MANUAL TÉCNICO

Identificación del perfil demográfico de influencers en Twitter utilizando deep learning

Juan Carlos Alonso Sánchez [1], Aldo Isaac Hernández Antonio[1], Diana Martínez Frías [1], Juan Aguilera Huerta[1] , Bruno Adrián Franco Ortiz[1] , Raúl Uriel Silva Ramírez[1] , Juan Carlos Gómez Carranza[1]

[1]Departamento de Ingeniería Electrónica, División de Ingenierías Campus Irapuato-Salamanca, Universidad de Guanajuato. {jc.alonsosanchez, ai.hernandezantonio, d.martinezfrias@ugto.mx, j.aguilerahuerta@ugto.mx, ba.francoortiz@ugto.mx, ru.silvaramirez@ugto.mx, jc.gomezl@ugto.mx

Para llevar a cabo el perfilado demográfico de influencers en Twitter se elaboraron 4 códigos en Python, en los cuales implementamos la extracción de los tweets, la extracción de características textuales de los tweets, la construcción y entrenamiento de los modelos y la obtención de los resultados del desempeño de estos.

1. Primer Código: 01_split_documents_py

En este primer código utilizamos la biblioteca ***json*** ya que los conjuntos de datos de prueba y de entrenamiento que usamos están en formato ***ndjson***, este código leerá los conjuntos y extraerá su contenido en diferentes archivos. Se leerán dos ***ndjson***, uno contiene las publicaciones producidas por los usuarios (tweets) y otro las etiquetas de dicho usuario (género, ocupación y año de nacimiento). Para el conjunto de prueba se utilizaron los tweets de los seguidores y para el conjunto de entrenamiento los tweets de las celebridades.

Primero definimos las variables del entorno de trabajo y de los archivos de salida. Cabe mencionar que los conjuntos de datos de entrenamiento y de prueba se encuentran en distintas carpetas, de la misma manera los archivos de salida se guardarán en carpetas distintas.

```
import json

w_d = 'C:/Users/User/Documents/Celebrity_Profiling/Data/Dataset_test/'
#/Dataset_training/
f_f = w_d+'follower-feeds.ndjson'
#f_f = w_d+'celebrity-feeds.ndjson'
l_f = w_d+'labels.ndjson'
w_d = 'C:/Users/User/Documents/Celebrity_Profiling/Data/Text_Files_test/'
#/Text_Files_Training/
g_t = w_d + 'genders.txt'
o_t = w_d + 'occupations.txt'
a_t = w_d + 'ages.txt'
t_t = w_d + 'tweets.txt'
i_t = w_d + 'id.txt'
```

Abrimos ambos ***ndjson*** a la vez e iteramos por cada uno de sus elementos, extraemos los datos y los guardamos en cinco archivos de salida: id de usuario, tweet, ocupación, género y año de nacimiento.

```
with open(f_f, 'r', encoding = 'utf-8') as c_r,
     open(l_f, 'r', encoding = 'utf-8') as l_r:
    for c,l in zip(c_r,l_r):
        c_l = json.loads(c.strip())
        l_l = json.loads(l.strip())
        total += len(c_l['text'])
        with open(g_t, 'a', encoding = 'utf-8') as g,
             open(o_t, 'a', encoding = 'utf-8') as o,
             open(a_t, 'a', encoding = 'utf-8') as a,
             open(t_t, 'a', encoding = 'utf-8') as t,
             open(i_t, 'a', encoding = 'utf-8') as i:
             for follower in c_l['text']:
                for tweet in follower:
                    i.write(str(l_l['id']) + '\n')
                    g.write(l_l['gender'] + '\n')
                    o.write(l_l['occupation'] + '\n')
                    a.write(l_l['birthyear'] + '\n')
                    tweet = tweet.replace("\n"," ")
                    tweet = tweet.replace("\r"," ")
                    t.write(tweet.strip() + '\n')
```

Para el conjunto de prueba no necesitamos iterar por seguidor dentro del conjunto de datos, ya que los tweets son pertenecientes a una sola celebridad.

```
            for tweet in c_l['text']:
                  i.write(str(l_l['id']) + '\n')
                  g.write(l_l['gender'] + '\n')
                  o.write(l_l['occupation'] + '\n')
                  a.write(l_l['birthyear'] + '\n')
                  tweet = tweet.replace("\n"," ")
                  tweet = tweet.replace("\r"," ")
                  t.write(tweet.strip() + '\n')
```

2. Segundo Código: 02_split_tweets.py

En este código, leemos el archivo de texto que contiene los tweets de los usuarios y extraemos las características de texto en diferentes archivos de salida. Usamos las siguientes bibliotecas para cumplir la tarea.

```
import re #regular expresions
from nltk.tokenize.casual import EMOTICON_RE as emo_re
import emoji
```

Primero definimos las variables del directorio de trabajo y archivos de salida.

```
work_d ='C:/Users/User/Documents/Celebrity_Profiling/Data/Text_Files_test/'
#/Text_Files_training/'
text_f = work_d+'tweets.txt' #File with the whole tweets
abvs_c_f = work_d+'abvs.txt' #List of abreviations
words_f = work_d+'split/words.txt' #Create the directory /split/
emo_f = work_d+'split/emoticons.txt'
hash_f = work_d+'split/hashtags.txt'
at_f = work_d+'split/ats.txt'
link_f = work_d+'split/links.txt'
abv_f = work_d+'split/abvs.txt'
```

Implementamos una función sencilla que lee un archivo de texto que contiene las abreviaturas más usadas en Twitter.

```
abv_d = read_abvs(abvs_c_f)
```

Usamos la librería RE para definir las expresiones regulares de los links, hashtags y menciones que se encuentran en los tweets

```
url_re = re.compile(URLS, re.VERBOSE | re.I | re.UNICODE)
hashtag_re = re.compile('(?:^|\s)[##]{1}(\w+)', re.UNICODE)
mention_re = re.compile('(?:^|\s)[@@]{1}(\w+)', re.UNICODE)
```

Abrimos el archivo de texto que contiene los tweets e iteramos en cada línea, usando expresiones regulares para extraer las características textuales de cada tweet, luego las removemos dejando únicamente el texto. Estas características se guardan en seis diferentes archivos.

```
with open(text_f, 'r', encoding = 'utf-8') as text_r,
     open(words_f, 'w', encoding='utf-8') as words_w,
     open(emo_f, 'w', encoding='utf-8') as emo_w,
     open(hash_f, 'w', encoding='utf-8') as hash_w,
     open(at_f, 'w', encoding='utf-8') as at_w,
     open(link_f, 'w', encoding='utf-8') as link_w,
     open(abv_f, 'w', encoding = 'utf-8') as abv_w :
     for line in text_r:
        line = line.rstrip().lower()
        hashs = hashtag_re.findall(line)
        ats = mention_re.findall(line)
        links = url_re.findall(line)
        line = clean(line,hashs,ats,links)
        emoticons = emo_re.findall(line)
        emojis = [w for w in line if w in emoji.UNICODE_EMOJI_ENGLISH]
```

```
        words = re.findall('[a-záéíóúñàèìòù][a-záéíóúñàèìòù_-]+',line)
        abvs = [w for w in re.findall('[a-z0-9ñáéíóú+/]+'
                , line) if w in abv_d and len(w) > 1]
        words_w.write(' '.join(w for w in words)+'\n')
        abv_w.write(' '.join(w for w in abvs)+'\n')
        emo_w.write(' '.join(w for w in emoticons+emojis)+'\n')
        hash_w.write(' '.join(w for w in hashs)+'\n')
        at_w.write(' '.join(w for w in ats)+'\n')
        link_w.write(' '.join(w for w in links)+'\n')
```

3. Tercer Código: *03_build_corpus.py*

En este código leemos el archivo que contiene las palabras extraídas de los tweets para agruparlos por celebridad. Adicionalmente llevamos acabo un proceso de limpieza al texto, usando la biblioteca de Stopwords para eliminar las palabras vacías.

```
from nltk.corpus import spotwords
```

Además de las stopwords, eliminaremos también todas las palabras que tengan una longitud menor a 2 caracteres y mayor a 35 caracteres con la función definida a continuación.

```
def clean_words(words, stop_words):
    temp = [word for word in words if len(word)>2 and len(word)<35 and word
                                                    not in stop_words]
    if len(temp) > 3:
  text = ' '.join([word for word in words if len(word)>2 and len(word)
                                    <35 and word not in stop_words])
    else:
  text = ''
    return text
```

Primero definimos las variables del directorio de trabajo, los archivos que se leerán y los archivos de salida.

```
root = 'C:/Users/Administrador/Documents/code/2021_Celebrity_Profiling_JCAS/
Celebrity_Profiling/'
work_d = root + 'Data/Texts_Files_training_followers/'
users_file = work_d+'ids.txt'
tiempos = output_d+'tiempos.txt'
data_file = work_d+'split/words.txt'
output_file = work_d+'words_corpus.txt'
stop_words = stopwprds.words('english')
```

Creamos un diccionario y leemos simultáneamente el archivo de usuarios y el archivoque contiene las palabras de los tweets, y guardamos los contenidos pertenecientes a un usuario en el diccionario con su id como llave

```
print('Grouping per user...')
d_u = {}
i += 0
with open(users_file, 'r', encoding = 'utf-8') as u_r,
with open(data_file, 'r', encoding = 'utf-8') as d_r:
     for u_l, d_l in zip(u_r, d_r):
         u_l = u_l.strip()
         words = d_l.strip().split()
         text = clean_words(words, stop_words) + ' '
         #ignore empty tweets
         if text != ' ':
         #concatenate the text corresponding to each user to a dictionary
         d_u[u_l] = d_u.get(u_l, '') + text
         i += 1
         #print tweet count each 10,000 tweets
         if i%10000 == 0:
            print(i)
#save the contents of the dictionary to the output file
with open(output_file, 'w', encoding = 'utf-8') as o_w:
     for user in d_u:
   o_w.write(d_u[user]+'\n')
```

Finalmente guardamos el contenido del diccionario en el archivo de salida.

4. Cuarto código: *04_group_labels.py*

En este código agrupamos las etiquetas de los usuarios en un solo archivo por atributo. Primero definimos los nombres de las etiquetas para cada atributo (género, ocupación y edad); luego abrimos los archivos que contienen las etiquetas y los ids de usuario y creamos un diccionario usando el id como llave y la etiqueta como contenido.

```
main_dir = 'C:/Users/Administrador/Documents/code/2021_Celebrity_Profiling_J
CAS/Celebrity_Profiling/Data/'
probs = ['genders', 'ages', 'occupations']
u_f = main_dir + 'Text_Files_test/ids.txt'

for prob in probs:
    print(prob)
    if prob == 'genders':
        labels_names = ['male', 'female']
```

```
    elif prob == 'occupations':
        labels_names = ['sports','politics','performer','creator']
    else:
        labels_names = ['1940', '1941', '1942', '1943', '1944',
                        '1945', '1946', '1947', '1948', '1949',
                        '1950', '1951', '1952', '1953', '1954',
                        '1955', '1956', '1957', '1958', '1959',
                        '1960', '1961', '1962', '1963', '1964',
                        '1965', '1966', '1967', '1968', '1969',
                        '1970', '1971', '1972', '1973', '1974',
                        '1975', '1976', '1977', '1978', '1979',
                        '1980', '1981', '1982', '1983', '1984',
                        '1985', '1986', '1987', '1988', '1989',
                        '1990', '1991', '1992', '1993', '1994',
                        '1995', '1996', '1997', '1998', '1999']


    l_f = main_dir + 'Text_Files_test/' + prob + '.txt'
    d_l = {}


    with open(l_f, 'r', encoding = 'uft-8') as l_r, open(u_f, 'r',
encoding = 'uft-8') as u_r:
        for l, u in zip (l_r, u_r):
            l = l.strip()
            u = u.strip()
            d_l[u] = labels_names.index(l)

    us = list(d_l)
    us.sort()

    o_f = main_dir + 'Text_Files_test/' + prob + '_grouped.txt'
    with open(o_f, 'w', encoding = 'uft-8') as o_w:
        for u in us:
            l = d_l[u]
            o_w.write(str(l)+'\n')
```

Al finalizar guardamos las etiquetas agrupadas y los ids agrupados en archivos nuevos.

```
u_f_g = main_dir + 'Text_Files_test/ids_grouped.txt'
with open(u_f_g, 'w', encoding = 'uft-8') as u_g_r:
    for u in us:
        u_g_r.write(u+'\n')
```

5. Quinto código: *05_text_preprocessing.py*

Ahora que tenemos nuestro corpus de característica agrupado, tenemos que llevar a cabo un preprocesamiento de texto para convertir las secuencias de caracteres del corpus en secuencias numéricas, ya que la red neuronal funciona tomando entradas numéricas. Para esto usamos la librería Tokenizer del módulo preprocessing de Keras.

```
from keras.preprocessing import Tokenizer
```

Primero definimos el directorio de trabajo y los nombres de los archivos que contienen el corpus de característica de entrenamiento y el de prueba, y cargamos en la memoria el corpus de entrenamiento.

```
main_dir = 'C:/Users/Administrador/Documents/code/2021_Celebrity_Profiling_J
CAS/Celebrity_Profiling/Data/'
f_tr = main_dir + 'Text_Files_training_followers/words_corpus.txt'
f_ts = main_dir + 'Text_Files_test/words_corpus_test.txt'

print('Loading training data to memory...')
corpus_tr = read_corpus(f_tr)
print('Training data loaded!')
```

Luego instanciamos el tokenizador y hacemos que aprenda el vocabulario del corpus de entrenamiento. Posteriormente, usamos la función text_to_sequences, la cual nos devuelve el corpus, pero remplaza cada palabra por su identificador de vocabulario.

```
print('Tokenizing training data...')
tokenizer = Tokenizer(filters= '') # No filters ensures to use only a
                                   split function
tokenizer.fit_on_texts(corpus_tr)
print('Training data tokenized!')
print('Transforming training text to ints...')
data_train = tokenizer.texts_to_sequences(corpus_tr)
print('Training text transformed!')
```

Guardamos el corpus de entrenamiento convertido en secuencias de enteros en su propio archivo.

```
o_tr = main_dir +'Text_Files_training_followers/words_sequenced_corpus.txt'
print('Saving sequenced training data to file...')
with open(o_tr, 'w', encoding = 'uft-8') as o_w:
    for u in data_train:
        line = ''.join(str(i) for i in u)
        o_w.write(str(l)+'\n')
```

```
print('Sequenced training data saved!')
```

Guardamos el tamaño del vocabulario en un archivo ya que este dato será usado en el entrenamiento de la red neuronal

```
v_s = main_dir
+'Text_Files_training_followers/words_sequenced_corpus_voc_size.txt'
print('Saving vocabulary size to file...')
with open(v_s, 'w', encoding = 'uft-8') as v_w:
    v_w.write(str(len(tokenizer.word_index)))
print('Vocabulary size saved!')
```

Repetimos el proceso de conversión a secuencia de enteros para el corpus de prueba.

```
print('\n\n')
print('Loading test data to memory...')
corpus_ts = read_corpus(f_ts)
print('Test data loaded!')
print('Transforming test text to ints...')
data_test = tokenizer.texts_to_sequences(corpus_ts)
print('Test text transformed!')
```

Guardamos el corpus de prueba convertido en secuencias de enteros en su propio archivo.

```
o_ts = main_dir +'Text_Files_test/words_sequenced_corpus_test.txt'
print('Saving sequenced test data to file...')
with open(o_ts, 'w', encoding = 'uft-8') as o_w:
    for u in data_test:
        line = ''.join(str(i) for i in u)
        o_w.write(line+'\n')
print('Sequenced test data saved!')
```

6. Sexto código: *06_celebrity_rnn.py*

En este código se construyen, entrenan y evalúan las redes neuronales recurrentes usando un conjunto de datos de entrenamiento y uno de prueba. Usamos la librería Keras para implementar las redes neuronales y la librería Sklearn para evaluar las predicciones del modelo.

```
from keras.layers import Dense, Embedding, LSTM, GRU, Dropout, Bidirectional
from keras.preprocessing.sequence import pad_sequences
from keras.callbacks import EarlyStopping
from keras.metrics import AUC, Accuracy
from keras.models import Sequential
```

```
from sklearn import metrics
import tensorflow as tf
import pandas as pd
import numpy as np
import time
```

Primero definimos el directorio de trabajo, así como los nombres de los archivos que contienen las etiquetas y los corpus de características convertidos en secuencias de enteros.

Uno de los objetivos de la investigación es comparar el desempeño de las redes neuronales recurrentes variando dos de los parámetros que la componen, uno es el tamaño de dimensión de la capa de embedding, y otro es el número máximo de palabras que toma como entrada por cada usuario. En esta parte del código definimos cuales son los valores que se van a iterar para cada parámetro y cuál atributo demográfico vamos a predecir.

```
main_dir = 'C:/Users/Administrador/Documents/code/2021_Celebrity_Profiling_J
CAS/Celebrity_Profiling/Data/'
prob = 'genders'
f_tr = main_dir + 'Text_Files_training_followers/words_sequenced_corpus.txt'
f_voc_s = main_dir +
'Text_Files_training_followers/words_sequenced_corpus_voc_size.txt'
f_l_tr = main_dir + 'Text_Files_training_followers/' + prob + '_grouped.txt'
f_ts = main_dir + 'Text_Files_test/words_sequenced_corpus_test.txt'
f_l_ts = main_dir + 'Text_Files_test/' + prob + '_grouped.txt'

embedding_dimension_sets = [50, 100, 200, 300]
epochs = 20
max_len_sets = [7500, 8750, 10000, 11250, 12500]

classifiers = ['LSTM', 'GRU']
```

Primero cargamos tanto el corpus de entrenamiento como el de prueba en la memoria.

```
reading_s = time.time()
print('Loading sequenced training data to memory...')
data_train_original = read_sequenced_corpus(f_tr)
voc_size = read_vocabulary_size(f_voc_s)
print('Sequenced training data loaded!')

print('Loading sequenced test data to memory...')
data_test_original = read_sequenced_corpus(f_ts)
print('Test data loaded!')
```

Luego definimos los nombres de las etiquetas y damos un formato al archivo de salida dependiendo del atributo que se esté evaluando.

```
m_f = True
print('Loading training labels to memory...')
if prob == 'genders':
    labels_names = ['male', 'female']
    m_f = False
    fmt = '%d', '%1.9f', '%1.9f', '%d', '%d'
elif prob == 'occupations':
    labels_names = ['sports','politics','performer','creator']
    fmt = '%d', '%1.9f', '%1.9f', '%1.9f', '%1.9f', '%d', '%d'
else:
    labels_names = ['1940', '1941', '1942', '1943', '1944',
                    '1945', '1946', '1947', '1948', '1949',
                    '1950', '1951', '1952', '1953', '1954',
                    '1955', '1956', '1957', '1958', '1959',
                    '1960', '1961', '1962', '1963', '1964',
                    '1965', '1966', '1967', '1968', '1969',
                    '1970', '1971', '1972', '1973', '1974',
                    '1975', '1976', '1977', '1978', '1979',
                    '1980', '1981', '1982', '1983', '1984',
                    '1985', '1986', '1987', '1988', '1989',
                    '1990', '1991', '1992', '1993', '1994',
                    '1995', '1996', '1997', '1998', '1999']
    fmt = '%d', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f',
          '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f',
          '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f',
          '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f',
          '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f',
          '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f',
          '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f',
          '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f',
          '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%1.5f', '%d', '%d'
```

Cargamos las etiquetas de entrenamiento y de prueba en la memoria y las convertimos a un arreglo *numpy*.

```
l_ls_tr = read_labels_grouped(f_l_tr)
print('Training labels loaded!')
ls_tr = np.array(l_ls_tr)
print('Loading training labels to memory...')
l_ls_ts = read_labels_grouped(f_l_ts)
print('Test labels loaded!')
ls_ts = np.array(l_ls_ts)
print('Generating dummies for labels...')
```

```
# Convert the train lables into a array of n=classes dimenssion

ls_tr = pd.get_dummies(l_tr).values
print('Dummies generates!')
reading_f = time.time()
```

Una vez tenemos cargados en memoria tanto los corpus cómo las etiquetas de estos, podemos construir la red neuronal recurrente. Todos los fragmentos de códigos que siguen se encuentran dentro de estos ciclos anidados donde iteramos sobre las variables previamente definidas. Esto se hace 10 veces para poder sacar la mediana de los resultados.

```
for i in range(1,11):
    for cl in classifiers:
        for embedding_dimension in embedding_dimension_sets:
            for max_len in max_len_sets:
                print('Problem: ', prob)
                print('Embeddings: ', embedding_dimension)
                print('Epochs: ', epochs)
                print('Max Length: ', max_len)
                print('Classifier: ', cl)
```

Primero truncamos los datos con el tamaño definido por la variable *max_len* usando la función de Keras *pad_sequences()*.

```
pad_s = time.time()
print('Padding training data...')
data_train = pad_sequences(data_train_original, maxlen=max_len,
                           padding='post', truncating='post')
print('Training data padded!')
print('Padding test data...')
data_test = pad_sequences(data_train_original, maxlen=max_len,
                          padding='post', truncating='post')
print('Test data padded!')
pad_f = time.time()
```

Reiniciamos la sesión de keras para impedir que los datos de la sesión anterior interfieran con la iteración actual, y comenzamos a construir la red neuronal dependiendo del clasificador. Nos apoyamos del modelo *Sequential()* porque es usaremos una pila simple de capas donde cada capa tiene exactamente un tensor de entrada y un tensor de salida.

Aquí definimos qué clasificador deberá usar la red, la capa de *embedding*, la capa de *dropout* y la capa densa.

Compilamos la red y la entrenamos haciendo 20 pasadas al conjunto de datos de entrenamiento.

```
# Clear session
tf.keras.backend.clear_session()
print('Building NN with method:', cl)
train_s = time.time()
if cl == 'LSTM':
     # Create the neural network
     clf = Sequential()
     clf.add(Embedding(input_dim=voc_size+1,
                       output_dim=embedding_dimension,
                       input_length=max_len))
     clf.add(LSTM(128))
     clf.add(Dropout(rate=0.5))
     clf.add(Dense(units=len(labels_names), activation='softmax'))
     clf.compile(optimizer='adamax',loss='categorical_crossentropy',
                 metrics=[AUC(), Accuracy()])

     # Training
     print('\tTraining NN...')
     clf.fit(data_train, ls_tr, epochs=epochs, steps_per_epoch=20,
              batch_size=64, verbose=0,validation_split=0.1,
              callbacks= [EarlyStopping(monitor='val_auc', patience=5,
              mode='max', restore_best_weights=True)])
     print('\tNN trained!')
```

Se hace lo mismo en caso de estar evaluando con el clasificador GRU.

```
elif cl == 'GRU':
    # Create the neural network
    clf = Sequential()
    clf.add(Embedding(input_dim=voc_size+1,
                      output_dim=embedding_dimension,
                      input_length=max_len))
    clf.add(GRU(128))
    clf.add(Dropout(rate=0.5))
    clf.add(Dense(units=len(labels_names), activation='softmax'))
    clf.compile(optimizer='adamax', loss='categorical_crossentropy',
                metrics=[AUC(), Accuracy()])
    # Training
    print('\tTraining NN...')
    clf.fit(data_train, ls_tr, epochs=epochs,steps_per_epoch=20,
             batch_size=64, verbose=0, validation_split=0.1,
             callbacks= [EarlyStopping(monitor='val_auc', patience=5,
             mode='max', restore_best_weights=True)])
```

```
    print('\tNN trained!')
train_f = time.time()
```

Hacemos predicciones sobre el conjunto de datos de prueba y evaluamos contra las etiquetas reales para sacar las métricas de evaluación de accuracy, precisión, recall , F1, AUC y kapha.

```
pred_s = time.time()
print('Making predictions over the test data...')
predicted = np.argmax(clf.predict(data_test), axis=-1)
Predicted_proba = clf.predict(data_test)

print('Measuring results...')
accuary = np.mean(predicted == ls_ts)
precission_macro = metrics.precision_score(ls_ts, predicted,average='macro')
recall_macro = metrics.recall_score(ls_ts, predicted, average='macro')
f1_macro = metrics.f1_score(ls_ts, predicted, average='macro')
kapha = metrics.cohen_kappa_score(ls_ts, predicted)
if m_f:
  if len(set(ls_ts)) != predicted_proba.shape[1]:
     missing_columns = [x for j in range(len(labels_names)) if x not in
                        set(ls_ts)]
     missing_columns = np.array(missing_columns)
     predicted_proba = np.delete(predicted_proba,
                         missing_columns, 1)
     for x in range(predicted_proba.shape[0]):
        dif = 1-sum(predicted_proba[x])
        dif /= predicted_proba.shape[1]
        predicted_proba[x] += dif
   roc = metrics.roc_auc_score(ls_ts, predicted_proba, average='macro ',
                               multi_class='ovo')
else:
   roc = metrics.roc_auc_score(ls_ts, predicted, average='macro',
                               multi_class='ovo')
pred_f = time.time()
```

Por último, guardamos las métricas de evaluación y los tiempos de ejecución en un archivo de salida. Cuyo nombre indica qué clasificador y cuáles parámetros se usaron al construir la red neuronal.

7. Séptimo código: *07_results_calculator.py*

En este código, calculamos la mediana de la métrica F1 de las 10 ejecuciones para cada combinación posible de parámetros y clasificadores. Primero definimos el directorio de trabajo y las variables de iteración.

```
import numpy as np
main_dir = 'C:/Users/Administrador/Documents/code/2021_Celebrity_Profiling_JCAS/Celebrity_Profiling/Otuput//NN_followers'
prob = 'genders'
output_file = main_dir + prob+ '_results.txt'
# Neuronal network parameter sets
embedding_dimension_sets = [50, 100, 200, 300]
epochs = 20
max_len_sets = [7500, 8750, 10000, 11250, 12500]

classifiers = ['LSTM', 'GRU']
```

Iteramos sobre las variables para acceder a todos los archivos de salida que contienen los resultados. Al abrir el archivo extraemos únicamente la métrica F1 y guardamos la mediana, media y desviación estándar en un archivo de salida.

```
for cl in classifiers:
    for embedding_dimension in embedding_dimension_sets:
        for max_len in max_len_sets:
            f1 = []
            for i in range(1,11):
                results_file = main_dir +str(i)+'/'+prob+'_'
                results_file += cl+'_'+str(iembedding_dimension)
                results_file += '_'+str(epochs)+'_'+str(max_len)+ '.txt'
                with open(results_file, 'r', encoding = 'uft-8') as r_r:
                    for line in r_r:
                        if 'F1: ' in line:
                            line = line.strip()
                            line = line.replace('F1: ','')
                            line = float(line)
                            f1.append(line)
            with open(output_file, 'a', encoding = 'uft-8') as o_w:
                r_string = cl+'_'+str(iembedding_dimension)
                r_string += '_'+str(epochs)+'_'+str(max_len)+ '-\t'
                r_string += 'median: %.4f'%(np.median(f1)) + ' '
                r_string += 'mean: %.4f'%(np.mean(f1)) + ' '
                r_string += 'std: %.4f '%(np.std(f1)) + '\n'
                o_w.write(r_string)
```