Clasificador de SPAM - NLP

May 31, 2021

1 Clasificador de SPAM

NLP

Por: Miguel Angel Soto Hernandez

1.1 Planteamiento del problema

El correo electrónico es una de las más potentes herramientas que la tecnología ha puesto a disposición del público en general para optimizar procesos y aumentar la productividad individual y colectiva. Mandar mensajes, documentos, fotografías, archivos, desde cualquier lugar del mundo a cualquier otro de forma instantánea y tan económica es de tal trascendencia que ha cambiado, de forma definitiva, los hábitos de todos.

A pesar de todo lo bueno que esto conlleva surge un cada vez mayor problema: impedir todos aquellos mensajes no deseados. Se trata de un problema que está generando un inevitable retraso en el desarrollo, por la desconfianza que genera en las personas.

2 Objetivo

Realizar un modelo que sea capaz de clasificar correctamente si un mensaje es spam o no, a partir de un conjunto de datos de entrenamiento.

A continuación se muestra un ejemplo de cada posible salida de nuestro modelo:

Spam > Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's

No SPAM > Oops, I'll let you know when my roommate's done

2.1 Conjunto de datos

El conjunto de datos que se utilizará para llevar a cabo esta tarea es de uso libre, y se puede encontrar accediendo a la siguiente liga:

Conjunto de datos

Para iniciar, necesitaremos importar las librerías básicas para poder manejar nuestro conjunto de datos

[2]:
```
import csv
import pandas as pd
import numpy as np
```

Para descargar el dataset ejecutaremos el siguiente comando:

[ ]:
```
![ ! -f spam.csv ] && wget https://raw.githubusercontent.com/mohitgupta-omg/
↪Kaggle-SMS-Spam-Collection-Dataset-/master/spam.csv
```

3 Marcar conjunto de datos

Convertimos el conjunto de datos en un DataFrame con la librería pandas y le asignamos nombres a las columnas del mismo.

[3]:
```
spam_o_no_spam = pd.read_csv("spam.csv", encoding='latin-1')[["v1", "v2"]]
spam_o_no_spam.columns = ['etiqueta', 'texto']
spam_o_no_spam.head()
```

[3]:

	etiqueta	texto
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

La etiqueta para los mensajes spam es 'spam', mientras que la etiqueta para los textos de no spam es 'ham'.

Para ver cuantos textos tenemos por cada etiqueta, ejecutamos el siguiente comando:

[4]:
```
spam_o_no_spam['etiqueta'].value_counts()
```

[4]:
```
ham     4825
spam     747
Name: etiqueta, dtype: int64
```

3.1 Extracción de características

- Tokenización: convertir un párrafo u oración a tokens, usualmente cada palabra es un token. En este caso, nuestra función `tokenize` es bastante simple (e ineficiente), pero sirve para nuestros simple propósito.
- Remover stopwords: eliminar tokens irrelevantes, tales como, palabas comunes y en ciertos casos signos de puntuación. En nuestro caso, únicamente estamos eliminando los símbolos de puntuación con ayuda del set `punctuation`.

[7]:
```
import string
puntuacion = set(string.punctuation)

def tokenizar(texto):
    tokens = []

    for token in texto.split():
        nuevo_token = []

        for caracter in token:
            if caracter not in puntuacion:
```

```
                nuevo_token.append(caracter.lower())

        if nuevo_token:
            tokens.append("".join(nuevo_token))

    return tokens
```

Probamos con un ejemplo sencillo que nuestra función `tokenizar` funcione corectamente

[8]:
```
tokenizar("Go until jurong point, crazy.. ")
```

[8]:
```
['go', 'until', 'jurong', 'point', 'crazy']
```

Ahora aplicamos la funcion `tokenizar` a nuestros textos de nuestro conjunto de datos

[9]:
```
spam_o_no_spam.head()['texto'].apply(tokenizar)
```

[9]:
```
0    [go, until, jurong, point, crazy, available, o...
1                        [ok, lar, joking, wif, u, oni]
2    [free, entry, in, 2, a, wkly, comp, to, win, f...
3    [u, dun, say, so, early, hor, u, c, already, t...
4    [nah, i, dont, think, he, goes, to, usf, he, l...
Name: texto, dtype: object
```

- Stemming/Lemmatización: convertir cada token a su forma base. En nuestro caso no estamos haciendo este paso, pero si es necesario, puedes revisar cosas como NLTK - stemming o acercamientos a lematización en Python
- One-Hot encoding: después de la tokenización, poner en una tabla todos los tokens en el vocabulario y por cada ocurrencia de un token en un texto, marcar con un 1 en la fila correspondiente, por ejemplo considerando las dos frases siguientes:

1. Call FREEPHONE 0800 542 0578 now!
2. Did you call me just now ah?

Obtendríamos algo como esto:

	0578	0800	542	ah	call	did	freephone	just	me	now	you
1	1	1	1	0	1	0	1	0	0	1	0
2	0	0	0	1	1	1	0	1	1	1	1

Aquí es donde entra **Scikit-Learn** a través de la clase `CountVectorizer` del módulo `sklearn.feature_extraction.text`.

[10]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

[17]:
```
vectorizador_demo = CountVectorizer(
    tokenizer = tokenizar,
    binary=True
)
```

Parámetros:

- tokenizer = tokenize: CountVectorizer tiene un tokenizador por default, al pasarle nuestra función lo estamos reemplazando con el que nosotros escribimos.
- binary = True: CountVectorizer por default en lugar de 1 cuenta el número de ocurrencias de cada token, al establecer binary = True, le estamos indicando que no importa cuantas veces ocurra una palabra, solamente la debe contar una vez

[18]:
```
ejemplos = [
    "Call FREEPHONE 0800 542 0578 now!",
    "Did you call me just now ah?"
]

vectorizador_demo.fit(ejemplos)
vectores = vectorizador_demo.transform(ejemplos).toarray()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/feature_extraction/text.py:507:
UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is
not None'
  warnings.warn("The parameter 'token_pattern' will not be used"
```

Usamos `fit` y `transform` de manera separada, aunque en este caso pudimos haber usado `fit_transform`.

Nota: usamos `toarray` para obtener un un *numpy array* ya que por default `transform` devuelve una matriz dispersa que, mientras que es buena para no consumir memoria, no es tan amigable para mostrar cómo es que se ven los datos.

[19]:
```
cabeceras = sorted(vectorizador_demo.vocabulary_.keys())
pd.DataFrame(vectores, columns=cabeceras)
```

[19]:

	0578	0800	542	ah	call	did	freephone	just	me	now	you
0	1	1	1	0	1	0	1	0	0	1	0
1	0	0	0	1	1	1	0	1	1	1	1

Separamos el conjunto de datos en datos de entrenamiento y datos de prueba con la función `train_test_split` del módulo `sklearn.model_selection`

[21]:
```
from sklearn.model_selection import train_test_split

texto_entrenamiento, texto_prueba, etiquetas_entrenamiento, etiquetas_prueba = \
    train_test_split(spam_o_no_spam["texto"], spam_o_no_spam["etiqueta"],
                     stratify=spam_o_no_spam["etiqueta"])

print(f"Ejemplos de entrenamiento: {len(texto_entrenamiento)}, ejemplos de␣
 ↪prueba {len(texto_prueba)}")
```

```
Ejemplos de entrenamiento: 4179, ejemplos de prueba 1393
```

Una vez separados los datos, podemos comenzar a entrenar nuestro algoritmo, comenzando por generar un nuevo vectorizador:

[22]:
```
vectorizador = CountVectorizer(tokenizer=tokenizar, binary=True)
```

```
entrenamiento_X = vectorizador.fit_transform(texto_entrenamiento)
prueba_X = vectorizador.transform(texto_prueba)

entrenamiento_X.shape
```

[22]: `(4179, 8146)`

- fit: procesa los datos y prepara el clasificador, ejecuta la parte del aprendizaje basdo en los datos de entrenamiento.
- predict: ejecuta la parte de la predicción del algoritmo, basándose en la información adquirida al ejecutar `fit`.

[23]:
```
from sklearn.svm import LinearSVC
```

[24]:
```
clasificador = LinearSVC()
clasificador.fit(entrenamiento_X, etiquetas_entrenamiento)
```

[24]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

[25]:
```
from sklearn.metrics import accuracy_score

predicciones = clasificador.predict(prueba_X)

exactitud = accuracy_score(etiquetas_prueba, predicciones)

print(f"Exactitud: {exactitud:.4%}")
```

```
Exactitud: 98.4207%
```

3.2 Predicciones con datos nuevos

[26]:
```
spam = "Want to win FREE tickets to a football match? txt WIN"
ham = "Do you want to go to a football match with me?"

ejemplos = [
    spam,
    ham
]

ejemplos_X = vectorizador.transform(ejemplos)
predicciones = clasificador.predict(ejemplos_X)
```

[27]:
```
for texto, etiqueta in zip(ejemplos, predicciones):
    print(f'{etiqueta:5} - {texto}')
```

```
spam  - Want to win FREE tickets to a football match? txt WIN
ham   - Do you want to go to a football match with me?
```

[ ]: