print(layers)
        model = make_my_model_multi(layers, 40, 18, activation_='relu' )
        preds = compile_fit_multiclass(model, X_train, X_test, y_train, 256,␣
 ↪300, verbose=0)
        metrics = compute_metrics_multiclass(np.argmax(preds, axis = 1), np.
 ↪argmax(y_test, axis = 1))
        confusion = confusion_matrix(np.argmax(preds, axis = 1), np.
 ↪argmax(y_test, axis = 1))
        aux = { "layer config" : layers,
               #"Model": model,
               "Predictions" : preds,
               "Metrics" : metrics,
               "Confusion" : confusion

        }
        print(metrics)
        results.append(aux)
```

```
[50]
[0.7132929436920884, 0.7132929436920884, 0.7132929436920883, 0.6795940399725571]
[50, 50]
Epoch 00118: early stopping
[0.7359230220955096, 0.7359230220955096, 0.7359230220955095, 0.7051213677686434]
[50, 50, 50]
Epoch 00075: early stopping
[0.7464362081254454, 0.7464362081254454, 0.7464362081254454, 0.7169690264820239]
[50, 50, 50, 50]
Epoch 00059: early stopping
[0.7293300071275838, 0.7293300071275838, 0.7293300071275838, 0.6976471559645523]
[50, 50, 50, 50, 50]
Epoch 00053: early stopping
[0.7323592302209551, 0.7323592302209551, 0.7323592302209551, 0.7011602714040706]
[50, 50, 50, 50, 50, 50]
Epoch 00054: early stopping
[0.7245188880969351, 0.7245188880969351, 0.7245188880969351, 0.6923109986216298]
[100]
Epoch 00176: early stopping
[0.7405559515324305, 0.7405559515324305, 0.7405559515324305, 0.7101055066873646]
[100, 100]
Epoch 00053: early stopping
```

```
[0.7624732715609408, 0.7624732715609408, 0.7624732715609408, 0.7348246606533331]
[100, 100, 100]
Epoch 00042: early stopping
[0.7859942979330007, 0.7859942979330007, 0.7859942979330007, 0.761075013060205]
[100, 100, 100, 100]
Epoch 00039: early stopping
[0.7845687811831789, 0.7845687811831789, 0.7845687811831789, 0.7594270506445013]
[100, 100, 100, 100, 100]
Epoch 00034: early stopping
[0.7802922309337135, 0.7802922309337135, 0.7802922309337134, 0.7544084918576515]
[100, 100, 100, 100, 100, 100]
Epoch 00029: early stopping
[0.7590876692801141, 0.7590876692801141, 0.7590876692801141, 0.7307269833855377]
[150]
Epoch 00150: early stopping
[0.7587312900926586, 0.7587312900926586, 0.7587312900926585, 0.7304863274796372]
[150, 150]
Epoch 00059: early stopping
[0.8080898075552387, 0.8080898075552387, 0.8080898075552387, 0.7856176438592615]
[150, 150, 150]
Epoch 00038: early stopping
[0.8071988595866001, 0.8071988595866001, 0.8071988595866, 0.7847174514580395]
[150, 150, 150, 150]
Epoch 00030: early stopping
[0.7995367070563079, 0.7995367070563079, 0.799536707056308, 0.7760993871183968]
[150, 150, 150, 150, 150]
Epoch 00024: early stopping
[0.785816108339273, 0.785816108339273, 0.785816108339273, 0.760816977877812]
[150, 150, 150, 150, 150, 150]
Epoch 00026: early stopping
[0.8013186029935851, 0.8013186029935851, 0.8013186029935851, 0.7778395069749614]
[200]
Epoch 00108: early stopping
[0.7633642195295794, 0.7633642195295794, 0.7633642195295794, 0.7355328449643446]
[200, 200]
Epoch 00041: early stopping
[0.8031004989308624, 0.8031004989308624, 0.8031004989308624, 0.7800639075355223]
[200, 200, 200]
Epoch 00032: early stopping
[0.8096935138987883, 0.8096935138987883, 0.8096935138987883, 0.787236216263784]
[200, 200, 200, 200]
Epoch 00031: early stopping
[0.8308980755523877, 0.8308980755523877, 0.8308980755523878, 0.8111400326514505]
[200, 200, 200, 200, 200]
Epoch 00028: early stopping
[0.8145046329294369, 0.8145046329294369, 0.8145046329294369, 0.7932395201038133]
[200, 200, 200, 200, 200, 200]
Epoch 00020: early stopping
```

```
[0.7856379187455452, 0.7856379187455452, 0.7856379187455452, 0.7609006763147126]
[250]
Epoch 00093: early stopping
[0.7756593014967926, 0.7756593014967926, 0.7756593014967926, 0.74921524924313]
[250, 250]
Epoch 00039: early stopping
[0.8241268709907341, 0.8241268709907341, 0.8241268709907341, 0.8035364436115411]
[250, 250, 250]
Epoch 00030: early stopping
[0.8301853171774768, 0.8301853171774768, 0.8301853171774768, 0.8104398918945435]
[250, 250, 250, 250]
Epoch 00023: early stopping
[0.8171774768353528, 0.8171774768353528, 0.8171774768353528, 0.7955487752872001]
[250, 250, 250, 250, 250]
Epoch 00022: early stopping
[0.8038132573057734, 0.8038132573057734, 0.8038132573057732, 0.7810034973873443]
[250, 250, 250, 250, 250, 250]
Epoch 00022: early stopping
[0.8084461867426942, 0.8084461867426942, 0.8084461867426942, 0.7861858456085945]
```

[50]:
```
import joblib

joblib.dump(results, 'results_1_onehot_joblib')
```

[50]:
```
['results_1_onehot_joblib']
```

2.2.2 Datos *one_hot* con *SMOTE*

[51]:
```
results_smote = []
seed = 1
```

[52]:
```
size_config = [50, 100, 150, 200, 250]
for size in size_config:
    layer_config = [[size], [size]*2, [size]*3, [size]*4, [size]*5, [size]*6]
    for layers in layer_config:
        np.random.seed(seed)
        tf.random.set_seed(seed)
        print(layers)
        model = make_my_model_multi(layers, 40, 18, activation_='relu' )
        preds = compile_fit_multiclass(model, X_train_smote, X_test,␣
 ↪y_train_smote, 256, 300, verbose=0)
        metrics = compute_metrics_multiclass(np.argmax(preds, axis = 1), np.
 ↪argmax(y_test, axis = 1))
        confusion = confusion_matrix(np.argmax(preds, axis = 1), np.
 ↪argmax(y_test, axis = 1))
        aux = { "layer config" : layers,
               #"Model": model,
```

```
            "Predictions" : preds,
            "Metrics" : metrics,
            "Confusion" : confusion

        }
        print(metrics)
        results_smote.append(aux)
```

```
[50]
[0.09853884533143265, 0.09853884533143265, 0.09853884533143265,
0.05990480607973814]
[50, 50]
Epoch 00270: early stopping
[0.06931575196008553, 0.06931575196008553, 0.06931575196008553,
0.037006904026981036]
[50, 50, 50]
Epoch 00155: early stopping
[0.06967213114754098, 0.06967213114754098, 0.06967213114754098,
0.03441175321042145]
[50, 50, 50, 50]
Epoch 00153: early stopping
[0.07448325017818959, 0.07448325017818959, 0.07448325017818959,
0.04118489587487906]
[50, 50, 50, 50, 50]
Epoch 00120: early stopping
[0.06789023521026372, 0.06789023521026372, 0.06789023521026372,
0.028956419894087593]
[50, 50, 50, 50, 50, 50]
Epoch 00200: early stopping
[0.05434782608695652, 0.05434782608695652, 0.05434782608695652,
0.018471319233337335]
[100]
[0.11101211689237349, 0.11101211689237349, 0.11101211689237349,
0.07134215115338971]
[100, 100]
Epoch 00285: early stopping
[0.1097647897362794, 0.1097647897362794, 0.1097647897362794,
0.07580671664360639]
[100, 100, 100]
Epoch 00226: early stopping
[0.11279401282965075, 0.11279401282965075, 0.11279401282965075,
0.0702367187003482]
[100, 100, 100, 100]
Epoch 00157: early stopping
[0.09764789736279401, 0.09764789736279401, 0.09764789736279401,
0.058277308802495265]
```

```
[100, 100, 100, 100, 100]
Epoch 00162: early stopping
[0.07947255880256593, 0.07947255880256593, 0.07947255880256593,
0.04168589497606212]
[100, 100, 100, 100, 100, 100]
Epoch 00154: early stopping
[0.07662152530292231, 0.07662152530292231, 0.07662152530292231,
0.041859363311582354]
[150]
[0.11119030648610122, 0.11119030648610122, 0.11119030648610122,
0.07057217306767738]
[150, 150]
Epoch 00192: early stopping
[0.12918745545260157, 0.12918745545260157, 0.12918745545260157,
0.09148213445074216]
[150, 150, 150]
Epoch 00166: early stopping
[0.13809693513898788, 0.13809693513898788, 0.13809693513898788,
0.0914795402998081]
[150, 150, 150, 150]
Epoch 00140: early stopping
[0.10816108339272987, 0.10816108339272987, 0.10816108339272985,
0.06738482220517195]
[150, 150, 150, 150, 150]
Epoch 00109: early stopping
[0.06699928724162509, 0.06699928724162509, 0.06699928724162509,
0.03164283938023682]
[150, 150, 150, 150, 150, 150]
Epoch 00093: early stopping
[0.06254454739843193, 0.06254454739843193, 0.06254454739843193,
0.02622167248184981]
[200]
Epoch 00192: early stopping
[0.09461867426942266, 0.09461867426942266, 0.09461867426942266,
0.0597149180355222]
[200, 200]
Epoch 00254: early stopping
[0.1876336421952958, 0.1876336421952958, 0.1876336421952958,
0.14429603051691353]
[200, 200, 200]
Epoch 00120: early stopping
[0.15217391304347827, 0.15217391304347827, 0.15217391304347827,
0.10779708923931663]
[200, 200, 200, 200]
Epoch 00129: early stopping
[0.1003207412687099, 0.1003207412687099, 0.1003207412687099,
0.06150141024915745]
[200, 200, 200, 200, 200]
```

```
Epoch 00081: early stopping
[0.08357091945830364, 0.08357091945830364, 0.08357091945830364,
0.047671217979086355]
[200, 200, 200, 200, 200, 200]
Epoch 00082: early stopping
[0.0792943692088382, 0.0792943692088382, 0.0792943692088382, 0.0425269634720844]
[250]
Epoch 00254: early stopping
[0.10655737704918032, 0.10655737704918032, 0.10655737704918032,
0.07031624765828526]
[250, 250]
Epoch 00174: early stopping
[0.14593727726300784, 0.14593727726300784, 0.14593727726300784,
0.10398057068418642]
[250, 250, 250]
Epoch 00106: early stopping
[0.17783321454027085, 0.17783321454027085, 0.17783321454027085,
0.13176963959600485]
[250, 250, 250, 250]
Epoch 00087: early stopping
[0.1172487526728439, 0.1172487526728439, 0.1172487526728439,
0.07717883510268653]
[250, 250, 250, 250, 250]
Epoch 00081: early stopping
[0.08802565930149679, 0.08802565930149679, 0.08802565930149679,
0.05372309817465015]
[250, 250, 250, 250, 250, 250]
Epoch 00058: early stopping
[0.08214540270848182, 0.08214540270848182, 0.08214540270848182,
0.047641839366908134]
```

[53]:
```
joblib.dump(results_smote, 'results_smote_onehot_joblib')
```

[53]:
```
['results_smote_onehot_joblib']
```

2.2.3 Datos *one_hot* con *dropout*

[54]:
```
results_dropout = []
seed = 1
```

[55]:
```
size_config = [50, 100, 150, 200, 250]
dropout_rate = ["0.1", "0.2", "0.3"]

for size in size_config:
    for size_d in (dropout_rate):
        layer_config_dense = [[size], [size]*2, [size]*3, [size]*4, [size]*5,␣
↪[size]*6]
```

```
      layer_config_dropout = [[size_d], [size_d]*2, [size_d]*3, [size_d]*4,␣
↪[size_d]*5, [size_d]*6]
      for layers_dense, layers_dropout in zip(layer_config_dense,␣
↪layer_config_dropout):
          final_design = [None]*(len(layers_dense)+len(layers_dropout))
          final_design[::2] = layers_dense
          final_design[1::2] = layers_dropout
          np.random.seed(seed)
          tf.random.set_seed(seed)
          print(final_design)
          model = make_my_model_multi_dropout(final_design, 40, 18,␣
↪activation_='relu' )
          preds = compile_fit_multiclass(model, X_train, X_test, y_train,␣
↪256, 300, verbose=0)
          metrics = compute_metrics_multiclass(np.argmax(preds, axis = 1), np.
↪argmax(y_test, axis = 1))
          confusion = confusion_matrix(np.argmax(preds, axis = 1), np.
↪argmax(y_test, axis = 1))
          aux = { "layer config" : final_design,
                 #"Model": model,
                 "Predictions" : preds,
                 "Metrics" : metrics,
                 "Confusion" : confusion

          }
          print(metrics)
          results_dropout.append(aux)
```

```
[50, '0.1']
[0.6911974340698503, 0.6911974340698503, 0.6911974340698503, 0.6544424729231852]
[50, '0.1', 50, '0.1']
Epoch 00249: early stopping
[0.7549893086243763, 0.7549893086243763, 0.7549893086243763, 0.7260805176671765]
[50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00244: early stopping
[0.7685317177476836, 0.7685317177476836, 0.7685317177476836, 0.7412666342935919]
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00204: early stopping
[0.7628296507483963, 0.7628296507483963, 0.7628296507483963, 0.7349489199206065]
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00204: early stopping
[0.755167498218104, 0.755167498218104, 0.755167498218104, 0.726566737550184]
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00138: early stopping
[0.7446543121881682, 0.7446543121881682, 0.7446543121881682, 0.7145851936293055]
[50, '0.2']
```

```
[0.6797933000712758, 0.6797933000712758, 0.6797933000712758, 0.6415147506179972]
[50, '0.2', 50, '0.2']
Epoch 00194: early stopping
[0.7211332858161084, 0.7211332858161084, 0.7211332858161085, 0.6880538576802332]
[50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00202: early stopping
[0.7189950106913756, 0.7189950106913756, 0.7189950106913756, 0.6857954888177956]
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00177: early stopping
[0.7049180327868853, 0.7049180327868853, 0.7049180327868853, 0.6696354479193032]
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00184: early stopping
[0.7040270848182466, 0.7040270848182466, 0.7040270848182466, 0.6688250607918782]
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00225: early stopping
[0.6983250178189594, 0.6983250178189594, 0.6983250178189594, 0.6625351015506182]
[50, '0.3']
Epoch 00290: early stopping
[0.6584105488239487, 0.6584105488239487, 0.6584105488239487, 0.6172402972361986]
[50, '0.3', 50, '0.3']
Epoch 00182: early stopping
[0.6837134711332858, 0.6837134711332858, 0.6837134711332858, 0.6456665282835925]
[50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00152: early stopping
[0.66928011404134, 0.66928011404134, 0.66928011404134, 0.6293258262824244]
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00182: early stopping
[0.655559515324305, 0.655559515324305, 0.655559515324305, 0.6142799530678658]
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00192: early stopping
[0.6407697790449037, 0.6407697790449037, 0.6407697790449037, 0.5974540868981456]
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00216: early stopping
[0.6076265146115467, 0.6076265146115467, 0.6076265146115467, 0.5617307756281984]
[100, '0.1']
Epoch 00267: early stopping
[0.7537419814682823, 0.7537419814682823, 0.7537419814682823, 0.7247488644953375]
[100, '0.1', 100, '0.1']
Epoch 00193: early stopping
[0.8221667854597291, 0.8221667854597291, 0.8221667854597291, 0.8014009505335618]
[100, '0.1', 100, '0.1', 100, '0.1']
Epoch 00182: early stopping
[0.8504989308624377, 0.8504989308624377, 0.8504989308624377, 0.8330632580919843]
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
Epoch 00146: early stopping
[0.8480042765502495, 0.8480042765502495, 0.8480042765502495, 0.8302515913483358]
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
Epoch 00148: early stopping
```

```
[0.8478260869565217, 0.8478260869565217, 0.8478260869565218, 0.8300096411794345]
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
Epoch 00150: early stopping
[0.8460441910192444, 0.8460441910192444, 0.8460441910192444, 0.8281644104059738]
[100, '0.2']
[0.7535637918745546, 0.7535637918745546, 0.7535637918745546, 0.7243112074616954]
[100, '0.2', 100, '0.2']
Epoch 00200: early stopping
[0.8059515324305061, 0.8059515324305061, 0.8059515324305061, 0.7831509826329692]
[100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00258: early stopping
[0.8403421240199572, 0.8403421240199572, 0.8403421240199572, 0.8216186407532786]
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00189: early stopping
[0.8145046329294369, 0.8145046329294369, 0.8145046329294369, 0.7927907643878691]
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00129: early stopping
[0.792943692088382, 0.792943692088382, 0.792943692088382, 0.7687186237795841]
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00236: early stopping
[0.8111190306486101, 0.8111190306486101, 0.8111190306486101, 0.7890115939135013]
[100, '0.3']
[0.7442979330007128, 0.7442979330007128, 0.7442979330007128, 0.7138441190567824]
[100, '0.3', 100, '0.3']
Epoch 00208: early stopping
[0.7888453314326443, 0.7888453314326443, 0.7888453314326443, 0.7640197609347319]
[100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00221: early stopping
[0.8061297220242338, 0.8061297220242338, 0.8061297220242338, 0.7833711703582897]
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00153: early stopping
[0.7676407697790449, 0.7676407697790449, 0.7676407697790449, 0.7403424377519607]
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00156: early stopping
[0.7667498218104063, 0.7667498218104063, 0.7667498218104063, 0.7393739781195857]
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00176: early stopping
[0.7414468995010691, 0.7414468995010691, 0.7414468995010691, 0.7111252105230017]
[150, '0.1']
[0.7820741268709908, 0.7820741268709908, 0.7820741268709906, 0.7564243977333958]
[150, '0.1', 150, '0.1']
Epoch 00129: early stopping
[0.855488239486814, 0.855488239486814, 0.855488239486814, 0.8385669050046001]
[150, '0.1', 150, '0.1', 150, '0.1']
Epoch 00079: early stopping
[0.8590520313613685, 0.8590520313613685, 0.8590520313613685, 0.8426740003139367]
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
Epoch 00109: early stopping
```

```
[0.8781183178902352, 0.8781183178902352, 0.8781183178902352, 0.8639675844777932]
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
Epoch 00094: early stopping
[0.8700997861724875, 0.8700997861724875, 0.8700997861724875, 0.8549745938221411]
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
Epoch 00093: early stopping
[0.8695652173913043, 0.8695652173913043, 0.8695652173913043, 0.8543679160066875]
[150, '0.2']
Epoch 00248: early stopping
[0.7751247327156094, 0.7751247327156094, 0.7751247327156094, 0.7486044176314802]
[150, '0.2', 150, '0.2']
Epoch 00220: early stopping
[0.860655737704918, 0.860655737704918, 0.860655737704918, 0.844376829162256]
[150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00231: early stopping
[0.8761582323592302, 0.8761582323592302, 0.8761582323592302, 0.8616730939921385]
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00199: early stopping
[0.8679615110477548, 0.8679615110477548, 0.8679615110477547, 0.852580315610961]
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00136: early stopping
[0.8601211689237348, 0.8601211689237348, 0.8601211689237348, 0.8437704300336455]
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00139: early stopping
[0.8535281539558089, 0.8535281539558089, 0.8535281539558089, 0.8364105315970465]
[150, '0.3']
Epoch 00253: early stopping
[0.7713827512473271, 0.7713827512473271, 0.7713827512473271, 0.744263007539312]
[150, '0.3', 150, '0.3']
Epoch 00214: early stopping
[0.8458660014255167, 0.8458660014255167, 0.8458660014255168, 0.8278767855942375]
[150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00220: early stopping
[0.840520313613685, 0.840520313613685, 0.840520313613685, 0.8219800613018564]
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00173: early stopping
[0.8300071275837491, 0.8300071275837491, 0.8300071275837491, 0.8101729751256426]
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00171: early stopping
[0.8218104062722738, 0.8218104062722738, 0.8218104062722738, 0.8010098078549712]
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00130: early stopping
[0.7952601568068425, 0.7952601568068425, 0.7952601568068425, 0.7712983060607658]
[200, '0.1']
Epoch 00228: early stopping
[0.7961511047754811, 0.7961511047754811, 0.796151104775481, 0.7720837157452116]
[200, '0.1', 200, '0.1']
Epoch 00097: early stopping
```

```
[0.8569137562366358, 0.8569137562366358, 0.8569137562366358, 0.8402549944374698]
[200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00098: early stopping
[0.8884533143264434, 0.8884533143264434, 0.8884533143264434, 0.8754311944881142]
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00065: early stopping
[0.8700997861724875, 0.8700997861724875, 0.8700997861724875, 0.854908986256218]
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00077: early stopping
[0.8845331432644333, 0.8845331432644333, 0.8845331432644333, 0.8710306582427136]
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00068: early stopping
[0.8734853884533144, 0.8734853884533144, 0.8734853884533144, 0.8588150087272403]
[200, '0.2']
Epoch 00292: early stopping
[0.7993585174625801, 0.7993585174625801, 0.7993585174625801, 0.7756947986927558]
[200, '0.2', 200, '0.2']
Epoch 00173: early stopping
[0.8761582323592302, 0.8761582323592302, 0.8761582323592302, 0.8616832826121679]
[200, '0.2', 200, '0.2', 200, '0.2']
Epoch 00178: early stopping
[0.8929080541696365, 0.8929080541696365, 0.8929080541696365, 0.8804258424474269]
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
Epoch 00122: early stopping
[0.8889878831076266, 0.8889878831076266, 0.8889878831076266, 0.876025663923951]
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
Epoch 00113: early stopping
[0.8752672843905915, 0.8752672843905915, 0.8752672843905915, 0.86073847814212]
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
Epoch 00162: early stopping
[0.8863150392017106, 0.8863150392017106, 0.8863150392017106, 0.8731436557602646]
[200, '0.3']
[0.7986457590876693, 0.7986457590876693, 0.7986457590876693, 0.7748684094073761]
[200, '0.3', 200, '0.3']
Epoch 00230: early stopping
[0.8674269422665716, 0.8674269422665716, 0.8674269422665717, 0.8519922476261953]
[200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00214: early stopping
[0.8788310762651461, 0.8788310762651461, 0.8788310762651462, 0.8646983800933996]
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00172: early stopping
[0.8717034925160371, 0.8717034925160371, 0.8717034925160371, 0.8566902616204991]
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00223: early stopping
[0.8738417676407698, 0.8738417676407698, 0.8738417676407699, 0.8591722739548441]
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00165: early stopping
[0.8581610833927299, 0.8581610833927299, 0.8581610833927299, 0.8415628528427186]
```

```
[250, '0.1']
Epoch 00175: early stopping
[0.7995367070563079, 0.7995367070563079, 0.799536707056308, 0.7759209362036418]
[250, '0.1', 250, '0.1']
Epoch 00089: early stopping
[0.8727726300784034, 0.8727726300784034, 0.8727726300784034, 0.8579218239707193]
[250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00071: early stopping
[0.8864932287954383, 0.8864932287954383, 0.8864932287954383, 0.873233938730633]
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00054: early stopping
[0.8807911617961511, 0.8807911617961511, 0.8807911617961511, 0.866858669993142]
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00069: early stopping
[0.8875623663578047, 0.8875623663578047, 0.8875623663578046, 0.8744596060520364]
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00087: early stopping
[0.894511760513186, 0.894511760513186, 0.894511760513186, 0.8821913607432853]
[250, '0.2']
Epoch 00177: early stopping
[0.7922309337134711, 0.7922309337134711, 0.7922309337134711, 0.7676712411254928]
[250, '0.2', 250, '0.2']
Epoch 00183: early stopping
[0.8918389166072701, 0.8918389166072701, 0.8918389166072701, 0.8792620364600945]
[250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00093: early stopping
[0.8877405559515325, 0.8877405559515325, 0.8877405559515325, 0.8746325738298955]
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00096: early stopping
[0.8872059871703493, 0.8872059871703493, 0.8872059871703494, 0.874061027009303]
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00090: early stopping
[0.8800784034212402, 0.8800784034212402, 0.8800784034212402, 0.8661096664833664]
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00124: early stopping
[0.8866714183891661, 0.8866714183891661, 0.8866714183891661, 0.8735009741103202]
[250, '0.3']
Epoch 00273: early stopping
[0.8107626514611547, 0.8107626514611547, 0.8107626514611547, 0.7883434831162922]
[250, '0.3', 250, '0.3']
Epoch 00182: early stopping
[0.8809693513898789, 0.8809693513898789, 0.8809693513898789, 0.8671143715434837]
[250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00133: early stopping
[0.8847113328581611, 0.8847113328581611, 0.884711332858161, 0.8712432971205801]
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00164: early stopping
[0.8850677120456165, 0.8850677120456165, 0.8850677120456164, 0.871636611045392]
```

```
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00189: early stopping
[0.8875623663578047, 0.8875623663578047, 0.8875623663578046, 0.874455680487535]
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00142: early stopping
[0.8711689237348539, 0.8711689237348539, 0.871168923734854, 0.856223700873793]
```

[56]:
```
joblib.dump(results_dropout, 'results_dropout_one_hot')
```

[56]:
```
['results_dropout_one_hot']
```

Datos *one_hot SMOTE dropout*

[28]:
```
results_dropout_smote = []
seed = 1
```

[29]:
```
size_config = [50, 100, 150, 200, 250]
dropout_rate = ["0.1", "0.2", "0.3"]

for size in size_config:
    for size_d in (dropout_rate):
        layer_config_dense = [[size], [size]*2, [size]*3, [size]*4, [size]*5,␣
↪[size]*6]
        layer_config_dropout = [[size_d], [size_d]*2, [size_d]*3, [size_d]*4,␣
↪[size_d]*5, [size_d]*6]
        for layers_dense, layers_dropout in zip(layer_config_dense,␣
↪layer_config_dropout):
            final_design = [None]*(len(layers_dense)+len(layers_dropout))
            final_design[::2] = layers_dense
            final_design[1::2] = layers_dropout
            np.random.seed(seed)
            tf.random.set_seed(seed)
            print(final_design)
            model = make_my_model_multi_dropout(final_design, 40, 18,␣
↪activation_='relu' )
            preds = compile_fit_multiclass(model, X_train_smote, X_test,␣
↪y_train_smote, 256, 300, verbose=0)
            metrics = compute_metrics_multiclass(np.argmax(preds, axis = 1), np.
↪argmax(y_test, axis = 1))
            confusion = confusion_matrix(np.argmax(preds, axis = 1), np.
↪argmax(y_test, axis = 1))
            aux = { "layer config" : final_design,
                    #"Model": model,
                    "Predictions" : preds,
                    "Metrics" : metrics,
                    "Confusion" : confusion
```

```
            }
            print(metrics)
            results_dropout_smote.append(aux)
```

```
[50, '0.1']
[0.17925873129009265, 0.17925873129009265, 0.17925873129009265,
0.12232832383335368]
[50, '0.1', 50, '0.1']
[0.1261582323592302, 0.1261582323592302, 0.1261582323592302,
0.08089709192596017]
[50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00248: early stopping
[0.0650392017106201, 0.0650392017106201, 0.0650392017106201,
0.03040751666739172]
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00189: early stopping
[0.042230933713471135, 0.042230933713471135, 0.042230933713471135,
0.01270048583862049]
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00194: early stopping
[0.038132573057733425, 0.038132573057733425, 0.038132573057733425,
0.005452686722108413]
[50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1', 50, '0.1']
Epoch 00232: early stopping
[0.0345687811831789, 0.0345687811831789, 0.0345687811831789,
0.008312049551169154]
[50, '0.2']
[0.1626870990734141, 0.1626870990734141, 0.1626870990734141,
0.10466572690499598]
[50, '0.2', 50, '0.2']
Epoch 00255: early stopping
[0.13863150392017107, 0.13863150392017107, 0.13863150392017107,
0.085656992318495]
[50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00128: early stopping
[0.05737704918032787, 0.05737704918032787, 0.05737704918032787,
0.02477694026158095]
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00191: early stopping
[0.03189593727726301, 0.03189593727726301, 0.03189593727726301,
0.0039150488723641574]
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
Epoch 00126: early stopping
[0.033856022808267994, 0.033856022808267994, 0.033856022808267994,
0.009683362769514314]
[50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2', 50, '0.2']
```

```
Epoch 00153: early stopping
[0.043478260869565216, 0.043478260869565216, 0.043478260869565216,
0.021719267754409355]
[50, '0.3']
[0.17587312900926586, 0.17587312900926586, 0.17587312900926586,
0.12135664118870049]
[50, '0.3', 50, '0.3']
Epoch 00250: early stopping
[0.101568068424804, 0.101568068424804, 0.101568068424804, 0.05585754164033441]
[50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00186: early stopping
[0.049002138275124736, 0.049002138275124736, 0.04900213827512473,
0.025606993164014047]
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00152: early stopping
[0.023521026372059873, 0.023521026372059873, 0.023521026372059876,
-0.0008839246776410903]
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00122: early stopping
[0.019600855310049892, 0.019600855310049892, 0.019600855310049892,
0.0035310058483916107]
[50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3', 50, '0.3']
Epoch 00110: early stopping
[0.02690662865288667, 0.02690662865288667, 0.02690662865288667,
0.009679007414853946]
[100, '0.1']
[0.20224518888096935, 0.20224518888096935, 0.20224518888096935,
0.14544292498918487]
[100, '0.1', 100, '0.1']
Epoch 00240: early stopping
[0.24679258731290094, 0.24679258731290094, 0.24679258731290094,
0.1934243712900997]
[100, '0.1', 100, '0.1', 100, '0.1']
Epoch 00198: early stopping
[0.09230220955096223, 0.09230220955096223, 0.09230220955096224,
0.05140908127877386]
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
[0.07305773342836779, 0.07305773342836779, 0.07305773342836779,
0.03229868220347709]
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
Epoch 00239: early stopping
[0.05238774055595153, 0.05238774055595153, 0.05238774055595153,
0.01593217893389176]
[100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1', 100, '0.1']
Epoch 00220: early stopping
[0.05327868852459016, 0.05327868852459016, 0.05327868852459016,
0.018773343435891765]
[100, '0.2']
```

```
[0.2241625089094797, 0.2241625089094797, 0.2241625089094797,
0.16380451668419682]
[100, '0.2', 100, '0.2']
[0.20990734141126158, 0.20990734141126158, 0.20990734141126158,
0.1564546087020312]
[100, '0.2', 100, '0.2', 100, '0.2']
[0.0935495367070563, 0.0935495367070563, 0.0935495367070563,
0.049653103710961655]
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00172: early stopping
[0.039379900213827514, 0.039379900213827514, 0.039379900213827514,
0.010444041860423137]
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
[0.06147540983606557, 0.06147540983606557, 0.06147540983606557,
0.016954035230797748]
[100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2', 100, '0.2']
Epoch 00139: early stopping
[0.05915894511760513, 0.05915894511760513, 0.05915894511760513,
0.018481805134154428]
[100, '0.3']
[0.20420527441197434, 0.20420527441197434, 0.20420527441197434,
0.146919703577035]
[100, '0.3', 100, '0.3']
[0.17230933713471133, 0.17230933713471133, 0.17230933713471133,
0.11901675774047382]
[100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00193: early stopping
[0.042943692088382036, 0.042943692088382036, 0.042943692088382036,
0.01206205191427645]
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00164: early stopping
[0.03991446899501069, 0.03991446899501069, 0.03991446899501069,
0.00613134448216468]
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00189: early stopping
[0.04240912330719886, 0.04240912330719886, 0.04240912330719886,
0.00991607580343623]
[100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3', 100, '0.3']
Epoch 00179: early stopping
[0.03563791874554526, 0.03563791874554526, 0.03563791874554526,
0.012061464890697371]
[150, '0.1']
[0.1970776906628653, 0.1970776906628653, 0.1970776906628653,
0.14580931580526268]
[150, '0.1', 150, '0.1']
[0.3145046329294369, 0.3145046329294369, 0.3145046329294369,
0.26160453102930736]
[150, '0.1', 150, '0.1', 150, '0.1']
```

```
Epoch 00217: early stopping
[0.17836778332145403, 0.17836778332145403, 0.17836778332145403,
0.13225478504378496]
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
[0.09390591589451176, 0.09390591589451176, 0.09390591589451176,
0.057542665344226474]
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
[0.0718104062722737, 0.0718104062722737, 0.0718104062722737,
0.033419143155595354]
[150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1', 150, '0.1']
Epoch 00233: early stopping
[0.0616535994297933, 0.0616535994297933, 0.0616535994297933,
0.02521055524335547]
[150, '0.2']
[0.22042052744119744, 0.22042052744119744, 0.22042052744119744,
0.1651157706698071]
[150, '0.2', 150, '0.2']
Epoch 00264: early stopping
[0.23431931575196008, 0.23431931575196008, 0.23431931575196008,
0.1814566324589243]
[150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00181: early stopping
[0.12883107626514612, 0.12883107626514612, 0.12883107626514612,
0.08123675934587382]
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00227: early stopping
[0.08125445473984319, 0.08125445473984319, 0.08125445473984319,
0.042625704486619176]
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00284: early stopping
[0.05666429080541696, 0.05666429080541696, 0.05666429080541696,
0.022088509992709504]
[150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2', 150, '0.2']
Epoch 00206: early stopping
[0.04686386315039202, 0.04686386315039202, 0.04686386315039202,
0.013243538873656369]
[150, '0.3']
[0.2166785459729152, 0.2166785459729152, 0.2166785459729152,
0.16013192583393254]
[150, '0.3', 150, '0.3']
Epoch 00269: early stopping
[0.20153243050605846, 0.20153243050605846, 0.20153243050605846,
0.1489163218797086]
[150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00233: early stopping
[0.09141126158232359, 0.09141126158232359, 0.09141126158232359,
0.05088555218453383]
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
```

```
Epoch 00235: early stopping
[0.048467569493941556, 0.048467569493941556, 0.048467569493941556,
0.01460299355630068]
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00239: early stopping
[0.04436920883820385, 0.04436920883820385, 0.044369208838203854,
0.0130637778773115]
[150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3', 150, '0.3']
Epoch 00229: early stopping
[0.04526015680684248, 0.04526015680684248, 0.045260156806842484,
0.007616860340662668]
[200, '0.1']
[0.2533856022808268, 0.2533856022808268, 0.2533856022808268, 0.197391515941576]
[200, '0.1', 200, '0.1']
[0.3708125445473984, 0.3708125445473984, 0.3708125445473984, 0.3210530887142903]
[200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00276: early stopping
[0.22202423378474698, 0.22202423378474698, 0.22202423378474698,
0.17342244981580968]
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00187: early stopping
[0.11546685673556664, 0.11546685673556664, 0.11546685673556664,
0.07693413882061018]
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00295: early stopping
[0.09248039914468995, 0.09248039914468995, 0.09248039914468995,
0.054437648776313186]
[200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1', 200, '0.1']
Epoch 00223: early stopping
[0.07056307911617961, 0.07056307911617961, 0.07056307911617961,
0.03598036748298006]
[200, '0.2']
[0.2569493941553813, 0.2569493941553813, 0.2569493941553813, 0.1990989622048731]
[200, '0.2', 200, '0.2']
Epoch 00300: early stopping
[0.28813257305773343, 0.28813257305773343, 0.28813257305773343,
0.23566194103112692]
[200, '0.2', 200, '0.2', 200, '0.2']
Epoch 00297: early stopping
[0.216143977191732, 0.216143977191732, 0.216143977191732, 0.16476544862583697]
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
Epoch 00206: early stopping
[0.09016393442622951, 0.09016393442622951, 0.09016393442622953,
0.051738070452585716]
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
[0.08107626514611546, 0.08107626514611546, 0.08107626514611546,
0.04782415240770388]
[200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2', 200, '0.2']
```

```
Epoch 00194: early stopping
[0.06290092658588739, 0.06290092658588739, 0.06290092658588739,
0.02886584321690111]
[200, '0.3']
[0.2753029223093371, 0.2753029223093371, 0.2753029223093371, 0.2168872081718204]
[200, '0.3', 200, '0.3']
[0.231111903064861, 0.231111903064861, 0.231111903064861, 0.17615806450636173]
[200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00246: early stopping
[0.10727013542409124, 0.10727013542409124, 0.10727013542409124,
0.06523663598466534]
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
[0.061831789023521024, 0.061831789023521024, 0.061831789023521024,
0.03089862212446237]
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00245: early stopping
[0.043300071275837494, 0.043300071275837494, 0.043300071275837494,
0.011053024973720404]
[200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3', 200, '0.3']
Epoch 00280: early stopping
[0.040805416963649324, 0.040805416963649324, 0.040805416963649324,
0.011774770560165515]
[250, '0.1']
[0.26300784034212404, 0.26300784034212404, 0.26300784034212404,
0.20673535816629096]
[250, '0.1', 250, '0.1']
Epoch 00241: early stopping
[0.3558446186742694, 0.3558446186742694, 0.35584461867426936,
0.3066293157389701]
[250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00169: early stopping
[0.24358517462580184, 0.24358517462580184, 0.24358517462580184,
0.1961415507087051]
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00167: early stopping
[0.1443335709194583, 0.1443335709194583, 0.1443335709194583,
0.10284190805322913]
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00163: early stopping
[0.08267997148966501, 0.08267997148966501, 0.08267997148966501,
0.04849906766953527]
[250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1', 250, '0.1']
Epoch 00245: early stopping
[0.08856022808267996, 0.08856022808267996, 0.08856022808267996,
0.05350068877068037]
[250, '0.2']
[0.2966856735566643, 0.2966856735566643, 0.2966856735566643,
0.23811703510767068]
```

```
[250, '0.2', 250, '0.2']
[0.3670705630791162, 0.3670705630791162, 0.3670705630791161, 0.3162973225364475]
[250, '0.2', 250, '0.2', 250, '0.2']
[0.17979330007127584, 0.17979330007127584, 0.17979330007127584,
0.1300558613010846]
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00276: early stopping
[0.12081254454739843, 0.12081254454739843, 0.12081254454739844,
0.08267604380040405]
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
Epoch 00188: early stopping
[0.07323592302209551, 0.07323592302209551, 0.07323592302209551,
0.039733431057557]
[250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2', 250, '0.2']
[0.07377049180327869, 0.07377049180327869, 0.07377049180327869,
0.041056259095956116]
[250, '0.3']
[0.3132573057733428, 0.3132573057733428, 0.3132573057733428,
0.25290627435956714]
[250, '0.3', 250, '0.3']
Epoch 00284: early stopping
[0.2425160370634355, 0.2425160370634355, 0.2425160370634355,
0.19115487345844973]
[250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00270: early stopping
[0.10762651461154668, 0.10762651461154668, 0.10762651461154668,
0.0654504653495317]
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00211: early stopping
[0.0778688524590164, 0.0778688524590164, 0.0778688524590164,
0.04145904731879446]
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
Epoch 00232: early stopping
[0.05167498218104063, 0.05167498218104063, 0.05167498218104063,
0.019842027765843095]
[250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3', 250, '0.3']
[0.05434782608695652, 0.05434782608695652, 0.05434782608695652,
0.026805608899612032]
```

[30]:
```
import joblib
joblib.dump(results_dropout_smote, 'results_dropout_smote_one_hot')
```

[30]:
```
['results_dropout_smote_one_hot']
```

2.3 Análisis de los resultados

Una vez que hemos producido los experimentos, leemos los conjuntos de datos y creamos un dataset con todas las configuraciones y las métricas obtenidas. Para ello hacemos uso de la función *ReadAnd-*

Create

[23]:
```
import numpy as np
import pandas as pd
import joblib
```

[24]:
```
layer_config = []
Precision = []
Recall = []
F1 = []
Cohen_kappa = []
```

[25]:
```
def ReadAndCreate(file):
    results_1 = joblib.load(file)
    layer_config = []
    Precision = []
    Recall = []
    F1 = []
    Cohen_kappa = []
    for i in range(len(results_1)):
        aux = results_1[i]
        layer_config.append(aux['layer config'])
        Precision.append(aux['Metrics'][0])
        Recall.append(aux['Metrics'][1])
        F1.append(aux['Metrics'][2])
        Cohen_kappa.append(aux['Metrics'][3])

    data = {'Hidden layers' : layer_config,
            'Precision' : Precision,
           'Recall' : Recall,
            'F1' : F1,
           'Cohen kappa' : Cohen_kappa}
    data = pd.DataFrame(data)
    return data
```

2.3.1 Resultados con datos raw

[26]:
```
results_data = ReadAndCreate("results_1_joblib")
results_data.sort_values("Precision", ascending=False).head(6)
```

[26]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
28	[250, 250, 250, 250, 250]	0.825196	0.825196	0.825196	0.804531
29	[250, 250, 250, 250, 250, 250]	0.821810	0.821810	0.821810	0.801151
16	[150, 150, 150, 150, 150]	0.820919	0.820919	0.820919	0.799992
23	[200, 200, 200, 200, 200, 200]	0.819138	0.819138	0.819138	0.797986
17	[150, 150, 150, 150, 150, 150]	0.818425	0.818425	0.818425	0.797329
27	[250, 250, 250, 250]	0.816108	0.816108	0.816108	0.794788

[27]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[27]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
0	[50]	0.556308	0.556308	0.556308	0.501478
6	[100]	0.584105	0.584105	0.584105	0.532926
12	[150]	0.605132	0.605132	0.605132	0.557198
18	[200]	0.628653	0.628653	0.628653	0.584144
24	[250]	0.641661	0.641661	0.641661	0.599029
1	[50, 50]	0.677655	0.677655	0.677655	0.639487

Estudiando los datos con mayor puntuación en Precision descubrimos que los modelos con mayor número de neuronas y capas obtienen buenas puntuaciones, sin llegar al 83%. Inicialmente podríamos esperar que mayor número de neuronas está directamente relacionado con puntuación en metricas, pero vemos que no es así. Configuraciones de capas distintas con números de neuronas por encima de 150 dan lugar a resultados similares. Probablemente, la diferencia en puntuación se deba a la naturaleza aleatoria de las redes neuronales a la hora de inicializar los pesos así como en el algoritmo de minimización. Todos los modelos con características parecidas dan lugar a resultados muy similares.

Por otro lado, en los peores resultados, obtenemos lo esperado. Los modelos de una capa con pocas neuronas clasifican muy mal y mejoran su puntuación en orden creciente de neuronas.

2.3.2 Resultados datos raw one-hot

[28]:
```
results_data = ReadAndCreate("results_1_onehot_joblib")
results_data.sort_values("Precision", ascending=False).head(6)
```

[28]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
21	[200, 200, 200, 200]	0.830898	0.830898	0.830898	0.811140
26	[250, 250, 250]	0.830185	0.830185	0.830185	0.810440
25	[250, 250]	0.824127	0.824127	0.824127	0.803536
27	[250, 250, 250, 250]	0.817177	0.817177	0.817177	0.795549
22	[200, 200, 200, 200, 200]	0.814505	0.814505	0.814505	0.793240
20	[200, 200, 200]	0.809694	0.809694	0.809694	0.787236

[29]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[29]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
0	[50]	0.713293	0.713293	0.713293	0.679594
5	[50, 50, 50, 50, 50, 50]	0.724519	0.724519	0.724519	0.692311
3	[50, 50, 50, 50]	0.729330	0.729330	0.729330	0.697647
4	[50, 50, 50, 50, 50]	0.732359	0.732359	0.732359	0.701160
1	[50, 50]	0.735923	0.735923	0.735923	0.705121
6	[100]	0.740556	0.740556	0.740556	0.710106

Estudiamos los modelos con mayor puntuación. El uso de usar la codificación *one-hot* de las variables dan lugar a prácticamente los mismos resultados que sin usar esta transformación. Además, los modelos que consiguen las mayores puntuaciones también son muy parecidos.

Sin embargo, en los modelos que obtienen los peores resultados, éstos son muchos mejores que los obtenidos por los peores modelos sin usar *one-hot*. Las peores métricas obtenidas está acotadas inferiormente, en el caso de una sola capa oculta de 50 neuronas, en el 71%. Esto nos puede indicar que, mientras que usar *one-hot* es beneficioso en este tipo de modelos, sea difícil mejorar los resultados obtenidos en los mejores casos.

2.3.3 Resultados datos raw smote

[30]:
```
results_data = ReadAndCreate("results_smote_joblib")
results_data.sort_values("Precision", ascending=False).head(6)
```

[30]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
29	[250, 250, 250, 250, 250, 250]	0.361725	0.361725	0.361725	0.300894
23	[200, 200, 200, 200, 200, 200]	0.356557	0.356557	0.356557	0.296138
16	[150, 150, 150, 150, 150]	0.355132	0.355132	0.355132	0.295350
27	[250, 250, 250, 250]	0.354775	0.354775	0.354775	0.294756
17	[150, 150, 150, 150, 150, 150]	0.354419	0.354419	0.354419	0.293245
28	[250, 250, 250, 250, 250]	0.349786	0.349786	0.349786	0.288579

[31]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[31]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
0	[50]	0.273521	0.273521	0.273521	0.210563
6	[100]	0.299715	0.299715	0.299715	0.237226
12	[150]	0.300249	0.300249	0.300249	0.238289
1	[50, 50]	0.301675	0.301675	0.301675	0.238587
18	[200]	0.312723	0.312723	0.312723	0.250874
2	[50, 50, 50]	0.319672	0.319672	0.319672	0.257319

En un intento de mejorar el desbalanceo existente en las clases a clasificar, utilizamos el algoritmo SMOTE para mejorar el dataset. El motivo por el que usamos un algoritmo de sobremuestreo en vez de *tirar a la baja* con undersampling es por el beneficio que tienen las redes neuronales al aumentar el tamaño del dataset. Además, en results_datas realizadas con árboles de decisión en otras asignaturas, descubrimos que, a pesar de romper el problema de desbalanceo, obtenemos peores resultados al disponer de menos casos.

No ha mejorado la clasificación. Los resultados son muy malos. Las mejores redes obtienen métricas alrededor del 35%, mientras que las mejoras obtienen un 27%. Al haber tan poca diferencia entre los mejores y peores resultados, podemos estar seguros que no se debe a un número insuficiente de neuronas y capas.

En los problemas en los que las clases están muy desbalanceadas, SMOTE no funciona bien. Esta es una de esas ocasiones. Además, este algoritmo funciona mejor en problemas de clasificación binarios, donde fue concebido.

Como hemos mencionado antes, SMOTE está diseñado para problemas de predictores continuos, donde es más natural la interpolación. Sin embargo, como vimos que no daba ningún error al ejecutar la librería, vimos interesante ver el desempeño de la misma en problemas categóricos.

2.3.4 Resultados datos smote one-hot

[32]:
```
results_data = ReadAndCreate("results_smote_onehot_joblib")
results_data.sort_values("Precision", ascending=False).head(6)
```

[32]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
19	[200, 200]	0.187634	0.187634	0.187634	0.144296
26	[250, 250, 250]	0.177833	0.177833	0.177833	0.131770
20	[200, 200, 200]	0.152174	0.152174	0.152174	0.107797
25	[250, 250]	0.145937	0.145937	0.145937	0.103981
14	[150, 150, 150]	0.138097	0.138097	0.138097	0.091480
13	[150, 150]	0.129187	0.129187	0.129187	0.091482

[33]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[33]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
5	[50, 50, 50, 50, 50, 50]	0.054348	0.054348	0.054348	0.018471
17	[150, 150, 150, 150, 150, 150]	0.062545	0.062545	0.062545	0.026222
16	[150, 150, 150, 150, 150]	0.066999	0.066999	0.066999	0.031643
4	[50, 50, 50, 50, 50]	0.067890	0.067890	0.067890	0.028956
1	[50, 50]	0.069316	0.069316	0.069316	0.037007
2	[50, 50, 50]	0.069672	0.069672	0.069672	0.034412

La codificación one-hot ha empeorado los resultados con SMOTE. Las métricas están acotadas entre un 18% y un 5%.

2.3.5 Resultados datos raw con dropout

[34]:
```
results_data = ReadAndCreate("results_dropout")
results_data.sort_values("Precision", ascending=False).head(6)
```

[34]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
94	[250, 0.1, 250, 0.1, 250, 0.1, 250, 0.1, 250, …	0.874020	0.874020	0.874020	0.859249
76	[200, 0.1, 200, 0.1, 200, 0.1, 200, 0.1, 200, …	0.873664	0.873664	0.873664	0.858904
95	[250, 0.1, 250, 0.1, 250, 0.1, 250, 0.1, 250, …	0.871347	0.871347	0.871347	0.856290
74	[200, 0.1, 200, 0.1, 200, 0.1]	0.870991	0.870991	0.870991	0.855863
75	[200, 0.1, 200, 0.1, 200, 0.1, 200, 0.1]	0.866180	0.866180	0.866180	0.850478
77	[200, 0.1, 200, 0.1, 200, 0.1, 200, 0.1, 200, …	0.864754	0.864754	0.864754	0.848903

[35]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[35]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
30	[50, 0.3]	0.511048	0.511048	0.511048	0.449370
24	[50, 0.2]	0.517284	0.517284	0.517284	0.456681
35	[50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, …	0.522630	0.522630	0.522630	0.463069
34	[50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]	0.526372	0.526372	0.526372	0.466761
0	[50, 0.1]	0.527263	0.527263	0.527263	0.469118
18	[50, 0.1]	0.530649	0.530649	0.530649	0.472995

Los mejores resultados mejoran levemente los obtenidos sin dropout. Al igual que en estos últimos, las modelos más grandes obtienen mejores resultados. Notemos que todos los que aparecen con mejores métricas tienen un porcentaje de desactivación muy pequeño.

En el polo opuesto, los modelos que obtuvieron peor resultado fueron los modelos con el mayor porcentaje de desactivación en dropout: 30% y menor número de neuronas. Esto es lógico, ya que no parece ser un dataset lo suficientemente grande como para que se produzca un sobreaprendizaje durante el entrenamiento.

Recordamos que el número de épocas en el entrenamiento viene controlado por tensorflow mediante los *Callbacks*, como comentamos anteriormente, por lo que si no mejora la *accuracy* del conjunto de validación en 10 épocas, finaliza el entrenamiento.

2.3.6 Resultados datos raw one-hot con dropout

[36]:
```
results_data = ReadAndCreate("results_dropout_one_hot")
results_data.sort_values("Precision", ascending=False).head(6)
```

[36]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
77	[250, 0.1, 250, 0.1, 250, 0.1, 250, 0.1, 250, …	0.894512	0.894512	0.894512	0.882191
62	[200, 0.2, 200, 0.2, 200, 0.2]	0.892908	0.892908	0.892908	0.880426
79	[250, 0.2, 250, 0.2]	0.891839	0.891839	0.891839	0.879262
63	[200, 0.2, 200, 0.2, 200, 0.2, 200, 0.2]	0.888988	0.888988	0.888988	0.876026
56	[200, 0.1, 200, 0.1, 200, 0.1]	0.888453	0.888453	0.888453	0.875431
80	[250, 0.2, 250, 0.2, 250, 0.2]	0.887741	0.887741	0.887741	0.875431

```
80  0.887741     0.874633
```

[37]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[37]:
```
                                        Hidden layers  Precision    Recall  \
17  [50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, …   0.607627  0.607627
16      [50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]   0.640770  0.640770
15               [50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]   0.655560  0.655560
12                                          [50, 0.3]   0.658411  0.658411
14                       [50, 0.3, 50, 0.3, 50, 0.3]   0.669280  0.669280
6                                           [50, 0.2]   0.679793  0.679793

          F1  Cohen kappa
17  0.607627     0.561731
16  0.640770     0.597454
15  0.655560     0.614280
12  0.658411     0.617240
14  0.669280     0.629326
6   0.679793     0.641515
```

Los resultados son esperables vistos los casos anteriores. Dropout mejora también el caso en el que usamos *one-hot* encoding. Estos son los mejores resultados que obtenemos.

2.3.7 Resultados datos raw smote con dropout

[38]:
```
results_data = ReadAndCreate("results_dropout_smote")
results_data.sort_values("Precision", ascending=False).head(6)
```

[38]:
```
                                        Hidden layers  Precision    Recall  \
62                    [200, 0.2, 200, 0.2, 200, 0.2]   0.403243  0.403243
58  [200, 0.1, 200, 0.1, 200, 0.1, 200, 0.1, 200, …   0.400214  0.400214
77  [250, 0.1, 250, 0.1, 250, 0.1, 250, 0.1, 250, …   0.400036  0.400036
83  [250, 0.2, 250, 0.2, 250, 0.2, 250, 0.2, 250, …   0.399501  0.399501
80                    [250, 0.2, 250, 0.2, 250, 0.2]   0.399501  0.399501
74                    [250, 0.1, 250, 0.1, 250, 0.1]   0.398610  0.398610

          F1  Cohen kappa
62  0.403243     0.346788
58  0.400214     0.342223
77  0.400036     0.341473
83  0.399501     0.341194
80  0.399501     0.342195
74  0.398610     0.341175
```

[39]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[39]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
17	[50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, …	0.215966	0.215966	0.215966	0.151163
16	[50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]	0.227548	0.227548	0.227548	0.161465
15	[50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]	0.243763	0.243763	0.243763	0.179339
12	[50, 0.3]	0.245902	0.245902	0.245902	0.185292
6	[50, 0.2]	0.249465	0.249465	0.249465	0.187577
48	[150, 0.3]	0.254098	0.254098	0.254098	0.189198

Se obtienen mejores resultados que sin incluir dropout, pero siguen siendo malos.

2.3.8 Resultados datos smote one-hot con dropout

[40]:
```
results_data = ReadAndCreate("results_dropout_smote_one_hot")
results_data.sort_values("Precision", ascending=False).head(6)
```

[40]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
55	[200, 0.1, 200, 0.1]	0.370813	0.370813	0.370813	0.321053
79	[250, 0.2, 250, 0.2]	0.367071	0.367071	0.367071	0.316297
73	[250, 0.1, 250, 0.1]	0.355845	0.355845	0.355845	0.306629
37	[150, 0.1, 150, 0.1]	0.314505	0.314505	0.314505	0.261605
84	[250, 0.3]	0.313257	0.313257	0.313257	0.252906
78	[250, 0.2]	0.296686	0.296686	0.296686	0.238117

[41]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[41]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
16	[50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]	0.019601	0.019601	0.019601	0.003531
15	[50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]	0.023521	0.023521	0.023521	-0.000884
17	[50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, …	0.026907	0.026907	0.026907	0.009679
9	[50, 0.2, 50, 0.2, 50, 0.2, 50, 0.2]	0.031896	0.031896	0.031896	0.003915
10	[50, 0.2, 50, 0.2, 50, 0.2, 50, 0.2, 50, 0.2]	0.033856	0.033856	0.033856	0.009683
5	[50, 0.1, 50, 0.1, 50, 0.1, 50, 0.1, 50, 0.1, …	0.034569	0.034569	0.034569	

```
5   0.034569     0.008312
```

Se repite el comportamiento, usar dropout con un porcentaje de desactivación pequeño mejora los resultados, pero éstos eran muy malos per se.

2.4 Conclusiones

Hemos construido modelos de redes neuronales del tipo perceptrón multicapa con distintas configuraciones de capas ocultas y neuronas. Los datos utilizados han sido los originales, la codificación de las variables predictoras mediante *one-hot* y sus equivalentes tras usar *SMOTE*. También se ha implementado *callbacks* para parar el entrenamiento si el valor de *accuracy* en el set de validación no mejora tras 10 épocas, evitando el sobreaprendizaje y retirando el número de épocas como parámetro a estudiar y modificar. A todos estos modelos, se hicieron experimentos introduciendo capas intermedias con *dropout* con tres posibles valores de desactivación de neuronas: 10, 20 y 30%.

Los resultados son los siguientes. El uso de *SMOTE* no aporta ningún beneficio. Las métricas obtenidas son mucho peores que usando datos sin *SMOTE*. La razón puede deberse al gran desbalanceo entre clases que existe en la variable a predecir y la naturaleza categórica de las predictoras. El algoritmo fue diseñado y funciona mejor en problemas binarios con entradas continuas donde la interpolación parece más natural.

Introducir *dropout* mejora el resultado en todos los casos, incluso en los peores.

Creemos haber llegado a resultados próximos a los máximos posibles usando perceptrones multicapa ya que, en los casos con mejor puntuación, no existe una relación directa entre mayor número de neuronas y rendimiento. Se dan casos en los que configuraciones con menos capas obtienen mejores resultados que equivalentes con más. Sin embargo, está claro que un número grande neuronas es beneficioso para el modelo.

Todas las métricas se comportan de forma parecida: si un modelo obtiene mejor resultado que otro, todas sus métricas son mejores que las del otro.

3 Creación de árboles de clasificación con TensorFlowDecisionTrees

Con el fin de investigar más y aprendiendo sobre TensorFlow, descubrimos que, recientemente (Mayo de 2021), se ha incluido en la librería una API en la que se implementan de tres algoritmos muy famosos de árboles de decisión: Random Forest, Gradient Boosted Trees y CART.

Estos algoritmos, los cuales hemos visto en el máster, se ejecutan en C++ con la librería, también de Google, Yggdrasil Decision Forests. En realidad, al igual que la API en Python de TensorFlow, estamos utilizando un wrapper en el cual creamos los modelos fácilmente en Python que posteriormente serán entrenados en C++. De momento, esta librería no hace uso de la tarjeta gráfica.

La API se llama TensorFlow Decision Forest y puede ser instalada con `pip3 install tensorflow_decision_forests --upgrade`.

[ ]: