MANUAL TÉCNICO

Perfilado demográfico de celebridades y seguimiento de usuarios en redes sociales

Juan Carlos Alonso Sánchez [1], Aldo Isaac Hernández Antonio [1], José Alfredo Romero González [1], Hugo Iván Lozoyo Belman [1], Luis Miguel López Santamaría [1], Juan Carlos Gómez Carranza [1]
[1]Departamento de Ingeniería Electrónica, División de Ingenierías Campus Irapuato-Salamanca, Universidad de Guanajuato.
{jc.alonsosanchez, ai.hernandezantonio, ja.romerogonzalez, hi.lozoyobelman, lm.lopezsantamaria, jc.gomez}@ugto.mx

A. Perfilado Demográfico de Celebridades de Redes Sociales

Para llevar a cabo el perfilado demográfico de celebridades de redes sociales se elaboraron 4 códigos en Python, en los cuales implementamos desde la adquisición de datos, extracción de características textuales a los tweets, la construcción y entrenamiento de los modelos hasta finalmente obtener los resultados de estos.

1. Primer Código: 00_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 el contenido producido 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 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'
```

Primero 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 01_split_tweets.py

En este código, leeremos el archivo de texto que contiene los tweets de los usuarios y extraeremos las características de texto en diferentes archivos de salida. Usaremos 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 simple función 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_wordvectors.py*

Para implementar las características de vectores de palabras usamos las siguientes bibliotecas. Usando distintos modelos pre entrenados generaremos archivos de texto convirtiendo el vocabulario de los usuarios en bancos de palabras, representando así a cada usuario como un vector de características densas de 200 o 300 dimensiones.

```
import fasttext.util
import gensim.downloader
from nltk.corpus import stopwords
import numpy as np
```

Primero declaramos las variables del directorio de trabajo, en este código iteraremos sobre las diferentes características de vectores de palabra *fasttext, glove* y *word2vec*.

```
root = 'C:/Users/User/Documents/Celebrity_Profiling/'
work_d = root + 'Data/Text_Files_test/'
users_file = work_d+'id.txt'
training_users = root + 'Data/Text_Files_training/users.txt'
labels_file = work_d+'words.txt'
t_labels_file = root + 'Data/Text_Files_training/words.txt'
feat_tag = ['fasttext', 'glove','word2vec']
```

Implementamos una función sencilla que lee los archivos que contienen los id de usuario.

```
print('Loading user list...')
users_list = read_users(users_file)
t_users_list = read_users(training_users)
```

Comenzamos a iterar cada característica, declaramos más variables del directorio de trabajo y si es necesario se descarga el modelo pre entrenado de las características. Se lee del archivo que contiene los tweets de tanto el conjunto de entrenamiento como el de prueba. En seguida implementamos la función `build_wordvector_files()` la cual lee el contenido de cada usuario y usando el modelo de vector de palabras calcula el promedio del valor de todas las palabras que se encuentran en sus tweets y las almacena en archivos de texto.

```
for feat in feat_tag:
    corpus_test_file = work_d+'Word_Vectors/'+feat
    corpus_train_file = root + 'Data/Text_Files_training/Word_Vectors/'+
    feat
```

```
    if feat == 'fasttext':
       data_file = work_d+'split/words.txt'
       training_file = root + 'Data/Text_Files_training/' +'split/words.txt
       fasttext.util.download_model('en', if_exists='ignore')
       ft = fasttext.load_model('cc.en.300.bin')
    elif feat == 'word2vec':
       data_file = work_d+'split/words.txt'
       training_file = root + 'Data/Text_Files_training/' +'split/words.txt
       gn = gensim.downloader.load('word2vec-google-news-300')
    elif feat == 'glove':
       data_file = work_d+'split/words.txt'
       training_file = root + 'Data/Text_Files_training/' +'split/words.txt
       gn = gensim.downloader.load('glove-twitter-200')
    corpus_test = read_text_data(data_file)
    print('Test corpus has been read')
    corpus_train = read_text_data(training_file)
    print('Training test has been read')
    print('Grouping tweet data per user...')
    corpus_test, user_list_inner = group_per_user(corpus_test, users_list)
    print('Test tweet data grouped!')
    corpus_train, t_user_list_inner = group_per_user(corpus_train, t_users_
list)
    print('Training tweet data grouped!')
    build_wordvector_files(corpus_test,corpus_test_file)
    print('Test corpus has been converted to files')
    build_wordvector_files(corpus_train, corpus_train_file)
    print('Training corpus has been converted to files')
```

4. Cuarto código: *03_celebrity_profiling_content.py*

En este código, creamos, entrenamos y evaluamos los distintos modelos de aprendizaje de máquina, todos los modelos los implementamos usando la biblioteca *sklearn*.

```
import time
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB as mnb
from sklearn import metrics
```

```
from nltk.corpus import stopwords
import numpy as np
from sklearn.preprocessing import normalize
```

Primero definimos las variables del directorio de trabajo, además definimos las listas de métodos, atributos y características que se evaluarán.

```
root = 'C:/Users/User/Documents/Celebrity_Profiling/'
work_d = root + 'Data/Text_Files_test/'
output_d = root +'Output/Test/'
wv_d_test = 'C:/Users/User/Documents/Celebrity_Profiling/Data/Text_Files_test/Word_Vectors/'
wv_d_train = 'C:/Users/User/Documents/Celebrity_Profiling/Data/Text_Files_training/Word_Vectors/'
users_file = work_d+'id.txt'
training_users = root + 'Data/Text_Files_training/users.txt'
method_tag = ['mnb','svm','knn','lr', 'rf','svc']
prob_tag = ['ages','occupations','ages']
feat_tag = ['words','emoticons','hashtags','ats','abvs','fasttext', 'glove', 'word2vec']
method = method_tag[5] #Elegir el metodo
```

Implementamos una función sencilla que lee los archivos que contienen los id de usuario.

```
print('Loading user list...')
users_list = read_users(users_file)
t_users_list = read_users(training_users)
print('User list loaded!')
```

En este código primero sobre las características y después sobre los atributos, esta decisión se tomó debido que el tiempo de lectura de los archivos de característica (tweets, hashtags, etc) era mucho mayor que el tiempo de lectura de los archivos de atributo (genero, ocupación, año de nacimiento). De tal manera pudimos entrenar y evaluar más modelos en menos tiempo. Para las características de vectores de palabras, en vez de leer el archivo de tweets, leemos los archivos generados previamente en el código *02_build_wordvectors.py*.

```
for feat in feat_tag:
    data_file = work_d+'split/'+feat+'.txt'
    training_file = root + 'Data/Text_Files_training/' +'split/'+feat+'.txt'
    print('Loading tweet data...')
    ### Desde este punto en adelante estaremos trabajando con contenido
        basado en características
    if feat == 'words':
        tiempo_lectura0 = time.time()
        corpus_test_feat = read_text_data(data_file)
        tiempo_lectura0 = time.time() - tiempo_lectura0
        tiempo_lectura1 = time.time()
```

```
        corpus_train_feat = read_text_data(training_file)
        tiempo_lectura1 = time.time() - tiempo_lectura1
    elif feat == 'fasttext' or feat == 'word2vec' or feat == 'glove':
        wv_file = wv_d_test + feat
        tiempo_lectura0 = time.time()
        corpus_test = read_word_vector_files(wv_file, 5)
        tiempo_lectura0 = time.time() - tiempo_lectura0
        wv_file = wv_d_train + feat
        tiempo_lectura1 = time.time()
        corpus_train = read_word_vector_files(wv_file, 20)
        tiempo_lectura1 = time.time() - tiempo_lectura1
    else: #cualquier otra característica textual
        tiempo_lectura0 = time.time()
        corpus_test_feat = read_extra_data(1,data_file)
        tiempo_lectura0 = time.time() - tiempo_lectura0
        tiempo_lectura1 = time.time()
        corpus_train_feat = read_extra_data(1,training_file)
        tiempo_lectura1 = time.time() - tiempo_lectura1
    print('Tweet data loaded!')
```

Una vez leímos los archivos de característica, construiremos los corpus usando una función que agrupa todo el contenido por usuario, creando así una lista de 400 usuarios para el conjunto de prueba y 1916 para el conjunto de entrenamiento.

```
    if feat == 'fasttext' or feat == 'word2vec' or feat == 'glove':
        tiempo_agrupamiento0 = 0
        tiempo_agrupamiento1 = 0
    else:
        tiempo_agrupamiento0 = time.time()
        corpus_test, user_list_inner = group_per_user(corpus_test_feat,
        users_list, 'feat')
        tiempo_agrupamiento0 = time.time() - tiempo_agrupamiento0
        print('Test feature data grouped!')
        tiempo_agrupamiento1 = time.time()
        corpus_train, t_user_list_inner = group_per_user(corpus_train_feat,
        t_users_list, 'feat')
        tiempo_agrupamiento1 = time.time() - tiempo_agrupamiento1
        print('Training feature data grouped!')
```

Ahora iteraremos sobre nuestra lista de atributos, declaramos una lista con los nombres de las etiquetas posibles para cada atributo

```
    for prob in prob_tag:
        if prob == 'genders':
            multiclass = 0 # 0 = no
            labels_names = ['male', 'female']
```

```
    elif prob == 'occupations':
        multiclass = 1 # 1 = yes
        labels_names = ['sports','politics','performer','creator']
    else:
        multiclass = 1 # 1 = yes
        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']
```

Leemos los archivos que contienen las etiquetas de atributos usando una función.

```
    labels_file = work_d+prob+'.txt'
    t_labels_file = root + 'Data/Text_Files_training/' +prob+'.txt'
    print('Loading users labels...')
    labels_list = read_labels(labels_file, labels_names)
    t_labels_list = read_labels(t_labels_file,labels_names)
    print('Users labels loaded!')
```

Agrupamos las etiquetas por usuario usando la misma función de agrupación previamente usada. Dando como resultado una lista con 400 etiquetas de usuario para el conjunto de prueba y una lista con 1916 etiquetas de usuario para el conjunto de entrenamiento.

```
    print('Grouping label data per user...')
    tiempo_agrupamiento_l0 = time.time()
    labels_list_inner, user_list_inner = group_per_user(labels_list,
    users_list, 'label')
    tiempo_agrupamiento_l0 = time.time() - tiempo_agrupamiento_l0
    print('Test label data grouped!')
    tiempo_agrupamiento_l1 = time.time()
    t_labels_list_inner,t_user_list_inner =group_per_user(t_labels_list,
    t_users_list, 'label')
    tiempo_agrupamiento_l1 = time.time() - tiempo_agrupamiento_l1
    print('Training label data grouped!')
```

Para crear y entrenar los modelos tenemos que transformar las listas de etiquetas en arreglos matemáticos usando la biblioteca *numpy*.

```
        labels_test = np.asarray(labels_list_inner)
        labels_train = np.array(t_labels_list_inner)
```

Ahora comenzaremos a construir el modelo, comenzamos declarando el archivo de salida en el cual se imprimirán las etiquetas predichas y las probabilidades calculadas.

```
        print('Building model')
        f_name = output_d +method+'_'+prob+'_'+feat+'.txt'
```

realizamos la vectorización *tfidf* para ambos corpus de prueba y de entrenamiento, cabe mencionar que para las características de vectores de palabras no se necesita hacer este paso debido a que los usuarios ya tienen una representación matricial derivada de los modelos de vectores de palabras.

```
        if feat == 'fasttext' or feat == 'word2vec' or feat == 'glove':
            train_tfidf = corpus_train
            test_tfidf = corpus_test
        else:
            data_train = corpus_train
            data_test = corpus_test
            vec = TfidfVectorizer(min_df=1, norm='l2', analyzer = 'word',
            tokenizer=my_tokenizer)
            train_tfidf = vec.fit_transform(data_train)
            test_tfidf = vec.transform(data_test)
```

Llevamos a cabo la validación del modelo, en la cual se hace una validación cruzada estratificada de 5 partes para encontrar el mejor hiperparámetro del modelo seleccionado. Se entrena el modelo usando el corpus de entrenamiento y sus etiquetas. Y usando el corpus de prueba se predicen las etiquetas de este en adición de las probabilidades de dichas etiquetas.

```
        tiempo0 = time.time()  #validación
        clf = classifier(method,train_tfidf, labels_train)
        tiempo0 = time.time() - tiempo0
        tiempo1 = time.time()  #entrenamiento
        clf.fit(train_tfidf, labels_train)
        tiempo1 = time.time() - tiempo1
        tiempo2 = time.time()  #predicción
        predicted = clf.predict(test_tfidf)
        tiempo2 = time.time() - tiempo2
        predicted_proba = clf.predict_proba(test_tfidf)
```

Guardamos las salidas del modelo en un archivo de texto.

```
        with open(f_name,'w',encoding='utf-8') as f_w:
            for p,p_b,l_t,t_i in zip(predicted, predicted_proba,labels_test,
```

```
            user_list_inner):
                s_b = str(t_i)+' '+str(l_t)+' '+str(p)+' '+
                      ' '.join(str(x) for x in p_b)+'\n'
                f_w.write(s_b)
```

Usando la biblioteca *sklearn.metrics* evaluamos seis métricas por cada modelo (*accuracy*, *precisión*, *recall*, *f1*, *kappa* y *auc*).

```
        accuracy = metrics.accuracy_score(labels_test, predicted)
        precision = metrics.precision_score(labels_test, predicted, average=
                    'macro')
        recall=metrics.recall_score(labels_test, predicted, average='macro')
        f1_macro = metrics.f1_score(labels_test, predicted, average='macro')
        kappa = metrics.cohen_kappa_score(labels_test, predicted)
        if multiclass:
            if len(set(labels_test)) != predicted_proba.shape[1]: # for ages
                missing_columns = [x for x in range(len(labels_names))
                                   if x not in set(labels_test)]
                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
            auc = metrics.roc_auc_score(labels_test, predicted_proba,
                  multi_class='ovo')
        else:
            auc = metrics.roc_auc_score(labels_test, predicted,
                  multi_class='ovo')
```

Finalmente guardamos estas métricas y los tiempos de ejecución del modelo en un archivo de resultados.

```
        resultados = output_d + method+'_results.txt'
        print('Models finished!')
        with open(resultados, 'a',encoding='utf-8') as r_w:
            r_w.write('Feature: '+ feat+'\n')
            r_w.write(prob[0].upper()+prob[1:]+' Accuracy: %0.2f' %
                      (accuracy)+'\n')
            r_w.write(prob[0].upper()+prob[1:]+'F1: %0.2f' %(f1_macro)+'\n')
            r_w.write(prob[0].upper()+prob[1:]+'AUC: %0.2f' %(auc)+'\n')
            r_w.write(prob[0].upper()+prob[1:]+'Kappa: %0.2f' %(kappa)+'\n')
```

```
            r_w.write('Tiempo de lectura test: '+ str(tiempo_lectura0)+'\n')
            r_w.write('Tiempo de lectura train: '+str(tiempo_lectura1)+'\n')
            r_w.write('Tiempo de agrupamiento feat test: '+
                      str(tiempo_agrupamiento0)+'\n')
            r_w.write('Tiempo de agrupamiento feat train: '+
                      str(tiempo_agrupamiento1)+'\n')
            r_w.write('Tiempo de agrupamiento label test: '+
                      str(tiempo_agrupamiento_l0)+'\n')
            r_w.write('Tiempo de agrupamiento label train: '+
                      str(tiempo_agrupamiento_l1)+'\n')
            r_w.write('Tiempo de validación: '+ str(tiempo0)+'\n')
            r_w.write('Tiempo de entrenamiento: '+ str(tiempo1)+'\n')
            r_w.write('Tiempo de predicción: '+ str(tiempo2)+'\n')
```

B. Identificación de Usuarios entre Redes Sociales

1. Código “02_users_track_classification_cross_network.py”

Este script fue codificado usando el lenguaje de programación *Python*, tiene como propósito principal recopilar y procesar los datos previamente obtenidos de las redes sociales, limpiarlos, construir y entrenar los modelos, realizar experimentos y finalmente obtener los resultados de estos.

Para logar esto, el script cuenta con las siguientes funciones.

1.1. my_tokenizer(s)

```
def my_tokenizer(s):
    return s.split()
```

Obtiene una lista de todas las palabras contenidas en el string “s” separadas por un carácter de espacio.

1.2. read_users(file)

Obtiene una lista de los id de los usuarios que se encuentran en el archivo “file”.

```
def read_users(file):
        users = []
        with open(file) as content_file:
                for line in content_file:
                users.append(int(line.strip()))
        return users
```

1.3. read_labels(file, labels_names)

Obtiene una lista de números los cuales representarán a los labels ingresados, por ejemplo [0,1] para [‘H’,’M’].

```
def read_labels(file, labels_names):
    label = []
    with open(file) as content_file:
        for line in content_file:
            # print(line.strip())
            category = line.strip().upper()
            label.append(labels_names.index(category))
    return label
```

1.4. clean_words(words, stop_words)

Obtiene un string con todas las palabras que se encuentren en la lista “words” con al menos de tres caracteres, menos de 35 caracteres y que no se encuentren en el lista “stop_words” separadas por un carácter de espacio.

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

1.5. read_text_data(lang, file)

Obtiene una lista de strings con las palabras del texto contenido en el archivo “file” limpiándolo con la función *clean_words*.

```
def read_text_data(lang, file):
    data = []
    if lang == 'spanish':
        stop_words = stopwords.words('spanish')
    elif lang == 'english':
        stop_words = stopwords.words('english')
    with open(file, encoding='utf-8') as content_file:
        for line in content_file:
            words = line.rstrip().split()
            text = clean_words(words, stop_words)
            data.append(text)
    return data
```

1.6. group_per_user(corpus, labels, users)

Obtiene una tupla de longitud 2, el primer elemento contiene una lista de los datos agrupados para cada usuario, el segundo elemento contiene una lista de etiquetas para cada usuario.

```
def group_per_user(corpus, labels, users):
    corpus_grouped = []
    labels_grouped = []
    d_text = {}
    d_label = {}
    for user,label,text in zip(users, labels, corpus):
        d_text[user] = d_text.get(user,'')+text+' '
        d_label[user] = label
    for user in d_text:
        corpus_grouped.append(d_text[user])
        labels_grouped.append(d_label[user])
    return corpus_grouped, labels_grouped
```

1.7. clean_en_words(corpus)

Obtiene una lista de strings, el cual, corresponde a las palabras obtenidas de “corpus” sin stopwords del idioma inglés.

```
def clean_en_words(corpus):
    data = []
    stop_words = stopwords.words('english')
    for line in corpus:
         words = line.rstrip().split()
         text = ' '.join([word for word in words if len(word)>2 and len (word)<35 and word not in stop_words])
         data.append(text)
    return data
```

1.8. accuracy_top_n(labels_test, predicted_proba, n)

Obtiene el accuracy del top-n de las muestras.

```
def accuracy_top_n(labels_test, predicted_proba, n):
    accuracy = 0
    for proba_l, l_test in zip(predicted_proba, labels_test):
        prob_t = [(prob, idx) for idx, prob in enumerate(proba_l)]
        prob_t.sort(reverse=True)
        idxs = [idx for prob, idx in prob_t[:n]]
        if l_test in idxs:
            accuracy += 1
    accuracy /= len(labels_test)
    return accuracy
```

1.9. training_test(clf, train_tfidf, labels_train, corpus_test, labels_test, labels_list_test, out_dir, start, classifier, C=-1)

Realiza el entrenamiento con el clasificador “clf” usando los datos de entrenamiento, obtiene una predicción y verifica con los datos de prueba.

En la salida estándar se imprime el nombre del clasificador usado y los resultados parametrizados obtenidos, la cual incluye, accuracy, precission, recall, f1 macro, accuracy top 2, accuracy top 5, accuracy top 10 y el tiempo de entrenamiento + prueba tomando como inicio el tiempo “start”.

En el archivo de salida encontrado en “out_dir” se obtiene los resultados de las pruebas, incluye, la probabilidad de la predicción, la predicción y la clase real de cada una de las pruebas realizadas.

Contenido del script

En este fragmento del código se importan las librerías necesarias para la ejecución del script y se establecen los parámetros del experimento (idioma, red de entrenamiento, red de prueba, ubicaciones de los archivos de entrada y salida, etc.)

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from nltk.corpus import stopwords
from sklearn import metrics
from sklearn import svm
import numpy as np
import time

# Files for train data
train_dir = train_network+'/corpus/'+lang+'/'
labels_file_train = train_dir+'labels.txt'
users_file_train = train_dir+'users.txt'
words_file_train = train_dir+'split/words.txt'

# Files for test data
test_dir = test_network+'/corpus/'+lang+'/'
labels_file_test = test_dir+'labels.txt'
users_file_test = test_dir+'users.txt'
words_file_test = test_dir+'split/words.txt'

labels_names_file = train_dir + 'all_users.txt'
labels_names = []
with open(labels_names_file, encoding='utf-8') as content_file:
    for line in content_file:
        labels_names.append(line.strip().upper())
```

Posteriormente se recopila y procesa la información de entrenamiento y prueba según los parámetros establecidos

```
# Reading train data
labels_list_train = read_labels(labels_file_train, labels_names)
users_list_train = read_users(users_file_train)
corpus_train = []
corpus_train = read_text_data(lang, words_file_train)
# Clean possible English stopwords that could be in the Spanish train corpus
if lang == 'spanish':
    corpus_train = clean_en_words(corpus_train)

corpus_train, labels_list_train = group_per_user(corpus_train, labels_list_train, users_list_train)
labels_train = np.asarray(labels_list_train)

# Reading test data
labels_list_test = read_labels(labels_file_test, labels_names)
users_list_test = read_users(users_file_test)
corpus_test = []
corpus_test = read_text_data(lang, words_file_test)
# Clean possible English stopwords that could be in the Spanish train corpus
if lang == 'spanish':
    corpus_test = clean_en_words(corpus_test)
corpus_test, labels_list_test = group_per_user(corpus_test, labels_list_test, users_list_test)
labels_test = np.asarray(labels_list_test)
```

Y por último se construyen los diferentes modelos, se entrenan, prueban y se reportan los resultados tanto en la salida estándar como en el archivo de salida.

```
# TF-IDF
vec = TfidfVectorizer(min_df=1, norm='l2', analyzer = 'word', tokenizer=my_tokenizer)
train_tfidf = vec.fit_transform(corpus_train)

# Models creation
clf_nb = MultinomialNB()
clf_svm = svm.SVC(C=10, kernel='linear') # cs
clf_log = LogisticRegression(C=100, penalty='l2', solver='liblinear') # cs
clf_knn = KNeighborsClassifier(n_neighbors=5) #ks
clf_rdf = RandomForestClassifier()
clf_sgdc = SGDClassifier(loss='log', alpha=1/10, max_iter=10000) # cs

# Training and testing with the best hyper-paramater
l_classsifier = ['KNN', 'LR', 'MNB', 'RF', 'SGD', 'SVM']
# l_classsifier = ['MNB']
# l_classsifier = ['RF']
for classifier in l_classsifier:
    if classifier == 'KNN':
        l_params_knn = [1, 2, 3, 5, 10]
        for C in l_params_knn:
            clf = KNeighborsClassifier(n_neighbors=C)
            out_dir =  lang +'/probabilities_v2_train_'+train_network+'_test_'+test_network+'/'+classifier+'/'+str(C)+'/'+feature+'/'

            start = time.time()
            training_test(clf, train_tfidf, labels_train, corpus_test, labels_test, labels_list_test, out_dir, start, classifier, C)
    elif classifier == 'LR':
        l_params_lr_sgd_svm = [0.01, 0.1, 1, 10, 100]
        for C in l_params_lr_sgd_svm:
            clf = LogisticRegression(C=C, penalty='l2', solver='liblinear')
            out_dir =  lang +'/probabilities_v2_train_'+train_network+'_test_'+test_network+'/'+classifier+'/'+str(C)+'/'+feature+'/'
            start = time.time()
            training_test(clf, train_tfidf, labels_train, corpus_test, labels_test, labels_list_test, out_dir, start, classifier, C)
    elif classifier == 'SGD':
        l_params_lr_sgd_svm = [0.01, 0.1, 1, 10, 100]
        for C in l_params_lr_sgd_svm:
            clf = SGDClassifier(loss='log', alpha=1/C, max_iter=10000)
            out_dir =  lang +'/probabilities_v2_train_'+train_network+'_test_'+test_network+'/'+classifier+'/'+str(C)+'/'+feature+'/'
            start = time.time()
            training_test(clf, train_tfidf, labels_train, corpus_test, labels_test, labels_list_test, out_dir, start, classifier, C)
    elif classifier == 'SVM':
        l_params_lr_sgd_svm = [0.01, 0.1, 1, 10, 100]
        for C in l_params_lr_sgd_svm:
            clf = svm.SVC(C=C, kernel='linear', probability=True)
            out_dir =  lang +'/probabilities_v2_train_'+train_network+'_test_'+test_network+'/'+classifier+'/'+str(C)+'/'+feature+'/'
            start = time.time()
            training_test(clf,train_tfidf, labels_train, corpus_test, labels_test, labels_list_test, out_dir, start, classifier, C)
    elif classifier == 'RF':
        l_params_rf = [10, 50, 100, 200, 500]
        # 10, 50, 100, 200, 500
        for C in l_params_rf:
            clf = RandomForestClassifier(n_estimators=C, random_state=0)
```

```
        out_dir =  lang +'/probabilities_v2_train_'+train_network+'_test_'+test_network+'/'+classifier+'/'+str(C)+'/'+feature+'/'

        start = time.time()
        training_test(clf, train_tfidf, labels_train, corpus_test, labels_test, labels_list_test, out_dir, start, classifier, C)
elif classifier == 'MNB':
        clf = MultinomialNB()
        out_dir =  lang +'/probabilities_v2_train_'+train_network+'_test_'+test_network+'/'+classifier+'/'+feature+'/'
        start = time.time()
        training_test(clf, train_tfidf, labels_train, corpus_test, labels_test, labels_list_test, out_dir, start, classifier)
```