Manual Técnico

Detección de noticias falsas (fake news) en internet utilizando deep learning

Mariana Esmeralda Centeno Reyes[1], Cristopher Jesús Chaire Rodríguez[1], Mario Isaac Fonseca Martínez[1], Diana Martínez Frías[1], Fernando Oviedo Paramo[1], Juan Carlos Gómez Carranza[1].
[1]Departamento de Ingeniería Electrónica, División de Ingenierías Campus Irapuato-Salamanca, Universidad de Guanajuato
{me.centenoreyes, cj.chairerodriguez, mi.fonsecamartinez, d.martinezfrias, f.oviedoparamo, jc.gomez}@ugto.mx[1]

Para llevar a cabo la clasificación entre noticias falsas y reales, los diferentes códigos hechos se construyeron con base en dos modelos:

Modelo de regresión logística, el cual está formado por procesamiento, transformación y su construcción.

Modelo transformador, el cual está formado solo por procesamiento y su construcción.

• Primer modelo: processing/01_get_clean_words.py

Su objetivo principal es obtener las palabras de las diferentes publicaciones informativas que hay en internet. Las librerías utilizadas son:

```
from nltk.corpus import stopwords
import pandas as pd
import re
```

Las funciones que contiene:

clean_post se encarga de quitar hipervínculos en las publicaciones

```
def clean_post(post, urls):
    for url in urls:
        post = post.replace(url,'')
    return post
```

clean_words se encarga de filtrar las publicaciones, es decir, guarda una lista que nosotros creemos son palabras, dejando de lado números, palabras sin espacio y conectores

```
def clean_words(words, sw):
    words = [w for w in words if w not in sw
                              and len(w) > 2
                              and len(w) < 25
                              and not w.isdigit()]
    return words
```

El programa cuenta con un código principal en el cual se hace la limpieza de cada una de las publicaciones y donde al final se guardan en un archivo .csv para poder trabajar posteriormente con él.

La siguiente parte nos muestra una expresión regular para detectar URLS dentro de un texto

```
# Pattern to match url links in a text
URLS = r"""     # Capture 1: entire matched URL
  (?:
  https?:       # URL protocol and colon
    (?:
      /{1,3}        # 1-3 slashes
      |         #   or
      [a-z0-9%]       # Single letter or digit or '%'
                                    # (Trying not to match e.g.
"URI::Escape")
    )
    |         #   or
                                    # looks like domain name followed
by a slash:
    [a-z0-9.\-]+[.]
    (?:[a-z]{2,13})
    /
  )
  (?:         # One or more:
    [^\s()<>{}\[\]]+      # Run of non-space, non-()<>{}[]
    |         #   or
    \([^\s()]*?\([^\s()]+\)[^\s()]*?\) # balanced parens, one level deep:
(...(...)...)
    |
    \([^\s]+?\)       # balanced parens, non-recursive: (...)
  )+
  (?:         # End with:
    \([^\s()]*?\([^\s()]+\)[^\s()]*?\) # balanced parens, one level deep:
(...(...)...)
    |
    \([^\s]+?\)       # balanced parens, non-recursive: (...)
    |         #   or
    [^\s`!()\[\]{};:'".,<>?«»“”‘’]  # not a space or one of these punct
chars
  )
  |         # OR, the following to match naked domains:
  (?:
    (?<!@)              # not preceded by a @, avoid matching
foo@_gmail.com_
    [a-z0-9]+
    (?:[.\-][a-z0-9]+)*
    [.]
    (?:[a-z]{2,13})
    \b
```

```
    /?
    (?!@)             # not succeeded by a @,
                      # avoid matching "foo.na" in
"foo.na@example.com"
  )
"""
```

En esta parte definimos los archivos de entrada y de salida, *f_name* es una lista la cual contiene las rutas de los archivos para prueba, para entrenamiento y para validación; *f_clean* contiene las rutas de esos mismos archivos, pero para después de haber sido limpiados

```
# Path to file
f_name = ['./data/constraint_test_with_labels.csv',
'./data/constraint_train.csv', './data/constraint_val.csv']  # name of
input file
f_clean = ['./clean_data/test_clean.csv', './clean_data/train_clean.csv',
'./clean_data/val_clean.csv'] # name of output file
```

Aquí se almacena en la variable *sw* el conjunto de palabras más comunes en inglés, las cuales nos ayudan a filtrar o eliminar palabras que son innecesarias para procesar texto.

```
# Read the stopwords in English
sw = stopwords.words('english')
sw = set(sw)
```

En la siguiente parte se crea un objeto de expresión regular con la función *re.compile* a partir del patrón que se generó con *URLS*, de esta manera podemos buscar y detectar URLs las veces que sean necesarias

```
# Compile the pattern for matching
url_re = re.compile(URLS, re.VERBOSE | re.I | re.UNICODE)
```

Por último, tenemos la limpieza y el procesamiento de los datos .CSV

```
# fi: file in; fo: file out
for fi, fo in zip(f_name, f_clean):
    # Read the input file as csv to create a DataFrame
    df = pd.read_csv(fi, encoding='utf-8')

    # List to append the clean posts with their labels
    l_clean = []

    for post in df.itertuples():
        label = post[3]
        post = post[2].lower()
        urls = url_re.findall(post)
        post = clean_post(post, urls)
        words = re.findall('[\w_-]+', post)
```

```
        words = clean_words(words, sw)
        post_clean = ' '.join(words)
        l_clean.append({'post': post_clean, 'label': label})

    # New DataFrame with the clean data
    df_clean = pd.DataFrame(l_clean)

    # Write the new DF to a CSV file (without the index)
    df_clean.to_csv(fo, index=False)
```

- **Primer modelo: transformation/02_get_word_vector.py**

Este código es el encargado de crear la vectorización numérica tanto de las palabras de cada una de las publicaciones, así como el vector de cada una de estas. Para ello las librerías usadas son las siguientes, así como el modelo de *fasttext*.

```
import pandas as pd
import numpy as np
import fasttext
import fasttext.util
```

Primeramente, se descarga el modelo, en este caso se descarga en el idioma inglés, posteriormente se carga, dando como requisito que las dimensiones sean 300 (el máximo según la propia página de FastText)

```
fasttext.util.download_model('en', if_exists='ignore')
ft = fasttext.load_model('cc.en.300.bin')
```

La siguiente parte almacena en *f_clean* una lista con las rutas de los archivos de prueba limpios, los de entrenamiento limpios y los de validación también limpios. En *f_embeddings* las rutas de los archivos que almacenaran las representaciones vectoriales

```
f_clean = ['./clean_data/test_clean.csv', './clean_data/train_clean.csv',
'./clean_data/val_clean.csv']
f_embeddings = ['./embeddings/test_embeddings.csv',
'./embeddings/train_embeddings.csv', './embeddings/val_embeddings.csv']
```

Por último, tenemos la creación del vector que representa a cada una de las publicaciones, este vector esta dado por la suma de cada palabra vectorizada divido entre el numero de palabras que contiene cada publicación, como dato importante dicho vector debe tener la misma dimensión que cada una de las palabras en este caso 300. Al final es guardado en un archivo .CSV junto con su clasificación correspondiente.

```
test = []

for fi, fo in zip(f_clean, f_embeddings):
    df = pd.read_csv(fi, encoding='utf-8')
    posts = df["post"].tolist()
    labels = df["label"].tolist()
```

```
    posts = [post.split() for post in posts]

    vectors = []

    for post, label in zip(posts, labels):
        sum_vector = np.array( np.sum([ft.get_word_vector(word) for word
in post], axis=0) )
        sum_vector = sum_vector/len(post)
        mean_vector = ' '.join(str(x) for x in sum_vector)
        vectors.append({'post': mean_vector, 'label': '0' if label ==
'fake' else '1'})

    df_vectors = pd.DataFrame(vectors)
    df_vectors.to_csv(fo, index=False)
```

- **Primer modelo: transformation/03_standardization_and_normalization.py**

Este código es el encargado de continuar la transformación del primer modelo, estandarizando y normalizando los vectores dados por el código *02_get_word_vector.py*. Para este fin, se usaron las librerías y el modelo *LogisticRegression*:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import multilabel_confusion_matrix
from sklearn.metrics import classification_report
```

Las funciones utilizadas son:

Se define una función llamada *train_selection* que permite seleccionar y devolver diferentes versiones procesadas del conjunto de datos de entrenamiento según el valor del argumento *process*

```
def train_selection(process):
    if process == 's':
        X_train_proc = X_train_st
    elif process == 'n':
        X_train_proc = X_train_norm
    elif process == 'sn':
        X_train_proc = X_train_st_norm
    else:
        X_train_proc = X_train
```

```
    return X_train_proc
```

Se define una función llamada *val_selection* que permite seleccionar y devolver diferentes versiones procesadas del conjunto de datos de validación según el valor del argumento *process.*

```
def val_selection(process):
    if process == 's':
        X_val_proc = X_val_st
    elif process == 'n':
        X_val_proc = X_val_norm
    elif process == 'sn':
        X_val_proc = X_val_st_norm
    else:
        X_val_proc = X_val
    return X_val_proc
```

Se define una función llamada *test_selection* que permite seleccionar y devolver diferentes versiones procesadas del conjunto de datos de prueba según el valor del argumento *process*.

```
def test_selection(process):
    if process == 's':
        X_test_proc = X_test_st
    elif process == 'n':
        X_test_proc = X_test_norm
    elif process == 'sn':
        X_test_proc = X_test_st_norm
    else:
        X_test_proc = X_test
    return X_test_proc
```

La primera parte del código carga en diferentes DataFrames los archivos CSV que contienen los datos de entrenamiento, prueba y validación, de los cuales se extrae la columna “post” de cada uno, guardando los datos como listas.

```
f_train = './embeddings/train_embeddings.csv'
f_val = './embeddings/val_embeddings.csv'
f_test = './embeddings/test_embeddings.csv'

df_train = pd.read_csv(f_train)
df_val = pd.read_csv(f_val)
df_test = pd.read_csv(f_test)

train_posts = df_train["post"].tolist()
val_posts = df_val["post"].tolist()
test_posts = df_test["post"].tolist()
```

El siguiente código convierte en las listas que guardamos anteriormente en matrices con la ayuda de la librería *NumPy*.

```
X_train = np.array([np.fromstring(post, dtype=float, sep=' ') for post in
train_posts])
X_val = np.array([np.fromstring(post, dtype=float, sep=' ') for post in
val_posts])
X_test = np.array([np.fromstring(post, dtype=float, sep=' ') for post in
test_posts])
```

Aquí se guardan los datos de la columna “label” en listas

```
Y_train = df_train["label"].tolist()
Y_val = df_val["label"].tolist()
Y_test = df_test["label"].tolist()
```

La connotación ‘X_’ representa los vectores de las publicaciones y ‘Y_’ la clasificación de cada una.

En el siguiente codigo se crea una instancia de *StandardScaler* de la biblioteca *preprocessing* de *sklearn* y se ajusta a los datos de entrenamiento *X_train*. El *StandardScaler* se utiliza para estandarizar los vectores eliminando la media y escalando a la varianza unitaria. La propiedad *mean* devuelve la media de cada vector calculada a partir de los datos de entrenamiento *X_train* y la propiedad *scale* devuelve su desviación estándar.

```
scaler = preprocessing.StandardScaler().fit(X_train)
scaler

scaler.mean_
scaler.scale_
```

Iniciamos el proceso de normalización y estandarización de los datos de entrenamiento, para cada vector transformamos los datos para que su media sea 0 y su desviación sea 1, guardando este proceso en la variable *X_train_st*, después hacemos que se normalicen estos mismos datos con la norma ‘L2’ (la raíz cuadrada de la suma de los cuadrados de sus elementos), almacenándolo en la variable X_train_st_norm. Por último, normalizamos los datos sin antes estandarizar.

```
# Normalization and standardization
X_train_st = scaler.transform(X_train)
X_train_st_norm = preprocessing.normalize(X_train_st, norm='l2')
X_train_norm = preprocessing.normalize(X_train, norm='l2')
```

Se repiten posteriormente los pasos anteriores, pero ahora para los datos de validación y prueba, respectivamente.

```
X_val_st = scaler.transform(X_val)
X_val_st_norm = preprocessing.normalize(X_val_st, norm='l2')
X_val_norm = preprocessing.normalize(X_val, norm='l2')

X_test_st = scaler.transform(X_test)
X_test_st_norm = preprocessing.normalize(X_test_st, norm='l2')
X_test_norm = preprocessing.normalize(X_test, norm='l2')
```

- **Primer modelo: model_construction/03_standardization_and_normalization.py**

En la siguiente parte del código se buscan los mejores parámetros (*penalty*, *C*, *process*) para un modelo de regresión logística mediante la evaluación de diferentes combinaciones y selecciona la que produce el mejor puntaje F1 macro en los datos de validación. Primeramente, creamos un bucle donde se itera sobre una lista de procesos de preprocesamiento ('s', 'n', 'sn', 'o'). Para cada proceso, se seleccionan los datos de entrenamiento y validación preprocesados usando las funciones *train_selection* y *val_selection*. Posteriormente, se itera sobre dos tipos de penalización ('l1' y 'l2') y una lista de valores para el parámetro de regularización C (0.01, 0.1, 1, 10, 100), donde se creará y entrenará un modelo de regresión logística usando los datos de entrenamiento preprocesados (*X_train_proc*), para después realizar predicciones sobre los datos de validación preprocesados (*X_val_proc*) se calcular la precisión (*accuracy*) y el puntaje F1 macro (*macrof1*) de las predicciones.

```
best_f1 = 0
best_parameters = ()
for process in ['s', 'n', 'sn', 'o']:
    X_train_proc = train_selection(process)
    X_val_proc = val_selection(process)

    for penalty in ['l1', 'l2']:
        for C in [0.01, 0.1, 1, 10, 100]:
            clf = LogisticRegression(random_state=0, penalty=penalty,
solver='liblinear', C=C, max_iter=1000)
            clf.fit(X_train_proc, Y_train)
            prediction = clf.predict(X_val_proc)
            accuracy = accuracy_score(Y_val, prediction)
            macrof1 = f1_score(Y_val, prediction, average='macro')

            if macrof1 > best_f1:
               best_parameters = (penalty, C, process)
               best_f1 = macrof1
               best_accuracy = accuracy
               best_cm = multilabel_confusion_matrix(Y_val, prediction)
               best_cr = classification_report(Y_val, prediction,
target_names=['fake', 'true'])
```

La siguiente parte del código combina los datos de entrenamiento y validación preprocesados en un solo conjunto, crea un array de etiquetas correspondiente, y preprocesa los datos de prueba usando el mejor proceso encontrado durante la búsqueda anterior.

```
X_total_proc = np.concatenate((train_selection(best_parameters[2]),
val_selection(best_parameters[2])))
Y_total = np.array(Y_train + Y_val)
X_test_proc = test_selection(best_parameters[2])
```

Para finalizar, el siguiente código entrena un modelo de regresión logística con los mejores parámetros encontrados, realiza predicciones sobre los datos de prueba, y evalúa el rendimiento del modelo usando varias métricas.

```
clf = LogisticRegression(random_state=0, penalty=best_parameters[0],
C=best_parameters[1], max_iter=1000, solver='liblinear')
```

```
clf.fit(X_total_proc, Y_total)
prediction = clf.predict(X_test_proc)
accuracy = accuracy_score(Y_test, prediction)
macrof1 = f1_score(Y_test, prediction, average='macro')
matrix = multilabel_confusion_matrix(Y_test, prediction)
cr = classification_report(Y_test, prediction, target_names=['fake',
'true'])
```

- **Segundo modelo: processing/01_get_clean_words_bert.py**

Su objetivo principal es limpiar las publicaciones. Las librerías utilizadas son:

```
import pandas as pd
import re
```

Las funciones que contiene:

clean_post se encarga de quitar hipervínculos en las publicaciones

```
def clean_post(post, urls):
    for url in urls:
        post = post.replace(url,'')
    return post
```

El programa cuenta con un código principal en el cual se hace la limpieza de cada una de las publicaciones y donde al final se guardan en un archivo .csv para poder trabajar posteriormente con él.

La siguiente parte nos muestra una expresión regular para detectar URLS dentro de un texto

```
# Pattern to match url links in a text
URLS = r"""     # Capture 1: entire matched URL
  (?:
  https?:       # URL protocol and colon
    (?:
      /{1,3}        # 1-3 slashes
      |         #   or
      [a-z0-9%]       # Single letter or digit or '%'
                                    # (Trying not to match e.g.
"URI::Escape")
    )
    |         #   or
                                    # looks like domain name followed
by a slash:
    [a-z0-9.\-]+[.]
    (?:[a-z]{2,13})
    /
  )
```

```
  (?:         # One or more:
    [^\s()<>{}\[\]]+      # Run of non-space, non-()<>{}[]
    |         #   or
    \([^\s()]*?\([^\s()]+\)[^\s()]*?\) # balanced parens, one level deep:
(...(...)...)
    |
    \([^\s]+?\)       # balanced parens, non-recursive: (...)
  )+
  (?:         # End with:
    \([^\s()]*?\([^\s()]+\)[^\s()]*?\) # balanced parens, one level deep:
(...(...)...)
    |
    \([^\s]+?\)       # balanced parens, non-recursive: (...)
    |         #   or
    [^\s`!()\[\]{};:'".,<>?«»“”‘’]  # not a space or one of these punct
chars
  )
  |         # OR, the following to match naked domains:
  (?:
    (?<!@)              # not preceded by a @, avoid matching
foo@_gmail.com_
    [a-z0-9]+
    (?:[.\-][a-z0-9]+)*
    [.]
    (?:[a-z]{2,13})
    \b
    /?
    (?!@)             # not succeeded by a @,
                      # avoid matching "foo.na" in
"foo.na@example.com"
  )
"""
```

En esta parte definimos los archivos de entrada y de salida, *f_name* representa la ruta completa del archivo de entrada.; *f_clean* representa la ruta completa del archivo de salida.

```
# Path to file
w_d = 'D:/data/Documentos/ug/proyectos/2024/verano_ciencia_ug/data/'
f_name = w_d + 'constraint_test_with_labels.csv' # name of input file
f_clean = w_d + 'test_bert.csv' # name of output file
```

En la siguiente parte se crea un objeto de expresión regular con la función *re.compile* a partir del patrón que se generó con *URLS*, de esta manera podemos buscar y detectar URLs las veces que sean necesarias

```
# Compile the pattern for matching
url_re = re.compile(URLS, re.VERBOSE | re.I | re.UNICODE)
```

En la siguiente parte almacenamos el archivo CSV que contiene la variable *f_name* en un DataFrame.

```
# Read the input file as csv to create a DataFrame
df = pd.read_csv(f_name, encoding='utf-8')
```

La siguiente parte del código se utiliza para la limpieza del CSV, donde obtiene la etiqueta de las publicaciones, después busca y limpia las URL's en la publicación, para al final almacenarlas en una lista *l_clean* junto con sus respectivas etiquetas

```
# List to append the clean posts with their labels
l_clean = []
labels = {'fake':0, 'real':1}

for post in df.itertuples():
    label = post[3]
    post = post[2].lower()
    urls = url_re.findall(post)
    post = clean_post(post, urls)
    l_clean.append({'post':post, 'label':labels[label]})
```

Para finalizar el procesamiento en el modelo de transformadores, creamos un nuevo DataFrame con los datos obtenidos en la limpieza, para posteriormente guardarlo en un archivo CSV

```
# New DataFrame with the clean data
df_clean = pd.DataFrame(l_clean)

# Write the new DF to a CSV file (without the index)
df_clean.to_csv(f_clean, index=False)
```

- **Segundo modelo: model_construction/02_train_test_bert_models.py**

El objetivo principal de este código es la construcción y entrenamiento de los modelos transformadores. Para la generación del modelo se utilizaron los módulos *ktrain*, *tensorflow*, *sklern* y *keras.* Y como transformadores *BERT, ELECTRA, RoBERTa* y *XLM-RoBERTa*.

```
import os
# Define the enviroment variable
os.environ['TF_USE_LEGACY_KERAS']="1"
from tensorflow.keras.callbacks import EarlyStopping
from sklearn import metrics
from ktrain import text
import pandas as pd
import ktrain
import time
```

Como función tenemos la siguiente, donde esta función toma un archivo CSV como entrada y devuelve dos listas: una para los valores de la primera columna y otra para los valores de la última columna.

```
def read_data(file):
    df = pd.read_csv(file)
    X = df[df.columns[0]]
    Y = df[df.columns[-1]]
    return X.tolist(), Y.tolist()
```

Como primera parte del cuerpo del código tenemos que, la variable *w_d* contiene la ruta del directorio en el que trabajamos, y las variables *f_train*, *f_val* y *f_test* contendrán la concatenación de *w_d* y el nombre de los archivos para entrenamiento, validación, y prueba, respectivamente.

```
w_d = 'D:/data/Documentos/ug/proyectos/2024/verano_ciencia_ug/data/'
f_train = w_d + 'train_bert.csv' # name of train file
f_val = w_d + 'validation_bert.csv' # name of validation file
f_test = w_d + 'test_bert.csv' # name of test file
```

La siguiente parte carga en las variables X_train, X_val y X_test las publicaciones de los archivos entrenamiento, validación y prueba, y en las variables Y_train, Y_val y Y_test contienen sus respectivas etiquetas.

```
# Read and split data
X_train, Y_train = read_data(f_train) # Trainin data
X_val, Y_val = read_data(f_val) # Validation data
X_test, Y_test = read_data(f_test) # Test data
```

Aquí solo se guardan las etiquetas únicas que pueden llegar a existir en una lista

```
# List of classes
labels = list(set(Y_train))
```

En la siguiente parte del codigo, establecemos el modelo a utilizar, en este caso sería el modelo *BERT*

```
model_name = 'bert-base-uncased'
```

Después creamos una instancia del modelo de preprocesamiento, con los parámetros que nosotros necesitamos.

```
# Instanciates the preprocessing model
preproc = text.Transformer(model_name, maxlen=512, class_names=labels)
```

Continuamos con el preprocesamiento de los datos, utilizando la instancia anterior y aplicándola a los datos de entrenamiento y validación.

```
# Data preprocesing: training and validation
X_train_proc = preproc.preprocess_train(X_train, Y_train)
X_val_proc = preproc.preprocess_test(X_val, Y_val)
```

En la siguiente parte del código, creamos una instancia correspondiente a un clasificador preentrenado del modelo.

```
# Instanciates a classifier
model = preproc.get_classifier()
```

Aquí creamos el aprendiz que creara y ajustara el modelo con los parámetros que definimos anteriormente.

```
# Creates a learner with the defined parameters
```

```
learner = ktrain.get_learner(model,
                             train_data=X_train_proc,
                             val_data=X_val_proc,
                             batch_size=6)
```

Posteriormente, ajustamos el modelo utilizando el método fit_onecycle(), donde establecemos la tasa de aprendizaje, indicamos el número de épocas (iteraciones completas sobre el conjunto de datos), que tanta información se muestra durante cada época, y para finalizar monitoreamos el proceso de entrenamiento donde calculamos la diferencia entre el tiempo actual y el tiempo al inicio de este

```
# Fit the learner with the defined parameters
tr_time = time.time()
learner.fit_onecycle(lr=9e-6,
                     epochs=7,
                     verbose=1,
                     callbacks=[EarlyStopping(monitor='val_auc',
                                              patience=2,
                                              mode='max',
                                              restore_best_weights=True)])
tr_time = time.time() - tr_time
```

En la siguiente parte del código creamos un objeto predictor, donde este utiliza el modelo de clasificación y el modelo de preprocesamiento para realizar predicciones en nuevos datos.

```
### Predict with the learned model
# Get the predictor and use the defined preprocessing model
predictor = ktrain.get_predictor(learner.model, preproc)
```

Ahora realizaremos la predicción de las etiquetas y la probabilidad de pertenencia a cada una de ellas en los datos de prueba, así como también almacenamos el tiempo que llega a tomar este proceso.

```
# Predict classes and probabilities over the test data
ts_time = time.time()
prediction = predictor.predict(X_test)
prediction_proba = predictor.predict_proba(X_test)
ts_time = time.time() - ts_time
```

En la siguiente parte evaluamos el rendimiento del modelo para el conjunto de datos de prueba, utilizando diferentes métricas y reportes.

```
# Evaluate the classification with several metrics,
# contingency table and classification report
target_names = ['fake', 'real']
accuracy = metrics.accuracy_score(Y_test, prediction)
f1_macro = metrics.f1_score(Y_test, prediction, average='macro')
prec_macro = metrics.precision_score(Y_test, prediction, average='macro')
rec_macro = metrics.recall_score(Y_test, prediction, average='macro')
f1_micro = metrics.f1_score(Y_test, prediction, average='micro')
prec_micro = metrics.precision_score(Y_test, prediction, average='micro')
rec_micro = metrics.recall_score(Y_test, prediction, average='micro')
```

```
matrix = metrics.confusion_matrix(Y_test, prediction)
cr = metrics.classification_report(Y_test, prediction,
target_names=target_names)
```

Por último, evaluamos el rendimiento del modelo para el conjunto de datos de validación, utilizando diferentes métricas y reportes.

```
## To obtain classification results with the validation set
prediction_val = predictor.predict(X_val)
prediction_val_proba = predictor.predict_proba(X_val)
accuracy_val = metrics.accuracy_score(Y_val, prediction_val)
f1_macro_val = metrics.f1_score(Y_val, prediction_val, average='macro')
prec_macro_val = metrics.precision_score(Y_val, prediction_val,
average='macro')
rec_macro_val = metrics.recall_score(Y_val, prediction_val,
average='macro')
f1_micro_val = metrics.f1_score(Y_val, prediction_val, average='micro')
prec_micro_val = metrics.precision_score(Y_val, prediction_val,
average='micro')
rec_micro_val = metrics.recall_score(Y_val, prediction_val,
average='micro')
matrix_val = metrics.confusion_matrix(Y_val, prediction_val)
cr_val = metrics.classification_report(Y_val, prediction_val,
target_names=target_names)
```