Fases de aprendizaje y de predicción

Los datos están preparados y ahora podemos pasar a las fases de aprendizaje y de predicción.

1. Distribución en conjuntos de pruebas y de aprendizaje

Como solemos hacer en estos casos, creamos un conjunto de aprendizaje y un conjunto de pruebas que contienen respectivamente un 80 % y un 20 % de las observaciones como información para predecir la característica CREENCIA.

from sklearn.model\_selection import train\_test\_split

**X\_train, X\_test, y\_train, y\_test =**

**train\_test\_split(mensajesTwitter['TWEET'].values,**

**mensajesTwitter['CREENCIA'].values,test\_size=0.2)**

Hemos optado por conservar los términos anglosajones (train y test) porque los encontrará de este modo en los distintos casos prácticos disponibles en Internet.

2. Creación de una canalización de aprendizaje

Una canalización de aprendizaje puede asimilarse a un conjunto de acciones consecutivas que se han de realizar sobre el conjunto de observaciones.

La primera acción añadida a la canalización se hará con la función CountVectorizer() y consistirá en crear la matriz de las apariciones de las distintas palabras en las distintas frases, como hemos aprendido a hacerlo de manera manual.

La matriz de apariciones de las palabras sirve para determinar la cantidad de veces que aparece una palabra en un texto. Sin embargo, eso no significa que una palabra que tiene una cantidad importante de apariciones sea determinante en la clasificación del mensaje. En efecto, puede suceder que la misma palabra se repita 10 veces en un mensaje, pero que solo se haya utilizado en un único mensaje. Es preferible dar un peso a las palabras en función de su frecuencia de aparición en un mensaje (Terme Frequency) y de su cantidad de apariciones en el conjunto de los mensajes (Inverse document frequency). Este peso, llamado TF-IDF, también sirve para expresar la escasez de la palabra en el conjunto de los mensajes.

No vamos a entrar en detalles sobre la fórmula matemática de determinación de este peso. No obstante, tiene que saber que:

* TF-IDF disminuye cuando una palabra está presente en muchos mensajes.
* TF-IDF también disminuye cuando está poco presente en un mensaje.
* TF-IDF es máximo para las palabras poco frecuentes que aparecen mucho en el conjunto de los mensajes que tenemos para analizar.

Así, en las siguientes frases:

El perro juega fuera con otros perros con una pelota azul.

El perro y los otros perros juegan fuera con una pelota roja.

* La palabra «perro» parece varias veces en las frases, su TF-IDF será bajo.
* Las palabras «azul» y «rojo» aparecen pocas veces, su TF-IDF también será bajo.
* Es evidente que la palabra «pelota» aparece pocas veces en las dos frases, pero al menos una vez en las dos. Por lo tanto, su frecuencia (TF) es débil, pero aparece en las dos frases (IDF). Eso significa que su TF-IDF será elevado.

El TF\_IDF se calcula con la función TdifTansformer() del módulo Scikit-Learn, que añadimos como segunda acción en nuestra canalización.

La tercera acción añadida a la canalización es el uso del algoritmo bayesiano ingenuo múltiple (MultinomialNB()), que permitirá realizar el aprendizaje:

from sklearn.pipeline import Pipeline

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn.naive\_bayes import MultinomialNB

**etapas\_aprendizaje = Pipeline([('frequence',**

**CountVectorizer()),**

**('tfidf', TfidfTransformer()),**

**('algoritmo', MultinomialNB())])**

3. Aprendizaje y análisis de los resultados

Ahora podemos iniciar la fase de aprendizaje y verificar sus resultados en los datos de las pruebas con la función classification\_report, tomando como parámetros:

* Los resultados de las observaciones de pruebas que se han de predecir.
* La predicción y la precisión decimal (digits) de los resultados mostrados.

modelo = etapas\_aprendizaje.fit(X\_train,y\_train)

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, modelo.predict(X\_test),

digits=4))

|  | **Precisión** | **Recall** | **F1-score** | **Support** |
| --- | --- | --- | --- | --- |
| 0 | 0.8519 | 0.3067 | 0.4510 | 225 |
| 1 | 0.7958 | 0.9806 | 0.8786 | 620 |
| Micro avg | 0.8012 | 0.8012 | 0.8012 | 845 |
| Macro avg | 0.8238 | 0.6437 | 0.6648 | 845 |
| Weight avg | **0.8107** | 0.8012 | 0.7647 | 845 |

A la vista de los números, podemos afirmar que obtenemos una precisión de clasificación del 81 %, ¡no está mal!

4. Clasificación de un mensaje nuevo

Ahora podemos probar nuestro modelo de aprendizaje proponiéndole el mensaje nuevo que aparece a continuación:

Why should trust scientists with global warming if they didnt

know Pluto wasnt a planet

*¿Por qué confiar en los científicos acerca del calentamiento*

*climático si no sabían que Plutón no era un planeta?*

Por supuesto, es conveniente normalizar las frases y proceder con las etapas de eliminación de las *stop words*, de aplicación de *stemming* y de lematización antes de proceder con la clasificación.

frase = "Why should trust scientists with global warming if they

didnt know Pluto wasnt a planet"

print(frase)

#Normalización

frase = normalisation(frase)

#Eliminación de las stops words

frase = ' '.join([palabra for palabra in frase.split()

if palabra not in (stopWords)])

#Aplicación de stemming

frase =  ' '.join([stemmer.stem(mot) for palabra in frase.split(' ')])

#Lematización

frase = ' '.join([lemmatizer.lemmatize(palabra) for palabra in

frase.split(' ')])

print (frase)

prediccion = modelo.predict([frase])

print(prediccion)

if(prediccion[0]==0):

   print(">> No cree en el calentamiento climático...")

else:

   print(">> Cree en el calentamiento climático...")

El resultado de la clasificación es el siguiente:

>> No cree en el calentamiento climático...

Esto parece coherente con el sentido de la frase.