# 

Ingeniería Lingüística

2019-2020

Práctica 2

# 

# Alumnos:

# Lydia González Cid

# Carmen Bermejo Hernández

# 

# 

# 

# Índice:

# 1- Introducción

# 2- Preparación de los documentos

# 3- Entrenamiento

# 4- Pruebas de rendimiento

# 5- Conclusiones

# 1- Introducción

Se trata de hacer un clasificador de documentos que sea capar de distinguir entre diferentes categorías de noticias.

Para implementarlo hemos escogido el lenguaje Python y la librería Sklearn que incluye la mayoría de la funcionalidad que necesitábamos para esta tarea.

TO DO: Escribir más aquí

# 2- Preparación de los documentos

Hemos obtenido los documentos de la plataforma Social Clipping, que realiza una búsqueda de las últimas noticias en multiples medios online. Para el ejercicio hemos elegido las categorías salud, política y tecnología.

Hemos descargado el resultado de la búsqueda en estas tres categorías, obteniendo noticias de los últimos dos meses.

**2.1 – Scripts de limpieza y generación de documentos:**

Como ocurre en muchos casos cuando se trabaja con datos que no han sido preparados para trabajar con ellos, el documento obtenido presentaba varios problemas, que hemos resuelto implementando dos scripts.

El primer script, clean\_documents.py resuelve los siguientes problemas:

- Elimina líneas intercaladas en el documento que no pertenecen a la noticia, como la que indica el número de página de la búsqueda.

- Arregla caracteres especiales del español, como la ñ y los acentos, que la fuente original no leía bien.

- Añade un delimitador que separa claramente cada una de las noticias contenidas en el documento.

El segundo script, generate\_documents.py, parte los documentos en las diferentes noticias, generando un documento por noticia.

**2.2 – Sets de entrenamiento y test:**

Al descargar todos los documentos disponibles que nos permitía social clipping, esto es todas las noticias disponibles para cada categoría en los últimos dos meses, hemos obtenido diferente cantidad de noticias para cada categoría. 73 noticias de política, 64 de salud y 67 de tecnología.

Para el set de entrenamiento hemos escogido las 40 primeras noticias de cada categoría, dejando para el dataset de testing el resto.

**2.3 – Construcción del dataset:**

La librería Sklearn cuenta con una funcionalidad que te permite crear la estructura de datasets, lista para ser usada por el resto de herramientas de la librería.

La funcionalidad está implementada en el método [5]:

sklearn.datasets.load\_files

Para usarla sólo es necesario indicar como parámetro el directorio donde se encuentran los archivos con los ejemplos de texto, cada uno dentro de una carpeta correspondiente a la categoría a la que pertenecen los ejemplos.

# 3- Entrenamiento

Como hemos mencionado anteriormente, la librería sklearn facilita mucho nuestro trabajo a la hora de implementar el clasificador.

El clasificador funciona en tres pasos:

**1- Genera una matriz de términos:**

Esta matriz contiene el número de veces que aparece cada término dentro de cada documento.

En sklearn esto se hace por medio de la clase CountVectorizer(), que recibiendo como entrada un dataset, devuelve dicha matriz de términos.

**2- Refina y mejora los datos de la matriz:**

En este paso los datos se modifican para hacer la información más útil con respecto al problema a resolver.

- Primero, los documentos de mayor longitud, al contener más cantidad de palabras en total, presentan más veces varios términos que documentos más cortos. Esto no nos ayuda a clasificar el documento. Es más útil tener la frecuencia de términos con respecto al total de palabras del documento.

- Segundo, hay algunos términos que aparecen de forma bastante frecuente en todos los documentos, independientemente de su categoría. Esta información tampoco nos resulta útli para clasificar los documentos y genera ruido. Por ello, es más útil si reducimos la importancia de estos términos en la matriz.

Ambas mejoras se implementan en Sklearn con la clase TfidfTransformer(), que recibe como entrada la matriz de términos creada en el paso anterior y devuelve la matriz modificada conforme a estas mejoras.

**3- Entrena el clasificador:**

El último paso es entrenar al clasificador. Sklearn permite crear clasificadores de varios tipos. Nosotras hemos escogido probar con dos métodos: Naive Bayes y SVM.

La clase MultinomialNB() permite crear una instancia de un clasificador Naive Bayes.

La clase SGDClassifier instancia un clasificador SVM (Support Vector Machines).

TO DO: Hablar de Naive Bayes y SVM

Incluir referencias [2], [3] y [6].

La instancia del clasificador, una vez creada, puede ser entrenada mediante el método fit(), que recibe como entrada la matriz de términos obtenida en los pasos anteriores y un vector que indica las categorías de cada uno de los ejemplos.

Una vez entrenado el clasificador, está ya listo para ser usado.

**Pipeline:**

En nuestra implementación, hemos optado por comprimir los tres pasos que acabamos de describir, usando la herramienta de Sklearn Pipeline, que permite encadenar CountVectorizer, TfidfTransformer y el clasificador, en una misma clase que recibe como entrada el dataset de entrenamiento y devuelve el clasificador ya entrenado.

*classifier = Pipeline([('vectorizer', CountVectorizer()),*

*('transformer', TfidfTransformer()),*

*('nb\_classifier', MultinomialNB())])*

# 4- Pruebas de rendimiento:

Una vez tenemos listo nuestro clasificador, lo probamos para comprobar su rendimiento.

El clasificador creado en el paso anterior cuenta con la función predict que, recibiendo como entrada un dataset compuesto de documentos a clasificar, nos dá como salida qué clase predice el clasificador que es cada uno de esos documentos.

Para comprobar el rendimiento del clasificador, utilizamos el dataset de testing. Como dijimos en el punto 2, hemos entrenado el clasificador con 40 ejemplos de cada categoría, dejado el resto de ejemplos para el dataset de testing, que son 33 noticias de política, 24 de salud y 27 de tecnología.

Utilizamos la función predict con el dataset de testing y obtenemos la clasificación que este predice para cada uno de los documentos.

*predicted = classifier.predict(test\_data.data)*

Una vez tenemos esa predicción, contrastamos la clasificación predicha con la real y calculamos la media de aciertos. Para este paso utilizamos la herramienta numpy.

*mean\_prediction = np.mean(predicted == test\_data.target)*

- predicted es un array que indica la clase predicha para cada uno de los ejemplos.

- test\_data es el dataset de testing.

- test\_data.target es el array que indica la categoría real de cada uno de los ejemplos de ese dataset.

Modern Information Retrieval 2nd ed

TO DO: Sacar tablas de la performance en las diferentes categorías. Hacer más pruebas de rendimiento.

# 5- Conclusiones

TO DO: Escribir las conclusiones

# Referencias

1 Machine Learning, NLP: Text Classification using scikit-learn, python and NLTK.

<https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a>

2 Introduction to Information Retrieval - Text classification and Naive Bayes

<https://nlp.stanford.edu/IR-book/html/htmledition/text-classification-and-naive-bayes-1.html>

3 Introduction to Information Retrieval - Support vector machines and machine learning on documents

<https://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-and-machine-learning-on-documents-1.html>

4 Sklearn documentation - Working With Text Data

<https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html>

5 Sklearn documentation - datasets.load\_files

<https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_files.html>

6 Sklearn documentation - Support Vector Machines

<https://scikit-learn.org/stable/modules/svm.html>

7 Modern Information Retrieval: The Concepts and Technology behind Search (2nd Edition) (ACM Press Books). Ricardo Baeza-Yates, Berthier Ribeiro-Neto