Ingeniería Lingüística

2019 - 2020

Práctica II - Clasificación de documentos

Alumnos:

Carmen Bermejo Hernández
Lydia González Cid

POLITÉCNICA

Universidad Politécnica de Madrid
E.T.S.I. Informáticos

Índice

1. Introducción

El objetivo de la presente práctica es diseñar e implementar un **clasificador documental**, el cual será capaz de distinguir entre diferentes temáticas de noticias.

Para implementarlo se ha escogido el lenguaje Python [1], principalmente junto a la librería Sklearn [2], la cual está enfocada en aprendizaje automático e incluye varias de las funcionalidades que se buscan para esta tarea.

Los documentos con los que se trabajará son noticias extraídas de *Social Clipping* [3], una herramienta enfocada a proporcionar un punto común de acceso a la información publicada en distintos medios, países e idiomas sobre las temáticas elegidas.

Una vez obtenidos los documentos, se ha llevado a cabo su **preprocesamiento**, fase que se describe a lo largo de la sección 2 e incluye la limpieza, y la preparación de los datasets de entrenamiento y testing.

Con esto, se ha procedido a la **implementación y entrenamiento** del sistema de clasificación, fase detallada en la sección 3 y durante la cual se han probado distintos métodos de clasificación, basados en Naive Bayes y Support Vector Machines (SVM).

A continuación se ha realizado el proceso de **evaluación** del sistema obtenido, probando el funcionamiento del sistema y su rendimiento en tres casos distintos, todo ello descrito en la sección 4.

El siguiente paso ha sido la implementación de un sistema que clasifica documentos, colocándolos en sus carpetas correctas. Para esto se ha optado por utilizar método de clasificación Support Vector Machines (SVM), que dio buen resultado en el paso anterior.

Por último, se incluyen en la sección 6 una serie de **conclusiones** finales sobre el proceso y los resultados obtenidos.

2. Preprocesamiento de los documentos

Se han extraído los documentos de la herramienta *Social Clipping*, capaz de realizar una búsqueda de las noticias más recientes en múltiples medios online. Para la presente práctica, se han elegido las siguientes opciones entre las disponibles en la aplicación:

- Medios: todos los disponibles

- Rango temporal: dos meses (rango máximo)
- País: todos
- Idioma: español
- Temáticas: salud, política y tecnología

Se ha descargado el resultado completo de la búsqueda en estas tres categorías, y se han seguido los pasos descritos a continuación.

2.1. Scripts de limpieza y generación de documentos

Como ocurre en muchos casos al trabajar con datos que no han sido procesados previamente, los archivos descargados presentaban algunos inconvenientes, los cuales se han resuelto implementando los siguientes dos scripts.

El primer script, *clean_documents.py*, se encarga de resolver los siguientes problemas:

- Elimina líneas intercaladas en el documento que no pertenecen al contenido de la noticia, como la que indica el número de página de la búsqueda.
- Repara caracteres especiales del español, como la "ñ" y las letras con tilde, puesto que al descargar los textos de la fuente original se produjeron errores de codificación.
- Añade un delimitador (===) que separa claramente cada una de las noticias contenidas en el documento.

El efecto del script de limpieza se demuestra a continuación en las figuras 1 y 2, en las que se observa parte de las primeras líneas del archivo de noticias de política, antes y después de ejecutar sobre él *clean_documents.py*:

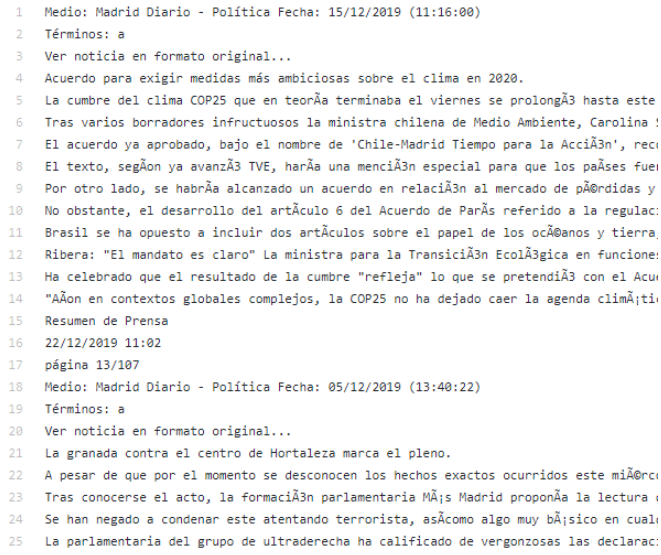

Figura 1: Archivo de noticias de política recién descargado

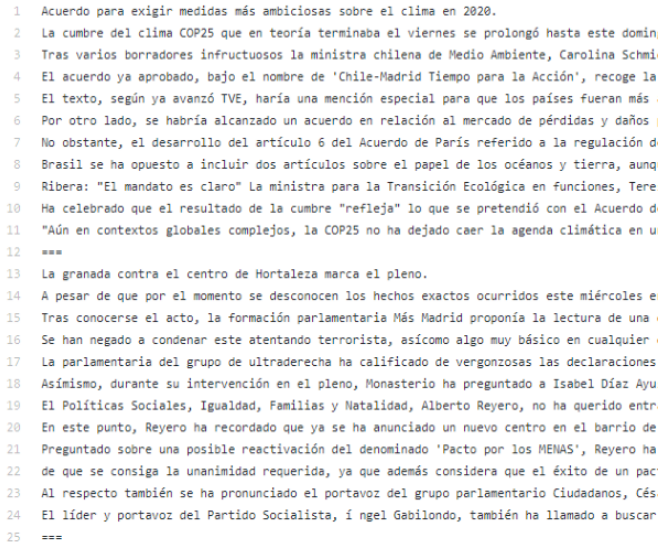

Figura 2: Mismo archivo, tras el script de limpieza

El segundo script, *generate_documents.py*, tiene como objetivo separar los archivos descargados en las diferentes noticias que contienen, generando un documento individual para cada noticia.

2.2. Sets de entrenamiento y test

Al descargar todos los documentos desde Social Clipping (esto es, todas las noticias disponibles en español para cada temática en los últimos dos meses), se han obtenido diferentes cantidades de noticias para cada una: 73 noticias de política, 64 de salud y 67 de tecnología.

Para el set de entrenamiento se han escogido las 40 primeras noticias de cada categoría, dejando para el dataset de testing las restantes.

Posteriormente se han descargado mas noticias, que sumadas a las noticias de testing llegan a un total de 296, para ser usadas con el segundo clasificador, que ordena los documentos en carpetas. Sin embargo se ha optado por usar estos documentos sólo en el segundo clasificador, y no en el dataset de testing del primer clasificador porque, aunque se intentó verificar que ningún documento coincidía con los ya existentes, el comparador no es infalible y algunos de los documentos pueden encontrarse duplicados, lo que podría afectar a la validez de los resultados de los tests al repetir ejemplos del dataset de entrenamiento.

2.3. Construcción del dataset

Sklearn (*"Scikit-learn"*, [4]) es una librería de Python enfocada al aprendizaje automático. Por tanto, una de las tareas principales para las que está equipada es la clasificación supervisada, varias de cuyas funciones se irán utilizando a lo largo del presente trabajo.

Entre muchas otras, Sklearn cuenta con una funcionalidad que permite generar la estructura de los datasets, preparándolos para ser utilizados por el resto de herramientas de la librería.

La funcionalidad está implementada en el método [5]: *sklearn.datasets.load_files*, y permite cargar una carpeta de ficheros de texto, agrupados en sus categorías o clases según el nombre de la subcarpeta en la que se encuentren.

En este caso, los ficheros con los que se trabaja son los documentos individuales para cada noticia que se han generado en la sección 2.1, y cuya categoría es la temática a la que pertenecen. Para usar esta funcionalidad una vez ordenados los documentos, sólo es necesario indicar como parámetro el directorio donde se encuentran, como se muestra a continuación:

```
def load_dataset(type):
    container_path = "documents/" + type + "_dataset"
    dataset = load_files(container_path, shuffle=True, encoding="utf-8")
    num_examples = len(dataset.data)
    print("Loaded " + type + " dataset with " + str(num_examples) + " examples")
    return dataset
```

Figura 3: Código en el que se cargan los datasets

Teniendo en cuenta que:

- El parámetro *type* indica si se desea cargar el dataset de entrenamiento ("train") o de prueba ("test").
- *container_path* guarda la dirección del directorio principal, el cual contiene a su vez una carpeta por cada temática de los documentos.
- Con el método de Sklearn *load_files*, se procede a la mezcla automática y carga de los distintos documentos en el dataset elegido, indicando también que se sigue la codificación UTF-8 para evitar la reaparición de errores en caracteres especiales, como se describió en la sección 2.1.
- Finalmente, se imprime por pantalla el número de elementos (documentos) que se han cargado en el dataset.

3. Entrenamiento del sistema

Como se ha mencionado anteriormente, la librería Sklearn también facilita en gran medida el trabajo realizado en esta sección, a la hora de implementar el clasificador.

El clasificador funciona siguiendo tres pasos principales, los cuales se describen a continuación.

3.1. Generación de la matriz de términos

El primer paso a la hora de trabajar con los datasets generados es construir la matriz de términos correspondiente.

También denominada "matriz término-documento", la matriz de términos contiene la frecuencia de aparición de cada uno de los términos dentro cada uno de los documentos. Esto es, para el conjunto de documentos del dataset, primero se

extraen todas las palabras distintas que aparecen en ellos (filas de la matriz), y en cada columna se indicará el número de veces que ese término aparece dentro de cada uno de los documentos.

Para visualizar cómo quedaría representado cada uno de los documentos o los términos dentro de la matriz, la figura 4 muestra un ejemplo:

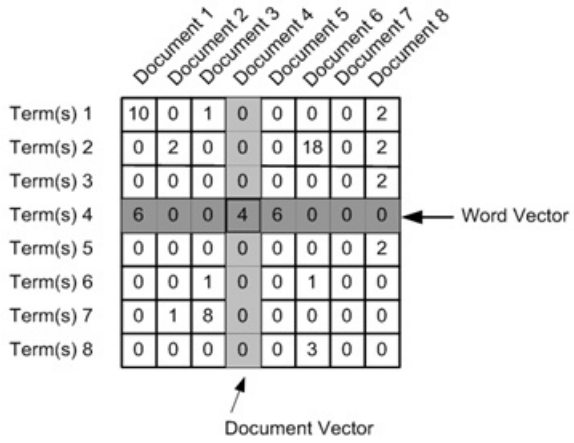

Figura 4: Estructura de una matriz de términos (de [6])

En Sklearn, este paso puede realizarse de manera automática por medio de la clase *CountVectorizer()* [7], la cual, recibiendo como entrada un dataset cualquiera, devuelve la correspondiente matriz de términos.

Si no se especifica un diccionario concreto o alguna selección de términos previa, la matriz devuelta tendrá tantas filas como el tamaño completo del vocabulario extraído del dataset, y tantas columnas como documentos haya en él. Por lo tanto, será una matriz notablemente dispersa.

Con el objetivo de mejorar u optimizar la información contenida en esta matriz, se han realizado varias tareas. La primera ha sido identificar y filtrar algunas palabras vacías o "stopwords". La selección de los términos que deben entrar o no en la lista de stopwords no es una tarea trivial pues, dependiendo del contexto, algunas de las palabras comúnmente filtradas pueden ser útiles para la clasificación. Por ello, la técnica elegida para seleccionarlas se ha basado en encontrar aquellas que aparecen con muy alta frecuencia en todas las clases de documentos, puesto que lo que se busca inicialmente son aquellas palabras que permitan distinguir las diferentes temáticas entre sí.

Esto se lleva a cabo a través de la propia clase *CountVectorizer()*, la cual tiene

como atributo el set de stopwords encontradas según esa característica.

Una vez realizado este filtrado, el resto tareas llevadas a cabo para intentar seguir mejorando la matriz de términos se describen en la próxima sección.

3.2. Refinamiento y mejora los datos de la matriz

En este punto del proceso, se realiza una serie de transformaciones no sobre las dimensiones de la matriz, sino sobre los datos que contiene, con la intención de incrementar la utilidad o valor de la información disponible para resolver el problema u objetivo planteado [8, 9].

Se parte de los valores ya presentes en la matriz, que indican la **frecuencia de aparición** ("Term Frequency", TF) de cada uno de los términos en los distintos documentos del dataset.

Se observó que algunos términos aparecen de forma bastante frecuente en todos los documentos, independientemente de su temática. Así pues, esta información no resultará útil para clasificarlos y generará ruido. Por ello, interesa reducir la importancia de esos términos en la matriz, lo que se tuvo en cuenta mediante el cálculo de su **frecuencia de documento inversa** ("Inverse Document Frequency", IDF). Se llevó a cabo aplicando la fórmula mostrada en la figura 5, en la que t es el término cuya IDF se está calculando, n representa el número total de documentos en el dataset, y *df(t)* es el número de documentos en los que aparece el término t.

$$\text{idf}(t) = \log \frac{n}{\text{df}(t)} + 1$$

Figura 5: Cálculo de la IDF para un término t cualquiera

La constante 1 se añade para que aquellos términos que aparecen en todos los documentos del dataset no sean completamente ignorados, sino que su frecuencia de aparición siga teniéndose en cuenta.

Estos dos valores, TF e IDF, fueron combinados tal y como se indica en la figura 6, siendo t un término cualquiera, presente en el documento d:

$$\text{tf-idf(t,d)} = \text{tf(t,d)} \times \text{idf(t)}$$

Figura 6: Cálculo del valor de TF-IDF

En el caso de aquellos documentos de mayor longitud, al contener una cantidad superior de palabras en total, presentarán para muchos términos una frecuencia de aparición (TF) más alta que en documentos más cortos. Este sesgo afectará a los valores obtenidos de TF-IDF, y no es beneficioso a la hora de ejecutar la clasificación. Una solución comúnmente usada frente a este problema es normalizar estos valores, para lo que se ha aplicado la norma Euclídea sobre cada uno de los vectores de término (que como se explicó mediante la figura 4), se corresponden con las filas de la matriz.

$$v_{norm} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{{v_1}^2+{v_2}^2+\cdots+{v_n}^2}}$$

Figura 7: Fórmula de la norma Euclídea, aplicada a cada vector de término, v

Estas transformaciones sobre los valores de la matriz de términos se pueden aplicar fácilmente gracias a la clase *TfidfTransformer()* [10] de Sklearn, la cual recibe como entrada la matriz de términos creada en el paso anterior y detalles sobre las transformaciones que se desea realizar, y devuelve como resultado la matriz modificada.

3.3. Entrenamiento del clasificador

El último paso a seguir en esta fase del proceso, ahora que ya están listos los datos de entrada, es entrenar el clasificador. Sklearn permite crear clasificadores de varios tipos. En este caso se ha escogido probar con dos métodos de aprendizaje: Naive Bayes y Support Vector Machines (SVM).

Naive Bayes

Los métodos basados en Naive Bayes, como se explica en [11], son un conjunto de algoritmos de aprendizaje supervisado que parten de la aplicación del teorema de Bayes asumiendo que todos los términos son independientes entre sí, dado un valor para la variable clase (en este caso, la temática del documento).

El teorema de Bayes asegura que, dada una clase y y un vector de términos $\{x_1, \ldots, x_n\}$, la probabilidad de que el documento del que provienen pertenezca a esa clase es la siguiente: $P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1,\ldots,x_n|y)}{P(x_1,\ldots,x_n)}$

Con la mencionada asunción de independencia, también se tiene que la probabilidad de aparición de un término cualquiera x_i condicionada a la clase de su

documento (y) no depende de la presencia del resto de términos que aparecen en él. O lo que es lo mismo: $P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y)$

Por tanto, la probabilidad de aparición de un conjunto de términos será equivalente al producto de la probabilidad de aparición de cada uno, pudiendo expresarse la fórmula inicial de la siguiente manera: $P(y \mid x_1, \ldots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1,\ldots,x_n)}$

Finalmente, teniendo en cuenta que la probabilidad de aparición de un conjunto de términos concreto es constante (pues será determinada por el contenido de los documentos del dataset de entrenamiento que se pasen al clasificador), se puede extraer de la fórmula, obteniendo finalmente la regla de clasificación utilizada por los sistemas basados en Naive Bayes:

$$P(y \mid x_1, \ldots, x_n) \propto P(y)\prod_{i=1}^{n} P(x_i \mid y) \Rightarrow \hat{y} = \arg\max_y P(y)\prod_{i=1}^{n} P(x_i \mid y)$$

Esto, dicho de otra manera, significa que la probabilidad de que el documento del que proviene el vector de términos $\{x_1, \ldots, x_n\}$ pertenezca a la clase y es proporcional al producto de:

- $P(y)$, la frecuencia relativa de la clase y en los documentos del dataset de entrenamiento.
- $\prod_{i=1}^{n} P(x_i \mid y)$, siendo $P(x_i \mid y)$ la probabilidad de que cada término x_i ocurra en un documento de la clase y, lo que puede ser interpretado como una medida de "cuánta evidencia contribuye ese término a la afirmación de que y es la clase correcta" [12].

Y por tanto, la clase que un clasificador de este tipo escogerá ($\hat{y}$, temática predicha) es aquella para la cual la probabilidad resultante sea más alta.

De entre los distintos métodos de aprendizaje basados en Naive Bayes, el utilizado en esta práctica es el **Multinomial Naive Bayes**, pues está enfocado a su aplicación sobre datos con distribución multinomial, la cual es una generalización, para k posibles valores, de la distribución binomial [13]. La clase *MultinomialNB* [14] es la que se utiliza en Sklearn para crear una instancia de un clasificador de este tipo.

Support Vector Machines

Support Vector Machines (SVM) son otro tipo de métodos de aprendizaje automático, esta vez basados en espacios vectoriales. Los diferentes documentos o puntos del dataset son representados mediante vectores en ese espacio, y la frontera (o hiperplano, al trabajar en un espacio multidimensional) que delimitará las diferentes

clases será aquella con máxima distancia a los puntos que estén más cerca de ella [15]. En otras palabras, se busca la zona de máxima separación posible entre cada par de clases.

Quedará de esta manera el espacio separado en secciones lo más amplias posible, que idealmente contendrán en su interior a los elementos de la clase correspondiente (con la posible excepción de unos pocos, que serán tomados como outliers o ruido).

En la siguiente imagen se muestra dónde estaría la frontera o hiperplano (línea discontinua) que separa dos clases, equidistante a los puntos de ambas que están más cercanos a la frontera, marcados con la línea continua.

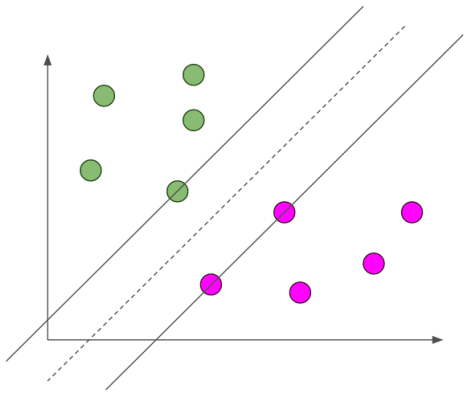

Figura 8: Ejemplo de separación de dos clases mediante SVM [16]

Los vectores pertenecientes a aquellos puntos, de dos clases diferentes, más cercanos al hiperplano que las separa, son los denominados *vectores de soporte*, y dan nombre a estos métodos.

Aunque estas ideas básicas permanecen, existen métodos muy distintos dentro del conjunto de los SVM a la hora de resolver problemas de clasificación multiclase. Ejemplo de ello son los mostrados en la figura siguiente, que representa dos posibles soluciones de clasificación de un dataset de tres especies de flores según las características de sus sépalos [11]. Se observa que la frontera no tiene por qué ser siempre lineal (figura 9); pueden utilizarse funciones de decisión no lineales (como son las polinómicas, figura 10) para calcularla.

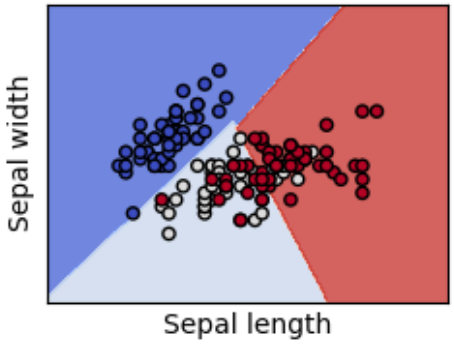

Figura 9: Utilizando un método SVM lineal

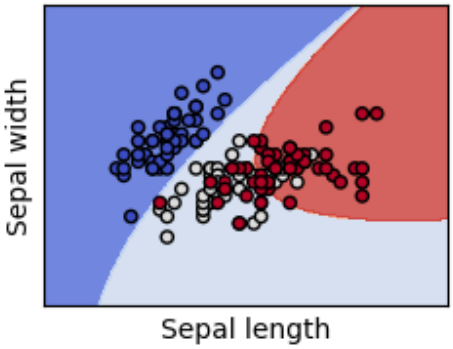

Figura 10: Utilizando un método SVM polinómico (grado 3)

En cuanto al uso de SVM para la clasificación [11], algunas de sus ventajas principales son:

- Son métodos efectivos en espacios de altas dimensiones, incluidos aquellos casos en los cuales el número de dimensiones es superior al de muestras.
- Eficientes en el uso de memoria, pues sólo utilizan un subconjunto de los elementos de entrenamiento, los vectores de soporte, para la función de decisión.
- Poder elegir distintos tipos de función para el cálculo de la frontera (e incluso introducir funciones propias, personalizadas para el problema por el usuario), hace de ellos métodos métodos muy versátiles.

Por otro lado, algunas de sus desventajas destacables son:

- Para los casos en los que el número de características es notablemente mayor al de muestras, es muy importante evitar el sobreentrenamiento al elegir las funciones de decisión.
- Los métodos SVM no devuelven directamente una estimación de la probabilidad, como pasa en otros métodos como Naive Bayes. Las estimaciones de probabilidad se tienen que calcular mediante un computacionalmente costoso paso de validación cruzada.

Entre las múltiples opciones de aplicación de SVM, en el presente trabajo se ha elegido utilizar un clasificador SVM lineal con aprendizaje por SGD (*Stochastic Gradient Descent*, Descenso de Gradiente Estocástico) [17]. En esta técnica, el gradiente de la pérdida se estima para las muestras de una en una, y el modelo se va actualizando, con una tasa de aprendizaje que va siendo reducida progresivamente.

Una de las ventajas de utilizar este tipo de aprendizaje es que permite trabajar por minibatches, con lo que se aumenta significativamente su eficiencia.

Para instanciar este clasificador se ha utilizado la clase *SGDClassifier* [17] de Sklearn.

En ambos casos (SVM y Naive Bayes), la instancia del clasificador, una vez creada, puede ser entrenada mediante el método *fit()*, el cual recibe como entrada la matriz de términos obtenida en los pasos anteriores (sección 3.2) y un vector que indica la temática de cada uno de los documentos.

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

3.4. Pipeline

En la implementación presentada en este trabajo, se ha optado por encadenar los pasos que se han descrito en las 3 subsecciones anteriores, utilizando la herramienta *Pipeline* [18] de Sklearn, la cual permite enlazar la salida de *CountVectorizer* con la entrada de *TfidfTransformer*, y la salida de éste con la entrada para el clasificador, obteniendo un proceso único que recibe como entrada inicial el dataset de entrenamiento y devuelve el clasificador una vez entrenado.

Esto se ha aplicado para ambos tipos de clasificador (Naive Bayes y SVM), en cada una de las pruebas realizadas para la evaluación de su funcionamiento, detalladas en la sección 4.2, y cuyo efecto en los resultados obtenidos se presentará a continuación.

4. Evaluación del rendimiento

Una vez listo el clasificador, se ha procedido a comprobar su correcto funcionamiento y evaluar su rendimiento.

4.1. Funcionamiento del clasificador

El clasificador implementado a lo largo de la sección anterior cuenta con la función *predict*, la cual, recibiendo como entrada un dataset compuesto de documentos a clasificar, devuelve como salida la clase (en este caso, la temática) predicha por el clasificador para cada uno de esos documentos.

Para comprobar el funcionamiento del clasificador, se ha utilizado el dataset de testing. Como se mencionó en la sección 2.2, se ha entrenado el clasificador con 40 ejemplos de cada temática, dejado el resto de ejemplos para el dataset de testing,

el cual estará compuesto, por tanto, de 33 noticias de política, 24 de salud y 27 de tecnología.

Utilizando la función *predict* con el dataset de testing se obtiene la clasificación predicha para cada uno de los documentos.

predicted = classifier.predict(test_data.data)

Una vez generada esa predicción, se ha contrastado la clasificación predicha con la real y se ha calculado la media de aciertos. Para este paso, se ha utilizado la librería *numpy*.

mean_prediction = np.mean(predicted == test_data.target)

- *predicted* es un array que contiene la clase predicha para cada uno de los documentos.
- *test_data* es el dataset de testing.
- *test_data.target* es el array que indica la temática real de cada uno de los documentos de ese dataset.

4.2. Pruebas realizadas

El funcionamiento de ambos tipos de clasificadores implementados ha sido probado bajo tres condiciones diferentes, las cuales tienen que ver con los posibles métodos de extracción de términos a partir de los documentos de los datasets.

Por defecto, según la implementación presentada, la selección de los términos de los documentos durante el paso de generación de la matriz (sección 3.1) se hace tomándolos palabra por palabra (esto es, extrayendo **unigramas**). Por tanto, de un documento se obtendrán tantos términos como palabras contenga, a excepción de las stopwords, que se descartarán.

Pero en muchos idiomas, entre ellos el español, es frecuente la presencia de expresiones o frases hechas. Casos en los que el orden de las palabras puede aportar una información adicional al significado de las palabras en sí. Con esta idea en mente, se decidió hacer una segunda prueba extrayendo de los documentos no solo unigramas, sino también **bigramas**, términos compuestos por dos palabras. Esto se indica en la implementación pasando el parámetro $ngram\_range = (1, 2)$ a la función *CountVectorizer()* que genera la matriz de términos.

Esto lleva a que el número de términos extraído de cada texto sea bastante superior al primer caso. Por ejemplo, en la oración “La casa es roja.”, si se extrajeran

únicamente unigramas se obtendrían 4 términos: {"La", "casa", "es", "roja"}. Pero extrayendo unigramas y bigramas, se obtendrían 3 términos adicionales: {"La casa", "casa es", "es roja"}. Como se explicará mas adelante, este aumento de la cantidad de términos tendrá un impacto significativo sobre los resultados obtenidos en esta prueba.

Finalmente, se decidió llevar a cabo una tercera prueba en la que se construye el vocabulario de términos que se utilizarán en la matriz, no mediante las herramientas de Sklearn como en las dos pruebas previas, sino mediante un recurso independiente y especializado: la aplicación **TesaurVAI** [19, 20]. Es una herramienta enfocada al análisis de texto y la creación y gestión de tesauros.

Gracias a ella se ha podido extraer, a partir de los documentos del dataset, un glosario con los términos más frecuentes y/o significativos para cada temática. Esta información se ha aplicado a la hora de generar la matriz de términos, mediante un parámetro pasado a la función *CountVectorizer()* que contiene la lista de términos del glosario, para trabajar solo con ellos.

4.3. Resultados obtenidos

Se muestran a continuación los resultados obtenidos con ambos clasificadores para las tres pruebas realizadas. Se incluye la media de aciertos (expresada como valor $\in [0, 1]$), tanto en el dataset completo como por temática de documentos.

Las conclusiones extraídas en base a los resultados presentados se discutirán en la sección 6.

Para el caso de extracción de unigramas, los resultados obtenidos han sido los siguientes:

Dataset:	**Completo**	**Política**	**Salud**	**Tecnología**
Naive Bayes	0.9205	0.9697	0.8571	0.9259
SVM	0.9432	0.9697	0.8929	0.9630

En la segunda prueba, en la que se trabaja tanto con unigramas como con bigramas, se han devuelto los siguientes resultados:

Dataset:	**Completo**	**Política**	**Salud**	**Tecnología**
Naive Bayes	0.9205	0.9697	0.8571	0.9259
SVM	0.9205	0.9394	0.8571	0.9630

Finalmente, en la tercera y última prueba, en la que se ha introducido el glosario extraído gracias a TesaurVAI, los resultados de la clasificación han sido los siguientes:

Dataset:	**Completo**	**Política**	**Salud**	**Tecnología**
Naive Bayes	0.9091	0.9394	0.8571	0.9259
SVM	0.9091	0.9394	0.8571	0.9259

5. Clasificación en carpetas

Como ultimo paso, se ha creado un segundo sistema clasificador, que ordena los archivos situados en la carpeta indicada en otra carpeta, dentro de la sub-carpeta correspondiente a la categoría que el clasificador predice que corresponde al documento.

Para esta implementación se ha optado por crear un script independiente, que utiliza un clasificador SVM implementado del mismo modo que en los pasos anteriores.

Este clasificador ha dado muy buenos resultados, colocando incorrectamente solo 2 documentos de política, 2 de tecnología y 4 de salud, de un total de 296 documentos.

Para una explicación del uso de este clasificador, consultar el anexo A: Instrucciones de ejecución.

6. Discusión & Conclusiones

Para finalizar este trabajo, se incluyen en esta sección las observaciones más relevantes extraídas de la realización de la práctica y de los resultados obtenidos de ella.

En primer lugar, el problema de clasificación planteado ha sido resuelto satisfactoriamente, como puede verse en las tablas de la sección anterior, en la que la menor media de aciertos conseguida en cualquiera de las pruebas ha sido del 85,71 %.

En segundo lugar, se pasa a realizar una comparativa del rendimiento de los clasificadores en las diferentes pruebas realizadas.

En el trabajo con **unigramas** se han obtenido los mejores resultados de las tres pruebas, y tanto en el dataset completo como en las temáticas de salud y tecnología, el clasificador SVM ha mostrado un rendimiento superior al basado en Naive Bayes. En la temática restante, política, ambos métodos han obtenido la misma puntuación.

En el caso de extracción de **unigramas y bigramas**, parece que el beneficio que podía reportar el hecho de conservar el orden de algunas palabras, como expresiones o frases hechas que puedan ayudar a la clasificación, se ha visto eclipsado por el perjuicio de aumentar de tal manera la cantidad de términos con los que debe trabajar el clasificador. Por tanto, se han obtenido resultados peores que en el caso de utilizar sólo unigramas. Comparando ambos clasificadores, esta vez Naive Bayes sí que ha conseguido superar al rendimiento de SVM en la temática de política, pero SVM queda por encima en tecnología.

Por último, al utilizar el glosario generado con **TesaurVAI**, se puede ver en la tabla 3 que se ha obtenido un rendimiento inferior a los casos anteriores. Esto ha sido un resultado inesperado, puesto que la herramienta está especializada en la tarea de extracción de los términos relevantes para el vocabulario con el que se trabaja, con lo que los resultados deberían mejorar. Por tanto, este rendimiento inferior debe provenir de una selección inadecuada por parte de los propios alumnos, autores de la práctica, de los términos del glosario, al descartar accidentalmente términos de las diferentes temáticas que se han juzgado como no relevantes, pero realmente eran muy útiles para la distinción entre las temáticas tratadas. Aun así, los valores de rendimiento obtenidos siguen siendo muy satisfactorios, pues casi todos los casos superan el 90 % de acierto.

En conclusión, se ha implementado correctamente un sistema de clasificación de documentos, capaz de distinguir, automáticamente y con un alto grado de éxito, entre noticias de las temáticas política, salud y tecnología.

A. Instrucciones de ejecución

A continuación se detallan las instrucciones para ejecutar los scripts proporcionados con la practica.

A.1. Clasificador 1: classifier1.py

Para ejecutar: *python3 classifier1.py*

Este clasificador carga los datasets de entrenamiento y testing utilizando los documentos que se encuentran en las carpetas especificadas en el fichero de configuración. Luego procede a entrenar y probar distintos clasificadores con el dataset de testing. Por ultimo muestra los resultados por pantalla.

A.2. Clasificador 2: classifier2.py

Para ejecutar: *python3 classifier2.py*

Este clasificador lee los documentos que se encuentran en la carpeta también especificada en el fichero de configuración, y los ordena en las categorías que predice, en sub-carpetas dentro del directorio especificado.

El resultado de la ejecución del clasificador se puede ver en la carpeta *documents/classified* en la que se han clasificado correctamente 290 de 296 documentos.

Nota: Como se explica en la seción 2.2, algunos de los documentos descargados para clasificar con el segundo clasificador pueden encontrarse duplicados, ya que se hizo una segunda descarga de documentos de social clipping y, aunque se intento verificar que ningún documento coincidía con los ya existentes, el comparador no es infalible.

A.3. Fichero de configuración: config.txt

En el fichero de configuración se especifican algunos parámetros que es necesario cambiar si se van a ejecutar los scripts en un sistema operativo distinto a linux.

- *Fila 1:* ruta donde se encuentran los documentos de entrenamiento para el clasificador 1.
- *Fila 2:* ruta para los documentos de prueba para el clasificador 1.
- *Fila 3:* ruta del glosario.
- *Fila 4:* True para usar el glosario, False para no usarlo.

- *Fila 5:* ruta donde se encuentran los documentos a ordenar por el clasificador 2
- *Fila 6:* ruta donde se guardan los documentos clasificados por el clasificador 2.
- *Fila 7:* carácter que separa los directorios en las rutas. Depende del sistema operativo.

A.4. Scripts de limpieza de documentos

Hay dos scripts que automatizan la preparación de los documentos obtenidos con Social Clipping. Las instrucciones para usar estos scripts se encuentran en el fichero README.md

Referencias

[1] Towards Data Science Medium. *Machine Learning, NLP: Text Classification using scikit-learn, python and NLTK*. 2017. URL: https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a (visitado 27-12-2019).

[2] Open source. *Scikit-learn: Machine Learning in Python*. 2019. URL: https://scikit-learn.org/stable/index.html (visitado 28-12-2019).

[3] DAIL Software. *Social Clipping*. 2019. URL: https://www.social-clipping.com/ (visitado 27-12-2019).

[4] F. Pedregosa y col. «Scikit-learn: Machine Learning in Python». En: *Journal of Machine Learning Research* 12 (2011), págs. 2825-2830.

[5] Machine Learning in Python scikit-learn. *sklearn.datasets.load_files*. 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_files.html (visitado 27-12-2019).

[6] Analytics Vidhya. *An Intuitive Understanding of Word Embeddings: From Count Vectors to Word2Vec*. 2019. URL: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/count-vector/ (visitado 28-12-2019).

[7] Scikit-learn. *CountVectorizer function*. 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer (visitado 28-12-2019).

[8] Prabhakar Raghavan Christopher D. Manning e Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[9] Ricardo Baeza-Yates y Berthier Ribeiro-Neto. *Modern information retrieval: The Concepts and Technology behind Search*. ACM press New York, 1999.

[10] Scikit-learn. *TfidfTransformer function*. 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html?highlight=tfidftransformer (visitado 28-12-2019).

[11] Machine Learning in Python scikit-learn. *Support Vector Machines*. 2019. URL: https://scikit-learn.org/stable/modules/svm.html (visitado 29-12-2019).

[12] Prabhakar Raghavan Christopher D. Manning e Hinrich Schütze. *Introduction to Information Retrieval*. Chapter 13: Text classification and Naive Bayes. Cambridge University Press, 2008.

[13] The Free Encyclopedia Wikipedia. *Mutinomial Distribution*. 2019. URL: https://en.wikipedia.org/wiki/Multinomial_distribution (visitado 30-12-2019).

[14] Machine Learning in Python scikit-learn. *MultinomialNB*. 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB (visitado 30-12-2019).

[15] Prabhakar Raghavan Christopher D. Manning e Hinrich Schütze. *Introduction to Information Retrieval*. Chapter 15: Support vector machines and machine learning on documents. Cambridge University Press, 2008.

[16] Python Machine Learning. *Classification with Support Vector Machines*. 2019. URL: https://pythonmachinelearning.pro/classification-with-support-vector-machines/ (visitado 30-12-2019).

[17] Python Machine Learning. *SGDClassifier*. 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html?highlight=sgdclassifier#sklearn.linear_model.SGDClassifier (visitado 31-12-2019).

[18] Scikit-learn. *Pipeline*. 2019. URL: https://scikit-learn.org/stable/modules/classes.html?highlight=pipeline#module-sklearn.pipeline (visitado 31-12-2019).

[19] DAIL Software. *TESAURVAI++*. 2019. URL: https://www.dail.es/shop/es/ (visitado 31-12-2019).

[20] Tendencias21. *Tesaurvai: un software lingüístico para analizar textos y crear tesauros*. 2019. URL: https://www.tendencias21.net/Tesaurvai-un-software-ling%5C%C3%5C%BCistico-para-analizar-textos-y-crear-tesauros_a40662.html (visitado 31-12-2019).