Universidad Politécnica de Madrid

Máster Universitario en Inteligencia Artificial

Clasificador Documental con Glosario

Asignatura: Ingeniería Lingüística

Autores:
Joel Pardo Ferrera
Daniel Carmona Pedrajas

Correos:
joel.pardof@alumnos.upm.es
daniel.carmonap@alumnos.upm.es

23 Diciembre 2022

Índice

1. Introducción

En esta sección presentaremos los objetivos de la práctica y la estructura de la memoria.

1.1. Objetivo de la práctica

El objetivo principal de la práctica es construir un sistema de **clasificación de textos**. Este problema consiste en asignar una clase o categoría a un documento dada una descripción del mismo en un cierto espacio documental, es decir, una codificación donde esté comprimida la información que contiene [1].

La clasificación de textos se puede abordar con métodos de aprendizaje supervisado y no supervisado. En nuestro caso utilizaremos el **aprendizaje supervisado** ya que nuestros textos están etiquetados por su categoría. En total, contaremos con 240 textos o noticias que se reparten de la siguiente manera:

- 60 noticias sobre **deportes**. Para obtener una representación completa de la categoría deportes hemos recogido:
  - 15 noticias sobre fútbol.
  - 15 noticias sobre baloncesto.
  - 15 noticias sobre tenis.
  - 15 noticias sobre Fórmula 1.
- 60 noticias sobre **salud**.
- 60 noticias sobre **ciencia**.
- 60 noticias sobre **política**

Para crear nuestro clasificador usaremos 30 noticias de cada caetgoría y posteriormente lo validaremos con la mitad restante.

1.2. Estructura

En esta memoria se encuentran documentados los pasos que hemos seguido para construir nuestro clasificador de textos:

- En la sección 2 explicamos las técnicas utilizadas para **obtener** nuestros textos
- En la sección 4 explicamos la **creación del glosario** para cada categoría con el que crearemos nuestro modelo.
- En la sección 5 explicamos los **modelos** utilizados para la clasificación de textos y analizamos los resultados obtenidos.
- En la sección 6 comentamos las **conclusiones** finales de la práctica.
- Por último en el apéndice encontraremos tablas e información adicional.

2. Obtención de textos

Lo primero que necesitamos para clasificar textos son textos, podríamos haber optado por recolectar una serie de textos de alguna base de datos de fácil acceso que tenga los documentos que buscamos pero hemos decidido suponer que no tenemos acceso a estas herramientas para simular un problema real y aprender otros métodos que pueden ser útiles en un futuro: **Web scraping** y **OCR**.

Como hemos mencionado anteriormente, el número de noticias recogidas es de 240 en total, 60 por categoría, obteniéndolas de 65 fuentes distintas (ver fuentes en apéndice A) para evitar sesgos.

2.1. Web scraping

Web scraping [2] es una técnica que permite recoger información de varias páginas web de forma automática explotando lenguajes de marcado que se pueden encontrar en la web. Los pasos que hemos seguido han sido los siguientes:

1. Seleccionamos una categoría de noticias en *Google News*, véase deportes.
2. Obtenemos una lista de links a noticias de deportes.
3. Utilizando la librería de python *request* obtenemos el html de cada link.
4. En cada html usamos la librería de python *BeautifulSoup* y recogemos todo el texto contenido en el tag <article> de ese mismo html.
5. Guardamos un archivo *.txt* con el texto de la noticia en el directorio correspondiente.

Hemos usado esta técnica para obtener noticias sobre **deportes** y **salud**. Cabe destacar que las noticias obtenidas con este método son todas correspondientes al mismo día, creemos que esto puede afectar al sesgo de los datos y sería interesante recoger noticias que cubriesen un espacio de tiempo mayor, incluso de años para obtener un modelo que generalice mejor, pero este no es el objeto de la práctica.

El proceso de Web scraping se describe con más detalle en el apéndice B.

2.2. OCR

Las siglas de esta técnica hacen referencia al reconocimiento óptico de caraácteres, de un determinado alfaéto, dentro de una imágen. Originalmente, se utiliza para digitalizar documentos de forma efectiva, tanto temporal como computacionalmente.

Uno de los problemas principales que muestra es el mal reconocimiento de los carácteres que localiza dentro de la imágen. Este hecho se produce sobretodo cuando la calidad de la misma no es muy buena, cuando aparecen manchas sobre el texto, o cuando son textos manuscritos a mano.

Para la implementaciń de la misma hemos seguido los siguientes pasos:

1. Abrimos desde el navegador un portal de noticias. Buscamos las categorías en cuestión.
2. Seleccionamos una notiícia y hacemos capturas de pantalla de la misma.
3. Almacenamos de forma ordenada las capturas de cada notícia correspondiente a una categoría.
4. Para cada notícia dentro de cada categoría, cargamos las capturas realizadas con la librería de python *PIL*.
5. Aplicamos el OCR a partir de la librería de python *pytesseract* especificando el lenguaje castellano.
6. Guardamos el resultado del OCR en un archivo en formato *.txt*.

Esta técnica únicamente la hemos empleado para obtener las noticias de las categorías **ciencia** y **política**. Como bien ocurre con la técnica explicada anteriormente, las noticias están fechadas en el mismo día, de recopilación de las mismas. Además de afectar al sesgo de los datos, el clasificador corre el riesgo de aprender el vocabulario contemporañeo desestimando así palabras que en un pasado no muy lejando podrián haber sido empleadas para la redacci ón de esa misma notícia.

La obtención de noticias mediante OCR se describe con más detalle en el apéndice C.

3. Preprocesamiento de datos

En esta sección presentamos cómo hemos normalizado nuestros documentos. En este punto, hemos obtenido 240 textos mediante Web Scraping y OCR pero su contenido está en crudo, es decir, contiene signos de puntuación, tildes, saltos de línea y otros caraceteres que no nos interesan a la hora de clasificar nuestro texto mediante un glosario. Aunque estos elementos no nos son útiles ahora, sí que podrían ser importantes si estuviesemos extrayendo conocimiento del texto, por ejemplo, la presencia de una interrogación puede preceder a una sección de texto con mucha información.

Además de esto eliminaremos las stop words y obtendremos el lema de la palabra.

Se puede ver toda una traza de ejecución del preprocesado en el notebook *Preprocessing.ipynb*.

3.1. Procesamiento de caracteres especiales

Como hemos comentado, vamos a prescindir de caracteres especiales así que primero quitamos los caracteres no ASCII, acentos y letras mayúsculas recurriendo a la librería de python *unicode* y eliminamos los signos de puntuación mediante *spaCy*. Un ejemplo de conversión sería el siguiente:

'El fútbol es un deporte.' $\longrightarrow$ 'el futbol es un deporte'

La razón por la que decidimos eliminar los acentos es porque son unos de los principales errores ortográficos que se cometen y podríamos obtener frecuencias de palabras incorrectas perdiendo así información.

3.2. Procesamiento de stop words

Gracias a la librería de python *spaCy* podemos usar un modelo del lenguaje español muy completo con el que podemos parsear, lematizar e incluso obtener análisis morfológicos de palabras. También cuenta con una lista de stop words de 521 palabras que podemos encontrar en el apéndice D.

Las stop words son palabras muy frecuentes de la lengua que no suelen aportar información útil sobre el documento al que pertenecen, es por esto que decidimos prescindir de ellas. De esta manera nuestra frase anterior se convertía en:

'el futbol es un deporte' $\longrightarrow$ 'futbol deporte'

3.3. Lematización

El último paso que damos para terminar de procesar nuestros textos es obtener el lema de los términos que no han sido filtrados hasta este punto. Lematizar un término consiste en encontrar el mayor representante de una palabra y sus derivaciones morfológicas. La librería *spaCy* cuenta con un modelo del lenguaje español llamado *es_core_news_lg* con el que podemos realizar esta tarea. Así tendríamos una conversión final como esta:

'chicago sufriendo' $\longrightarrow$ 'chicago sufrir'

Usando los lemas de los términos agrupamos todas las derivaciones morfológicas de una palabra en un mismo elemento, dando así más peso a este grupo de palabras que nos puede dar información sobre la categoría a la que pertenece un texto. Por ejemplo, si en un texto aparecen los términos 'marcó', 'marcaba' y 'marcaron' únicamente contaríamos una repetición por cada uno de ellos, por lo que la información se dispersa, si en cambio usamos su lema 'marcar' tenemos una frecuencia de tres, lo que nos ayuda a detectar que el texto al que pertenecen estos términos posiblemente pertenezca a la categoría de deportes.

3.4. División del conjunto de datos

Como sabemos estamos en el proceso de crear un clasificador de textos mediante aprendizaje supervisado, así que para crear nuestra herramienta vamos a utilizar un conjunto de entrenamiento y para validarlo un conjunto de validación por lo que en este momento vamos a dividir nuestras noticias en dos conjuntos:

- 120 noticias para crear nuestro clasificador guardadas en *noticias_train.csv*
- 120 noticias para validar nuestro clasificador guardadas en *noticias_test.csv*

En cada conjunto hemos introducido de forma aleatoria 30 noticias de cada categoría para mantener nuestros conjuntos balanceados.

4. Creación del glosario

En esta sección explicaremos cómo hemos conseguido nuestro glosario. Para detalles de la implementación consultar el notebook *Glosarios.ipynb* donde se muestra la traza de la última ejecución.

En primer lugar, debemos definir qué es un glosario. Pues bien, un *glosario* es un conjunto de términos representativos de un conjunto de documentos que permite agruparlos bajo una misma categoría. En nuestro caso, queremos obtener 4 glosarios distintos:

- Glosario de **deportes**.
- Glosario de **salud**.
- Glosario de **ciencia**.
- Glosario de **política**

En la sección 5 utilizaremos estos glosarios para crear nuestro clasificador.

4.1. Método de creación del glosario

Para crear nuestro glosario usaremos el método **tf-idf**. El esquema **tf-idf** [1] es una combinación de las métricas *term frequency* e *inverse document frequency*:

- Mediante **term frequency** obtenemos la frecuencia en la que el término t aparece en el documento d y lo denotamos como $tf_{t,d}$.
- Mediante **inverse term frequency** obtenemos una medida de cómo de frecuente es un término t en un conjunto de documentos de tamaño N y lo denotamos como idf_t.

Cabe destacar que idf_t se puede expresar de muchas maneras, en nuestro caso usaremos la expresión $log\frac{N}{df_t}$ dónde df_t es el número de documentos que contiene el término t.

Por lo tanto obtenemos la métrica **tf-idf**:

$$tf - idf_{t,d} = (1 + logtf_{t,d}) \times idf_t = (1 + logtf_{t,d}) \times log\frac{N}{df_t} \tag{1}$$

En otras palabras, $tf - idf_{t,d}$ asigna a cada término t un peso en un documento d tal que:

- El peso es alto cuando t aparece muchas veces en un número pequeño de documentos.
- El peso es medio cuando t aparece menos veces en un documento o aparece en muchos documentos.
- El peso es bajo cuando t aparece en muchos documentos.

4.2. Selección de términos para el glosario

Habiendo definido la métrica **tf-idf** de un término en un documento, seleccionamos los términos que pertenecen al glosario de un categoría de la siguiente manera:

1. Creamos un diccionario con todas las palabras de los documentos de la categoría en cuestión.
2. Calculamos el tf-idf de cada término en cada documentos y los sumamos obteniendo su peso tf-idf total.
3. Ordenamos los términos de mayor a menor según su peso tf-idf total.

4. Seleccionamos los 50 primeros términos.

Mostramos los primeros 5 términos de cada glosario en la tabla 3. Para comprobar los glosarios completos consultar el apéndice E.

deportes	**salud**	**ciencia**	**política**
seguidor	estigma	sal	silva
mbappe	atras	cola	rosell
correr	cancer	meteoro	vuelo
suarez	anemia	agujero	precio
resto	cafa	llama	torra
mans	aceite	gemanidas	reyes

Tabla 1: 5 primeros términos de cada glosario ordenados según peso tf-idf total

5. Clasificación

En esta sección mostraremos los distintos clasificadores y resultados obtenidos.

5.1. Clasificador mediante TF-IDF en Vector Space Model

Aprovechando que estamos representando nuestros documentos como vectores, vamos a utilizar el Vector Space Model que nos proporciona tf-idf para clasificar las noticias

Vector Space Model [1] es un espacio vectorial común a una serie de vectores que codifican documentos y que permite crear clusters de documentos, clasificar documentos y crear métricas para puntuar documentos en base a una query.

5.1.1. Elementos del Vector Space Model

Los elementos que necesitamos para representar nuestros documentos en forma vectorial son:

- *Diccionario*: un conjunto de términos que nos permitan representar todos los documentos en base a ellos. En nuestro caso el diccionario es la **unión de términos de todos los glosarios** obtenidos en la sección 4
- Un método de **vectorizar** nuestros documentos en base a nuestro diccionario. En nuestro caso usaremos el método **bag of words** ,proporcionado por la librería *gensim*, que para cada documento guarda un vector con la frecuencia de cada palabra del diccionario.
- **Modelo tf-idf** creado con los **bag of words** de los documentos para ahora poder expresarlo con su *tf-idf* que como hemos visto anteriormente contiene mejor la información del documento.

5.1.2. Proceso de clasificación

Para clasificar los documentos utilizaremos como referencia los glosarios de cada categoría.

Creamos un vector tf-idf de cada glosario y gracias a que estamos en un **Vector Space Model** [1] podemos compararlos con los vectores tf-idf de los documentos calculando su similaridad de similaridad del coseno:

$$sim(d1, d2) = \frac{d1 \cdot d2}{|d1||d2|} \tag{2}$$

Teniendo en cuenta esta métrica, calcularemos la similaridad entre el vector tf-idf de cada documento con los vectores de cada glosario y asumiremos que pertenece a la categoría con cuyo glosario tenga mayor similitud. Además de esto, guardamos los documentos en carpetas según la categoría asignada y ordenados de mayor a menor similitud con el glosario de la misma.

5.1.3. Resultados

De 120 documentos, hemos conseguido clasificar 111, los 9 no clasificados son documentos que han obtenido una similaridad igual con todas las categorías o su representación en el modelo es un vector de 0s. Los resultados que obtenemos con respecto a los documentos clasificados son los siguientes:

Y la matriz de confusión:

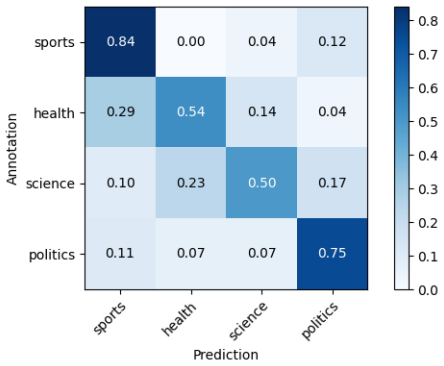

Figura 1: Matriz de confusión de nuestro clasificador

Hemos obtenido un modelo que no es demasiado bueno, como vemos en la matriz de confusión, las categorías con las que más problema tiene son con *health* y *science*, ambas tienen solapamiento semántico entre ellas porque la medicina pertenece al campo de la salud y la ciencia simultáneamente. Ocurre lo mismo con *health* y *sports*.

Ya habíamos comentado que nuestras noticias pertenecen a un periodo temporal muy pequeño y esto puede haber provocado que se centren simplemente en los temas de actualidad de ese momento, habiendo obtenido así un glosario sesgado y que no represente correctamente las características de cada una de las categorías, lo que podría explicar por qué ha habido documentos que no se han conseguido clasificar. Es por todo lo anterior que concluímos que para mejorar nuestros resultados con este modelo tendríamos que obtener un mejor glosario. De todas formas, decidimos probar con otro modelo para intentar mejorar nuestros resultados.

Para más detalle sobre la clasificación mediante tf-idf y resultados consultar el apéndice F y el notebook *Clasificacion-TF-IDF.ipynb* donde se encuentra la traza de la última ejecución realizada.

5.2. Otros clasificadores

Vamos a hablar de otros calsificadores como la regresión lógistica, el SVM lineal, SVM con kernel gausiano y Naive Bayes. Todos ellos implementados en la librería *Scikit-Learn*.

Para la tokenización de palabras empleamos el modelo *es_core_news_lg* de la librería *Spacy*. Este es un pipeline para invocar a un modelo optimizado para la CPU.

Hemos diseñado un proceso secuencial donde:

1. Probamos los clasificadores con las diferentes mutaciones entre los datos.
2. Escogemos los tipos de mutaciones de los datos que mejor rendimiento han obtneido junto con los los cuatro clasificadores que han mostrado mejor rendimiento y variamos sus parámentros.
3. Escogemos los tres mejores modelos con los respectivos parámetros que mejor rendimiento aportan y los comparamos con la matriz de confusión.

5.2.1. Entradas

A partir vocabulario recogido para el modelo de la subseción anterior, generamos una bolsa de palabras. Invocamos tambien un modelo TF-IDF distinto.

Ahora, mediante la interacción de estos componentes junto con los corpus originales, vamos a generar mutaciones de las entradas. Finalmente, quedan cuatro tipos de entradas:

- Bolsa de palabras (BoW) a partir de los corpus originales.
- Caracteristicas TF-IDF a partir de la bolsa de palabras.
- Promediado de los word embeddings de todos los corpus correspondientes a cada documento a partir de dichos corpus originales.
- Caracteristicas TF-IDF con los word embeddings promediados a partir de la bolsa de palabras, de las caracteristicas TF-IDF y de los corpus originales.

El que mejor funciona es el tercero; promediado de los word embeddings de todos los corpus correspondientes a cada documento a partir de dichos corpus originales.

5.2.2. Entrenamiento, evaluación y producción

El proceso de aprendizaje del modelo se lleca de forma secuencial; prinero el entrenamiento para enseñarle al modelo las muestras de cada una de las clases, evaluación del entrenamiento para ver como de bien el modelo es capaz de clasificar y por último la producción donde empleamos el modelo para predecir muestras nuevas que nunca ha visto.

Como método de evaluación nos basamos tanto en métricas como en la matriz de confusión. Para recordar voy a hacer una sintésis de estos dos elementos.

La matriz de confusión es una matriz que permite la visualización directa del rendimiento de un algoritmo. Cada fila como cada columna representa una de las clases. La diferencia entre las filas y columnas es que las primeras son la clase verdadera mientras la segunda es la clase predicha.

Por ejemplo, tenemos un documento de la categoría deportes y lo clasificamos. El modelo puede predecir que ese documento es de deportes o es de cualquier otra categoría. El modelo clasifica dicho documento como deportes, por lo tanto ha acertado. El modelo clasifica dicha notícia como ciencia, por lo tanto ha fallado. Estamos, en definitiva, viendo el error del clasificador.

Las métricas númericas diseñadas para evaluar el rendimiento del modelo. Están muy cercanas a la matriz de confusión. Las métricas que empleamos son las comúnes:

- Exactitud (Accuracy)
- Precision
- Recall (Exhaustividad)
- F1 score

5.2.3. Regresión logística

Es un tipo de análisis donde la variable objetivo, es decir, la variable a predecir es una variable categórica a partir de unas variables predictoras (características). Una variable categórica es una variable que puede adoptar un número limitado de estados (categorías).

Este modelo es de los mejores en todas las iteraciones que hemos hecho. Ha llegado, en muchas ocasiones a la última fase y en varias, ha sido seleccionado al mejor modelo. En la segunda fase variamos el *solver*, es decir, el algoritmo que utiliza para optimizar el resultado para el problema dado. El mejor valor de este parámetro es *liblinear* y *newton-cholesky*.

5.2.4. Naive Bayes

Este clasificador entra dentro de la familia de clasificadores probabilísticos. Se basa en el teorema de Bayes con cierta independencia (naive) entre las características.

El rendimiento de este modelo es bastante bueno. En la segunda aproximación cuando variamos el parámetro que modifica la varianza para calcular la estabilidad *var_smoothing*, no provoca cambios en el rendimiento. En ninguna de las iteraciones que hemos hecho, ha llegado a la última fase seleccionado como uno de los mejores modelos.

5.2.5. SVM lineal

Entra dentro la familia de clasificadores lineales como la regresión logística o las máquinas de vectores de soporte con un descenso del gradiente estocástico como sería el algortimo de optimización SGD. En este caso, el gradiente es estimado para cada muestra en una época y el modelo se actualiza al final reduciendo la tasa de aprendizaje.

Sin duda, es de los que mejores resultados nos ha proporcionado. Ha llegado en todas las iteraciones a la última fase y ha sido en todas seleccionado de los mejores modelos. En la segunda fase modificamos la función de perdida que usa *loss*. Los mejores valores de este parámetro son *squared_hinge*, *modified_huber*, *perceptron* y *hinge*, es decir, casi todos los valores disponibles.

5.2.6. SVM con kernel gaussiano

Clasificador C-vector de soporte, es deicr, una máquina de vectores de soporte gaussiana. En este caso, separa espacios de puntos para todas las categorías y luego comienza a esbozar fronteras de separación entre los distintos conjuntos intentando maximizar la distancia entre ellos. El vector de soporte se realiza con el esquea uno a uno, es decir, clase contra clase.

Los resultados de este modelo no son buenos. No pasa de la segunda fase donde variamos el parámetro *gamma* que varia el coeficiente del kernel.

5.2.7. Resultados

El mejor modelo es son las SVM con kernel lineal mientras que el peor son las SVM con kernel gaussiano (no lineal).

Naive Bayes es bueno pero no lo suficiente. La regresión logística es el segundo mejor modelo. Introducimos los matrices de confusión de los tres primeros.

	Accuracy	**Recall**	**Precision**	**F1 score**
modified_huber - Linear SVM	88.47	86.7	88.47	88.4
liblinear - Logistic Regression	87.5	88.2	87.5	87.7
newton-cholesky - Logistic Regression	85.8	86.9	85.8	86.2
log - Linear SVM	85	87.7	85	85.1
hinge - Linear SVM	84.2	85.6	84.2	84.5
perceptron - Linear SVM	83.3	85.2	83.3	83.7

Tabla 2: Métricas de los mejores clasificadores en la tercera fase

Para más detalle sobre la clasificación mediante tf-idf y resultados consultar el apéndice G y el notebook *Clasificacion-Resto.ipynb* donde se encuentra las trazsa de la última ejecución realizada.

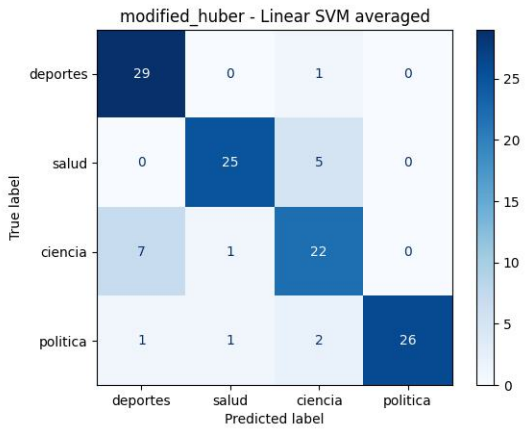

Figura 2: Matriz de confusión de modified_huber - Linear SVM

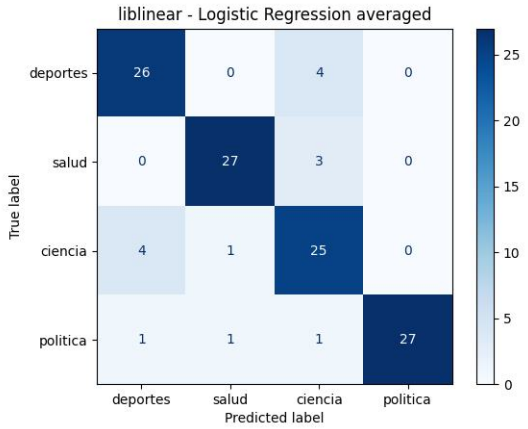

Figura 3: Matriz de confusión de liblinear - Logistic Regression

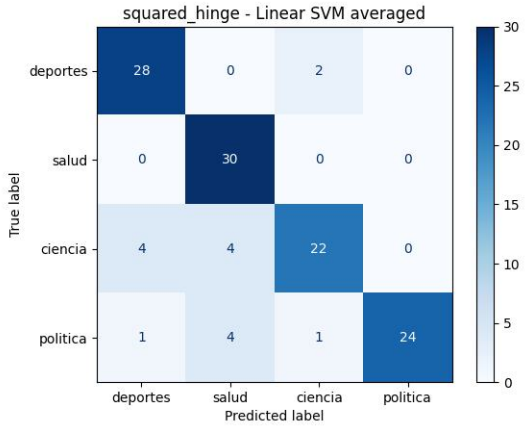

Figura 4: Matriz de confusión de squared_hinge - Linear SVM

6. Conclusiones

Hemos tocado desde la obtención hasta la obtención de resutlados de los clasificadores. Finalmente concluimos que:

- La obtención de noticias es más efectiva empleando la técnica de Web Scrapping. Sin embargo, para emplear el clasificador sobre noticias de fuentes analógicas, se podría utilizar de forma que agilizase el proceso de digitalización.
- En cuanto al preprocesado, hemos eliminado desde acentos hasta stop words. Hemos lematizado el texto y por último hemos dividido el cojunto de datos toal en conjuntos definidos de entrenamiento y prueba.
- Para crear el glosario y poder acceder a él de forma sencilla, el método TF-IDF. De esta forma, seleccionamos las más relevantes por categoría que formarían parte de este cojunto de palabras.
- En cuanto a la clasificación, hemos planteado cinco modelos diferentes desde TF-IDF en Vector Space Model hasta una Regresión logística. El modelo que mejor rendimiento presenta de todos en un SVM con kernel lineal. Sin embargo, cabe destacar el buen rendimiento de la regresión logística posicionándose así como segundo en el ranking.
- Para finalizar, hemos cumplido con el objetivo de este proyecto que sería el de implementar un clasificador de textos. Hemos planteado varios y analizado sus rendimientos tras haber recopilado todos los datos necesarios y formado un glosario por cada una de las categorías que pretendíamos diferenciar.

A. Fuentes de noticias

Las noticias recogidas vienen de 65 fuentes distintas:

- Antena 3 Noticias
- El Periódico
- El Confidencial
- ABC
- Okdiario
- Diario de Almería
- Geriatricarea.com
- Defensa Central
- laSexta
- Diario de Ávila
- Redacción Médica
- Planeta Triatlón
- elDiario.es
- MUI Kitchen
- Vista al Mar
- Periodismo.com
- ContraRéplica
- Newtral
- Car and Driver

- miarevista.es
- Fitness
- Cadena SER
- El Día de Valladolid
- Business Insider España
- ATP Tour
- Marca
- Medicina y Salud Publica
- ES
- AS
- 20minutos.es
- Faro de Vigo
- Superdeporte
- Eurosport ES
- Onda Cero
- Ser Padres
- El Debate
- El Imparcial de Oaxaca
- El Periódico de Extremadura
- Caracol Radio
- La Voz de Medina Digital
- Alimente
- NotiPress
- La Nueva España
- Murcia.com
- MSN
- asajajaen.com
- Málaga Hoy
- La Razón
- La Voz de Galicia
- MARCA.com
- MUNDO DEPORTIVO
- Vitónica
- Diario de Castilla y León
- INFORMACIÓN

- Diario de Cádiz
- Infosalus
- Bernabéu Digital
- Qué!
- Noticias de Gipuzkoa
- EL ESPAÑOL
- TRT Español
- EL PAÍS
- nbamaniacs
- Deia
- Sporting News
- El Independiente
- El Mundo

B. Web scraping en detalle

El código fuente que hemos utilizado para hacer Web scraping se encuentra en *google_news_url_scraping.py* y *text_scraping.py*.

Lo primero que hacemos es obtener los links y títulos de las noticias mediante el script *google_news_url_scraping.py*, ¿Cómo? Si accedemos a Google News y escogemos un tema o *topic* llegaremos a una página que tiene interfaz gráfica para el usuario, pero si modificamos la url y escribimos *rss* delante de *topic*, nos dirigiremomos a una página como esta:

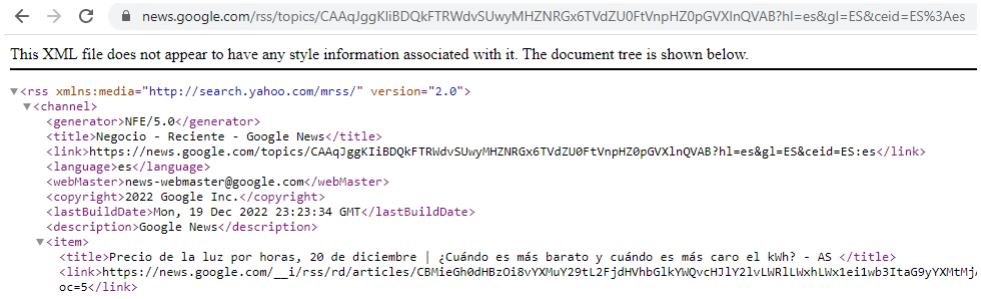

Figura 5: XML Google News

Es un archivo XML en el que cada <item> representa una noticia con su título y link que podemos parsear con la librería *BeautifulSoup* como se ve en el código 1.

```
1
2  # Obtenemos el archivo XML
3  response = requests.get(gnews_url)
4  # Parseamos el XML
5  soup = BeautifulSoup(response.text, "xml")
6  # Dividimos todos los items que contienen noticias
7  news = soup.find_all("item")
8  i = 0
9  retrieved = 0
10
11 # De cada item obtenemos el titulo y el link de la noticia
```

```
while retrieved < 15 and i < len(news)-1:
    title = news[i].find('title')
    link = news[i].find('link')
    retrieved += 1
    i += 1

```

Código 1: Uso de *BeautifulSoup* para scraping de links en Google News

Todos estos links se guardan en el dataframe *urls.csv* que tiene columnas de categoría, título y link con una fila por noticia.

Posteriormente con el script *text_scraping.py* obtenemos el html haciendo un request para cada link y obteniendo el cuerpo de la noticia accediendo al cuerpo de la noticia que se encuentra dentro del tag <article> como se muestra en el código 2.

```
# Obtenemos el archivo html
response = requests.get(link)
# Parseamos el archivo html
soup = BeautifulSoup(response.text, 'html.parser')
# Nos quedamos con el tag article del html
article = soup.find('article')

if article == None:
    print("Error, no se puede leer la noticia")
    return None
# Eliminamos figuras
for fig in article.select('figure'):
        fig.extract()
full_text = []
# Unimos todos los parrafos de article
for parragraph in article.find_all('p'):
    full_text.append(parragraph.text)
'\n'.join(full_text)

```

Código 2: Guardado de noticias en documentos .txt

Hemos de aclarar que ha funcionado para la mayoría de links, aunque para algunos de ellos hemos tenido que eliminar a mano ciertas palabras que no pertenecían al artículo en sí como pueden ser hipervínculos o títulos de noticias relacionadas o incluso tener que copiar el texto del artículo directamente en el archivo *.txt*.

Por último, guardamos la noticia en su carpeta correspondiente como un archivo *.txt* como se ve en el fragmento de código 3.

```
def save_text(category, text, count):
    file_name = f"{DATA_FOLDER}{category}/{category}_{count}.txt"
    if not exists(file_name):
        with open(file_name, "w") as f:
            if text is not None:
                f.write(text)

```

Código 3: Uso de *BeautifulSoup* para scraping de links en Google News

C. OCR en detalle

El código fuente que hemos empleado para hacer OCR se encuentra en *OCR.py*, *registros.py* y *Carpetas.py*.

Comenzamos definiendo el espacio de almacenamiento. Cada carpeta categoía va a tener una carpeta. Cada subcarpeta de cada categoría es un número del 1 al 60, ya que 60 es el número máximo de noticias por categoría. Dentro de estar subcarpetas alamcenamos las capturas de pantalla. Las imágenes con las que trabajamos están en formato *.png*.

Surge un problema a la hora de coger los titulos de las noticias. El OCR no distingue si el texto de la la imágen que está visualiza se trata de un título o del cuerpo de una notícia. Por lo tanto, creamos otra subcarpeta

para almacenar las capturas de las imágenes que contienen los titulos. En esta carpeta tendremos 60 imágenes donde cada una representará el titulo de la notícia a la que ahce referencia su nombre.

Por otra parte, para agrupar todos los documentos que extrae el OCR de una categoría concreta, hemos creado otra carpeta para este uso específico.

Empleamos la librería *os* para la creación de estos repositorios:

```
clases = ["Ciencia", "Politica"]

for z in clases:
    # Introducimos el nombre de la clase y donde queremos que se guarden los documentos '.txt'
    path = "../Documentos/" + z + "/"
    path_guardado = path + "txt"
    path_titulo = path + "titulo"

    carpetas = os.listdir(path)

    # Creamos los path de las carpetas (documentos)
    for i in range(1, 61):
        if str(i) not in carpetas:
            os.mkdir(path + str(i))

    if "txt" not in carpetas:
        os.mkdir(path_guardado)

    if "titulo" not in carpetas:
        os.mkdir(path_titulo)

```

Código 4: Uso de *os* para la creacion de los repositorios de almacenamiento

Vamos con la parte central de la cuestión a tratar en esta sección, el OCR. Su funcionamiento se basa en el recorrido de las subcarpetas de cada categoría creadas en el apartado anterior para obtener por cada una de ellas las capturas que forman una notícia y transformarla a un documento *.txt* almacenado en el directorio destinado a este fin. Podemos ver la forma en la que realiza esta tarea en el 5.

```
clases = ["Ciencia", "Politica"]

for z in clases:
    # Introducimos el nombre de la clase y donde queremos que se guarden los documentos '.txt'
    path = "../Documentos/" + z + "/"
    path_guardado = path + "txt/"
    path_titulo = path + "titulo/"


    # Listamos todos los documentos para esta clase
    carpetas = os.listdir(path)

    # Eliminamos la carpeta de guardado y '.DS_Store'
    carpetas = [c for c in carpetas if c != '.DS_Store' and c != 'txt' and c != 'titulo']


    # Creamos las rutas de cada uno de los documentos
    rutas = []
    for i in carpetas:
        rutas.append(path + i + "/")



    # Para los documentos y las rutas
    for c, j in zip(carpetas, rutas):

        # Sacamos las imagenes por cada documento
        imagenes = os.listdir(j)

        # Copiamos el archivo que contiene un titulo a la carpeta de titulos
        shutil.copy(j + "1.png", path_titulo + str(c)+ ".png")

```

```
        # Inicializamos lista a vacia. Sirve para guardar el texto de un mismo documento
        texto = []

        # Para cada una de las imagenes
        for h in imagenes:
            if h != '.DS_Store':
                # Cargo la imagen
                img = Image.open(j + h) # Abre la imagen con pillow
                img.load()

                # Saco el texto
                text = pytesseract.image_to_string(img, lang='spa') # Extrae el texto de la
    imagen
                texto.append(text)


        # Guardo el documento '.txt' en la carpeta de guardado
        f = open (path_guardado+str(c)+".txt" ,'w')
        f.writelines(texto)
        f.close()

        print(c)



    # Sacamos los doscumentos exitentes en el path de guardado para comprobar que estan todos
    documentos = os.listdir(path_guardado)
    print(len(documentos))

```

Código 5: Uso de *pytesseract* y *PIL* para la creacion de los documentos *.txt* que representan cada una de las noticias

En una primera instancia hemos obtenido las noticias para dos de las categorías gracias al uso de Web Scrapping. Ahora que tenemos el conjunto de las noticias extraidas con OCR, vamos a generar un registro en formato *.csv* donde almacenamos información de cada una de las noticias. Nuevamente, se vuelve a empelear OCR sobre la carpeta de titulos para especificar este campo dentro del registro.

```
# Cargamos el registro
df = pd.read_excel("/Users/joelpardo/Desktop/Practica 2/Documentos/urls1.xlsx")


# Asignamos las variables
indices = df["Unnamed: 0"].tolist()
category = df["category"].tolist()
title = df["title"].tolist()
links = df["link"].tolist()


generico = "../Documentos"
path = generico + "/Links"


clases = ["Ciencia", "Politica"]
contador = indices[len(indices)-1] # 119

for z in clases:

    doc = path + "/" + str(z) + '.txt'

    path_titulos = generico + "/" + str(z) + "/titulo"


    titulos = os.listdir(path_titulos)
    titulos = [f for f in titulos if f != ".DS_Store"]


    with open(doc,'r') as f:
        lin = f.readlines()
        c = 0

```

```
        for i in lin:
            if i != "\n":
                if z == "Ciencia":
                    clase = "science"
                    category.append(clase)

                if z == "Politica":
                    clase = "politics"
                    category.append(clase)

                print(c)
                # Cargo la imagen
                img = Image.open(path_titulos + "/" + titulos[c]) # Abre la imagen con pillow
                img.load()


                # Saco el texto
                text = pytesseract.image_to_string(img, lang='spa') # Extrae el texto de la
    imagen
                text = text.replace('\n', '')
                title.append(text.strip())
                c += 1

                new_string = i.replace('\n', '')
                links.append(new_string.strip())


                contador += 1
                indices.append(contador)


print(len(indices))
print(len(category))
print(len(title))
print(len(links))


df_new = pd.DataFrame()
df_new["index"] = indices
df_new["category"] = category

path_rutas = "./Datos/Raw_data" #./TextClassification/

clases2 = df_new["category"].unique().tolist() #0-3

rutas_docs = []
r = []
for h in clases2:
    for a in range(1,61):
        p = path_rutas+ "/" + h + "/" + str(a) + ".txt"
        rutas_docs.append(p)
        r.append(a)

df_new["n_doc"] = r
df_new["title"] = title
df_new["path"] = rutas_docs
df_new["link"] = links

df_new.to_csv('../Documentos/urls.csv', header=True, index=False)

```

Código 6: Uso de *pandas* para la creacion del registro que representan cada una de las noticias

D. Lista de stop words

La lista de stop words del modelo *es_core_news_lg* de la librería *spaCy* es:
a, acuerdo, adelante, ademas, además, afirmó, agregó, ahi, ahora, ahí, al, algo, alguna, algunas, alguno, algunos, algún, alli, allí, alrededor, ambos, ante, anterior, antes, apenas, aproximadamente, aquel, aquella, aquellas, aquello, aquellos, aqui, aquél, aquélla, aquéllas, aquéllos, aquí, arriba, aseguró, asi, así, atras, aun, aunque, añadió,

aún, bajo, bastante, bien, breve, buen, buena, buenas, bueno, buenos, cada, casi, cierta, ciertas, cierto, ciertos, cinco, claro, comentó, como, con, conmigo, conocer, conseguimos, conseguir, considera, consideró, consigo, consigue, consiguen, consigues, contigo, contra, creo, cual, cuales, cualquier, cuando, cuanta, cuantas, cuanto, cuantos, cuatro, cuenta, cuál, cuáles, cuándo, cuánta, cuántas, cuánto, cuántos, cómo, da, dado, dan, dar, de, debajo, debe, deben, debido, decir, dejó, del, delante, demasiado, demás, dentro, deprisa, desde, despacio, despues, después, detras, detrás, dia, dias, dice, dicen, dicho, dieron, diez, diferente, diferentes, dijeron, dijo, dio, doce, donde, dos, durante, día, días, dónde, e, el, ella, ellas, ello, ellos, embargo, en, encima, encuentra, enfrente, enseguida, entonces, entre, era, eramos, eran, eras, eres, es, esa, esas, ese, eso, esos, esta, estaba, estaban, estado, estados, estais, estamos, estan, estar, estará, estas, este, esto, estos, estoy, estuvo, está, están, excepto, existe, existen, explicó, expresó, fin, final, fue, fuera, fueron, fui, fuimos, gran, grande, grandes, ha, haber, habia, habla, hablan, habrá, había, habían, hace, haceis, hacemos, hacen, hacer, hacerlo, haces, hacia, haciendo, hago, han, hasta, hay, haya, he, hecho, hemos, hicieron, hizo, hoy, hubo, igual, incluso, indicó, informo, informó, ir, junto, la, lado, largo, las, le, les, llegó, lleva, llevar, lo, los, luego, mal, manera, manifestó, mas, mayor, me, mediante, medio, mejor, mencionó, menos, menudo, mi, mia, mias, mientras, mio, mios, mis, misma, mismas, mismo, mismos, modo, mucha, muchas, mucho, muchos, muy, más, mí, mía, mías, mío, míos, nada, nadie, ni, ninguna, ningunas, ninguno, ningunos, ningún, no, nos, nosotras, nosotros, nuestra, nuestras, nuestro, nuestros, nueva, nuevas, nueve, nuevo, nuevos, nunca, o, ocho, once, os, otra, otras, otro, otros, para, parece, parte, partir, pasada, pasado, paìs, peor, pero, pesar, poca, pocas, poco, pocos, podeis, podemos, poder, podria, podriais, podriamos, podrian, podrias, podrá, podrán, podría, podrían, poner, por, porque, posible, primer, primera, primero, primeros, pronto, propia, propias, propio, propios, proximo, próximo, próximos, pudo, pueda, puede, pueden, puedo, pues, qeu, que, quedó, queremos, quien, quienes, quiere, quiza, quizas, quizá, quizás, quién, quiénes, qué, realizado, realizar, realizó, repente, respecto, sabe, sabeis, sabemos, saben, saber, sabes, salvo, se, sea, sean, segun, segunda, segundo, según, seis, ser, sera, será, serán, sería, señaló, si, sido, siempre, siendo, siete, sigue, siguiente, sin, sino, sobre, sois, sola, solamente, solas, solo, solos, somos, son, soy, su, supuesto, sus, suya, suyas, suyo, suyos, sé, sí, sólo, tal, tambien, también, tampoco, tan, tanto, tarde, te, temprano, tendrá, tendrán, teneis, tenemos, tener, tenga, tengo, tenido, tenía, tercera, tercero, ti, tiene, tienen, toda, todas, todavia, todavía, todo, todos, total, tras, trata, través, tres, tu, tus, tuvo, tuya, tuyas, tuyo, tuyos, tú, u, ultimo, un, una, unas, uno, unos, usa, usais, usamos, usan, usar, usas, uso, usted, ustedes, va, vais, vamos, van, varias, varios, vaya, veces, ver, verdad, verdadera, verdadero, vez, vosotras, vosotros, voy, vuestra, vuestras, vuestro, vuestros, y, ya, yo, él, ésa, ésas, ése, ésos, ésta, éstas, éste, éstos, última, últimas, último, últimos

E. Glosarios completos

Los glosarios de train que contienen 50 términos por categoría son los siguientes:

deportes	**salud**	**ciencia**	**política**
seguidor	estigma	sal	silva
mbappe	atras	cola	rosell
correr	cancer	meteoro	vuelo
suarez	anemia	agujero	precio
resto	cafa	llama	torra
mans	aceite	gemanidas	reyes
boston	oliva	banyoles	plataforma
estabilidad	mascarilla	neandertal	anunciara
horford	pulmonar	congelacia	perao
cristiano	unicef	leo	sinema
exencion	cannabis	latigo	baron
enrique	pet	supersa	turismo
booker	bronquiolitis	cernan	ayres
tatum	requerir	quipus	animal
cambio	congelado	congelar	versia
formato	fisioterapia	nudo	podem
club	quiraorgica	fuego	unilateral
juventus	creatividad	estampido	junts
ronaldo	tc	sonido	aborto
wiggins	rehabilitacia	menta	juventud
gonzalez	research	supermasivo	adjudicacia
gonzalo	signo	observacia	procesado
colombia	uk	perseidas	ex
magnussen	vejiga	inca	dema
steiner	marfan	matematica	magistrado
gasto	sindrome	alcoholismo	abascal
resistencia	deberiar	fumador	cgp
lakers	oxigeno	fumar	arizona
pts	lobo	izquierdo	desbloqueo
reb	manada	mutacion	perro
verdasco	parasito	coruaa	normalidad
kosmos	beneficio	diplodocus	sandro
ktm	permafrost	roth	ribera
moto	recoletas	iphone	ven
atletico	mascaras	mandabula	tope
gimenez	fibrotico	martanez	geno
siebert	estancamiento	pertenecia	hidra
rez	expreso	sapiens	corrupto
dolares	viatris	galaxia	consistorio
grada	macula	calcio	bolaaos
krack	vision	puente	infraestructura
marko	farmaca	intercalar	ceuta
aerodinamico	utico	rotacia	deu
motogp	ir	atma	marruecos
golden	poveda	sfera	lamite
state	cereal	comercio	mejoraa
warriors	oleocantal	dinosaurio	aentiendena
medicamento	dlm	bunsen	denominador
diciembre	velazquez	comandante	entiendena
gira	gondii	masa	page

Tabla 3: Términos de cada glosario ordenados según peso tf-idf total

F. Modelo tf-idf en detalle

F.1. Proceso de clasificación

Utilizamos la librería *gensim* de python para crear nuestros elementos del Vector Space Model.

En primer en el código **??** mostramos la creación de nuestro diccionario en base a nuestros glosarios.

```
def create_dictionary(glosarios):
    # Obtenemos los tokens de nuestros glosarios
    doc_tokens = [[termino for termino in glosario] for glosario in glosarios.values()]
    # Los guardamos en un diccionario de la libreria gensim
    dictionary = corpora.Dictionary(doc_tokens)
    return dictionary

```

Código 7: Obtención del diccionario

En el código 8 mostramos la creación de los *bag of words* de nuestros documentos en base a nuestro diccionario:

```
def create_bag_of_words(docs_list, dictionary):
    # Para cada documento creamos una lista de tokens
    doc_tokens = [simple_preprocess(corpus) for corpus in docs_list]
    # Para cada documento creamos un bow en base a nuestro diccionario
    docs_bow = [dictionary.doc2bow(doc) for doc in doc_tokens]
    return docs_bow

```

Código 8: Creación de bag of words

El código 9 muestra la conversión de nuestros documentos de *bag of words* a tf-idf

```
def create_tfidf(docs_bow):
    docs_tfidf = models.TfidfModel(docs_bow, smartirs="lfc")
    return docs_tfidf

```

Código 9: Conversión a tf-idf

Una vez realizadas las operaciones anteriores obtenemos las representaciones tf-idf de los glosarioso como mostramos en el código 11

```
deportes_bow = glosarios_dict.doc2bow(glosarios["deportes"])
deportes_tfidf = tfidf_model[deportes_bow]

salud_bow = glosarios_dict.doc2bow(glosarios["salud"])
salud_tfidf = tfidf_model[salud_bow]

ciencia_bow = glosarios_dict.doc2bow(glosarios["ciencia"])
ciencia_tfidf = tfidf_model[ciencia_bow]

politica_bow = glosarios_dict.doc2bow(glosarios["politica"])
politica_tfidf = tfidf_model[politica_bow]

```

Código 10: Creación de bag of words

Calculamos la similaridad de cada documento con cada glosario y definimos su clase para guardarlos en un csv.

```

# Funcion que compara la similaridad de un documento con cada glosario y le asigna la
    categoria con la que tenga mayor similitud
def get_prediction(n_documento,
                   similarities_deportes,
                   similarities_salud,
                   similarities_ciencia,
                   similarities_politica):
    mejor_match = 0
    mejor = "No clasificado"
    if (similarities_deportes[n_documento] > mejor_match):
        mejor_match = similarities_deportes[index]
```

```
        mejor = "sports"
    if (similarities_salud[n_documento] > mejor_match):
        mejor_match = similarities_salud[index]
        mejor = "health"
    if (similarities_ciencia[n_documento] > mejor_match):
        mejor_match = similarities_ciencia[index]
        mejor = "science"
    if (similarities_politica[n_documento] > mejor_match):
        mejor_match = similarities_politica[index]
        mejor = "politics"
    return mejor, mejor_match

# Creamos las matrices de similaridad de gensim para cada glosario
index = similarities.MatrixSimilarity(tfidf_model[docs_bow])
similarities_deportes = index[deportes_tfidf]
similarities_salud = index[salud_tfidf]
similarities_ciencia = index[ciencia_tfidf]
similarities_politica = index[politica_tfidf]

# Para cada documento, definimos su categoria
predictions = []
similarities = []
for i in range(len(docs_bow)):
    pred, sim = get_prediction(i,
                           similarities_deportes,
                           similarities_salud,
                           similarities_ciencia,
                           similarities_politica)
    similarities.append(sim)
    predictions.append(pred)

# Guardamos los resultados en un dataframe
noticias_test_dataframe.insert(2, "preds", predictions)
noticias_test_dataframe.insert(3, "similarity", similarities)

```

Código 11: Clasificación de los documentos

F.2. Resultados en detalle de la clasificación

A continuación mostramos las noticias clasificadas en cada categoría ordenadas por similaridad con la misma.

Título	Clasificación	Categoría original	Similaridad
Golden State asesta otro ...	sports	sports	0.356
Gobert toca fondo - AS ...	sports	sports	0.355
El fútbol es un cuento - ...	sports	sports	0.325
El fútbol fue lo de menos...	sports	sports	0.305
De jefe de estrategia de ...	sports	sports	0.273
Ferrari se enreda otra ve...	sports	sports	0.272
LeBron James: "No necesit...	sports	sports	0.272
Bolaños asegura que los b...	sports	politics	0.232
Domenicali: "Fernando Alo...	sports	sports	0.219
Djokovic, espectador de l...	sports	sports	0.219
Luis Enrique: «Si a Gavi ...	sports	sports	0.2
Los diagnósticos ocultos ...	sports	health	0.195
Houston da la sorpresa re...	sports	sports	0.194
El mejor golpe de Rafa Na...	sports	sports	0.187
¿Debo entrenar si estoy e...	sports	health	0.175
Sacramento arrasa en su v...	sports	sports	0.161
Rafa Nadal podría empezar...	sports	sports	0.155
Hawks y Nets han negociad...	sports	sports	0.155
Jimmy Butler regresará es...	sports	sports	0.155
El corredor de hidrógeno ...	sports	politics	0.153
UNAM se suma a la lucha c...	sports	health	0.152
T.J. Warren vuelve vuelve...	sports	sports	0.129
Los antidepresivos pueden...	sports	health	0.117
Europa recomienda retirar...	sports	health	0.117
Qué medicamentos tomar y ...	sports	health	0.105
.Es el mismo Alonso que c...	sports	sports	0.098
¿Qué trastorno se esconde...	sports	health	0.098
El Código Penal incluirá ...	sports	politics	0.078
La Agencia EspacialEspaño...	sports	science	0.078
Canadá Logra Su Primera C...	sports	sports	0.078
En todos los idiomas, las...	sports	science	0.077
Los pasajeros olvidados d...	sports	science	0.077
OMS cambia el nombre de l...	sports	health	0.072
Suspendido por una tangan...	sports	sports	0.072
Ferrero: "Trabajar con Zv...	sports	sports	0.058

Tabla 4: Documentos clasificados en la categoría *sports* ordenados por similaridad con el glosario

Título	Clasificación	Categoría original	Similaridad
Huevos de codorniz, una s...	health	health	0.297
“No se puede condenar a u...	health	health	0.258
El número de hogares acog...	health	politics	0.246
La revolucionariahipótesi...	health	science	0.225
¿De qué color es el fuego...	health	science	0.221
Los animales también tien...	health	science	0.218
Un fósil ubica al primere...	health	science	0.215
El aparato más desconocid...	health	science	0.209
¿Cómo prevenir la caída d...	health	health	0.191
¿Es cierto que comer rápi...	health	health	0.188
”Tenemos una epidemia de ...	health	health	0.174
BRONQUIOLITIS — Los pedia...	health	health	0.166
¿La bronquiolitis puede d...	health	health	0.166
Coosalud EPS conmemora el...	health	health	0.166
Conoce algunos de los rem...	health	health	0.166
La bronquilotis. Pregunta...	health	health	0.145
Guía para entender la epi...	health	health	0.139
Los condilomas o verrugas...	health	health	0.139
Una inteligencia artifici...	health	science	0.122
15 recetas con nueces par...	health	health	0.109
A la caza del segundoaguj...	health	science	0.104
Las bacterias intestinale...	health	health	0.096
Detectan envejecimiento p...	health	health	0.091
Sánchez aspira a que la U...	health	politics	0.075

Tabla 5: Documentos clasificados en la categoría *health* ordenados por similaridad con el glosario

Título	Clasificación	Categoría original	Similaridad
Cuándo es la última lunal...	science	science	0.39
Un ordenador cuánticologr...	science	science	0.281
¿Podían los diplodocuspro...	science	science	0.28
Marte no está muerto:hall...	science	science	0.262
Un meteorito sobrevuelala...	science	science	0.244
Medio siglo desde el Apol...	science	science	0.237
¿Por qué se echa sal a la...	science	science	0.223
La catástrofe del vacío (...	science	science	0.215
PSOE y Unidas Podemos pla...	science	politics	0.214
Los responsables de medir...	science	science	0.2
Lluvia de estrellas de la...	science	science	0.189
Nudos que representan núm...	science	science	0.186
¿Y si los dinosaurios no ...	science	science	0.186
MBST® Resonancia Terapéut...	science	health	0.179
Buenos hábitos para contr...	science	health	0.17
Así es el síndrome de la ...	science	science	0.168
La ciencia confirma que e...	science	health	0.143
Hallada en Marte una zona...	science	science	0.139
Xavier Trias llama a los ...	science	politics	0.139
¿Existen minisatélites qu...	science	science	0.114
El mejor superalimento fi...	science	health	0.107
Stakhovsky carga contra l...	science	sports	0.094

Tabla 6: Documentos clasificados en la categoría *science* ordenados por similaridad con el glosario

Título	Clasificación	Categoría original	Similaridad
El Ministerio de Igualdad...	politics	politics	0.375
El espionaje a periodista...	politics	politics	0.324
La vicepresidenta de Ceut...	politics	politics	0.293
Que los politicos noJalte...	politics	politics	0.282
Los 5 alimentos populares...	politics	health	0.265
El Poder Judicial propone...	politics	politics	0.248
David De la Cruz, un cand...	politics	politics	0.247
Perú teme una debacle del...	politics	politics	0.225
Feijóo rechaza la moción ...	politics	politics	0.204
La comisión de Justicia a...	politics	politics	0.198
Los conservadores del Pod...	politics	politics	0.194
Podemos propone un cheque...	politics	politics	0.194
Sandro Rosell anunciará t...	politics	politics	0.193
Podemos ve con temor la r...	politics	politics	0.191
Que el ruido no ensordezc...	politics	politics	0.183
China, Ucranía y el litio...	politics	science	0.18
España y el bloque de paí...	politics	politics	0.174
No te obsesiones con laNo...	politics	science	0.173
Una senadora demócrata de...	politics	politics	0.172
El Gobierno cambia la ley...	politics	politics	0.172
El PSOE y Unidas Podemos ...	politics	politics	0.154
Leclerc explica por qué s...	politics	sports	0.15
Trias se suma a la carrer...	politics	politics	0.145
Audi crece para la F1 - A...	politics	sports	0.143
EE.UU. anunciará el marte...	politics	science	0.143
Campazzo quiere jugar en ...	politics	sports	0.142
El ADN más antiguo jamás ...	politics	science	0.139
Hito histórico de EEUU en...	politics	science	0.134
UPN rompe con PP y Ciudad...	politics	politics	0.112
En defensa del voto a los...	politics	politics	0.083

Tabla 7: Documentos clasificados en la categoría *politics* ordenados por similaridad con el glosario

Título	Clasificación	Categoría original	Similaridad
Meloni cancela su presenc...	No clasificado	politics	0.0
Álex Palou, confirmado co...	No clasificado	sports	0.0
La respuesta de Rafa Nada...	No clasificado	sports	0.0
La dieta que reduce en un...	No clasificado	health	0.0
Doncic se sale en la Meca...	No clasificado	sports	0.0
El descalabro gigante de ...	No clasificado	sports	0.0
La lesión de Juancho Hern...	No clasificado	sports	0.0
Seis niños mueren en el R...	No clasificado	health	0.0
El H2Med costará 2.500 mi...	No clasificado	politics	0.0

Tabla 8: Documentos no clasificados

G. Otros clasificadores en detalle

El código fuente que hemos empleado para implementar otro tipo de clasificadores se encuentra en *Clasificacion-Resto.ipynb*

Comenzamos cargando el tokenizador a partir de la librería *spacy* y definimos las funciones para crear las diferentes mutaciones de datos.

```
    nlp = spacy.load("es_core_news_lg") #Mejor modelo optimizado para la CPU

    #BoW
    bow_vectorizer = CountVectorizer(min_df=0.01, max_df=0.9, vocabulary=vocabulario_train)



    #Tf-idf
    tfidf_vectorizer = TfidfTransformer()

    #Funciones de WV.
    def averaged_word_vectorizer(corpus):
        '''Aplica la funci n de c lculo del WE promedio a todos los
        documentos del corpus (cada doc es una lista de tokens)'''
        features = [nlp(doc).vector
                        for doc in corpus]
        return np.array(features)

    def tfidf_wtd_avg_word_vectors(doc, word_tfidf_map):
        '''Aplica la funci n de c lculo del WE ponderado por TF-IDF
        a un documento (como lista de tokens)'''
        tokens = doc.split()

        feature_vector = np.zeros((nlp.vocab.vectors_length,),dtype="float64")
        wts = 0.
        for word in tokens:
            if nlp.vocab[word].has_vector and word_tfidf_map.get(word, 0): #s lo considera
    palabras conocidas
                weighted_word_vector = word_tfidf_map[word] * nlp.vocab[word].vector
                wts = wts + 1
                feature_vector = np.add(feature_vector, weighted_word_vector)
        if wts:
            feature_vector = np.divide(feature_vector, wts)

        return feature_vector

    def tfidf_weighted_averaged_word_vectorizer(corpus, word_tfidf_map):
        '''Aplica la funci n de c lculo del WE ponderado por TF-IDF a todos los
        documentos del corpus (cada doc es una lista de tokens)'''
        features = [tfidf_wtd_avg_word_vectors(doc, word_tfidf_map)
                        for doc in corpus]
        return np.array(features)

```

Código 12: Mutaciones de los datos

De esta forma, quedaría:

```
    # caracter sticas bag of words
    bow_train_features = bow_vectorizer.fit_transform(norm_train_corpus)
    bow_test_features = bow_vectorizer.transform(norm_test_corpus)

    # caracter sticas tfidf (a partir del BoW)
    tfidf_train_features = tfidf_vectorizer.fit_transform(bow_train_features)
    tfidf_test_features = tfidf_vectorizer.transform(bow_test_features)

    # caracter sticas averaged word vector
    avg_wv_train_features = averaged_word_vectorizer(norm_train_corpus)
    avg_wv_test_features = averaged_word_vectorizer(norm_test_corpus)

    # caracter sticas tfidf weighted averaged word vector
    word_tfidf_map = {key:value for (key, value) in zip(bow_vectorizer.get_feature_names_out()
    , tfidf_vectorizer.idf_)}
```

```
tfidf_wv_train_features = tfidf_weighted_averaged_word_vectorizer(norm_train_corpus,
word_tfidf_map)

tfidf_wv_test_features = tfidf_weighted_averaged_word_vectorizer(norm_test_corpus,
word_tfidf_map)
```

Código 13: Diferentes tipos de entradas

Denifimos ahora las diferentes métricas a emplear y la implementación del entrenamiento y evaluación.

```
def get_metrics(true_labels, predicted_labels):
"""Calculamos distintas metricas sobre el
rendimiento del modelo. Devuelve un diccionario
con los parametros medidos"""

return {
    'Accuracy': np.round(
                    metrics.accuracy_score(true_labels,
                                           predicted_labels),
                    3),
    'Precision': np.round(
                    metrics.precision_score(true_labels,
                                           predicted_labels,
                                           average='weighted',
                                           zero_division=0),
                    3),
'Recall': np.round(
                    metrics.recall_score(true_labels,
                                           predicted_labels,
                                           average='weighted',
                                           zero_division=0),
                    3),
'F1 Score': np.round(
                    metrics.f1_score(true_labels,
                                           predicted_labels,
                                           average='weighted',
                                           zero_division=0),
                    3)}


def train_predict_evaluate_model(classifier,
                                 train_features, train_labels,
                                 test_features, test_labels):
    """Funcion que entrena un modelo de clasificacion sobre
    un conjunto de entrenamiento, lo aplica sobre un conjunto
    de test y devuelve la prediccion sobre el conjunto de test
    y las metricas de rendimiento"""
    # genera modelo
    classifier.fit(train_features, train_labels)
    # predice usando el modelo sobre test
    predictions = classifier.predict(test_features)
    # eval a rendimiento de la predicci n
    metricas = get_metrics(true_labels=test_labels,
                predicted_labels=predictions)
    return predictions, metricas
```

Código 14: Definición de métricas y proceso

Pasamos a la primera fase donde probamos los distintos tipos de clasificadores del paquete *Scikit-Learn* con los distintos tipos de datos.

```
modelLR = LogisticRegression(solver='liblinear')
modelNB = GaussianNB() #var_smoothing=1e-9
modelSVM = SGDClassifier(loss='hinge', max_iter=1000)
modelRBFSVM = SVC(gamma='scale', C=2)

modelos = [('Logistic Regression', modelLR),
           ('Naive Bayes', modelNB),
           ('Linear SVM', modelSVM),
           ('Gauss kernel SVM', modelRBFSVM)]
```

```
    metricas = []
    resultados = []

    # Modelos con caracter sticas BoW
    bow_train_features_ = bow_train_features.toarray()
    bow_test_features_ = bow_test_features.toarray()
    for m, clf in modelos:
        prediccion, metrica = train_predict_evaluate_model(classifier=clf,
                                            train_features=bow_train_features_,
                                            train_labels=train_labels,
                                            test_features=bow_test_features_,
                                            test_labels=test_labels)
        metrica['modelo']=f'{m} BoW'
        resultados.append(prediccion)
        metricas.append(metrica)

    # Modelos con caracter sticas TF-IDF
    tfidf_train_features_ = tfidf_train_features.toarray()
    tfidf_test_features_ = tfidf_test_features.toarray()
    for m, clf in modelos:
        prediccion, metrica = train_predict_evaluate_model(classifier=clf,
                                            train_features=tfidf_train_features_,
                                            train_labels=train_labels,
                                            test_features=tfidf_test_features_,
                                            test_labels=test_labels)
        metrica['modelo']=f'{m} tfidf'
        resultados.append(prediccion)
        metricas.append(metrica)

    # Modelos con caracter sticas averaged word vectors
    avg_wv_train_features_ = avg_wv_train_features
    avg_wv_test_features_ = avg_wv_test_features
    for m, clf in modelos:
        prediccion, metrica = train_predict_evaluate_model(classifier=clf,
                                            train_features=avg_wv_train_features_,
                                            train_labels=train_labels,
                                            test_features=avg_wv_test_features_,
                                            test_labels=test_labels)
        metrica['modelo']=f'{m} averaged'
        resultados.append(prediccion)
        metricas.append(metrica)

    # Modelos con caracter sticas tfidf weighted averaged word vectors
    tfidf_wv_train_features_ = tfidf_wv_train_features
    tfidf_wv_test_features_ = tfidf_wv_test_features
    for m, clf in modelos:
        prediccion, metrica = train_predict_evaluate_model(classifier=clf,
                                            train_features=tfidf_wv_train_features_,
                                            train_labels=train_labels,
                                            test_features=tfidf_wv_test_features_,
                                            test_labels=test_labels)
        metrica['modelo']=f'{m} tfidf wv'
        resultados.append(prediccion)
        metricas.append(metrica)

```

Código 15: Definición de modelos y entrenamiento con diferentes entradas

Obtenemos los resultados de la primera fase de la siguiente forma:

```
    metricas = pd.DataFrame(metricas)
    metricas = metricas.sort_values('Accuracy',ascending=False)
    display(metricas)

    metricas.head(1)

```

Código 16: Resultados de la primera fase

Pamos pues a la segunda fase escogiendo los datos que mejor funcionan y los cuatro mejores modelos. Generalmente, son los promediados de los word embeddings. Entrenamos los modelos escogidos probando diferentes parámetros para comprobar si su funcionamiento mejora.

```
modelos = metricas['modelo'].tolist()

mejor = modelos[0]
segundo = modelos[1]
tercero = modelos[2]
cuarto = modelos[3]

moa = [mejor, segundo, tercero, cuarto]
print(moa)


```

Código 17: Selección de los mejores modelos

Como los modelos tienen en el nombre los datos con los que han sido entrenados, es necesario separar dicho nombre para obtener por un lado el nombre de modelo y por otro el conjunto de datos.

```
def separar_modelo(modelo):
""" funcion para separar del nombre de modelo los datos con los que ha entrenado
- modelo: nombre del modelo
"""
sep = modelo.split(' ')

if sep[len(sep)-1]== "wv":
    mo = ' '.join(sep[:len(sep)-2])

else:
    mo = ' '.join(sep[:len(sep)-1])


datos = sep[len(sep)-1]


return mo, datos




dat = []
mod = []

for i in moa:
    m, d = separar_modelo(i)
    mod.append(m)
    dat.append(d)


print(mod)
print(dat)

```

Código 18: Selección de los mejores modelos

Implementación de la segunda fase:

```
metricas2 = []
resultados = []


for datos, mo in zip(dat, mod):
    # DATOS
    if datos == 'Bow':
        train_features_ = bow_train_features.toarray()
        test_features_ = bow_test_features.toarray()

    if datos == 'tfidf':
        train_features_ = tfidf_train_features.toarray()
        test_features_ = tfidf_test_features.toarray()

    if datos == 'averaged':
        train_features_ = avg_wv_train_features
        test_features_ = avg_wv_test_features

```

```
        else: # tfidf wv
            train_features_ = tfidf_wv_train_features
            test_features_ = tfidf_wv_test_features




        # MODELOS

        if mo == 'Logistic Regression':
            # https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

            sol = ["lbfgs", "liblinear", "newton-cg", "newton-cholesky", "sag", "saga"]

            for i in sol:
                modelLR = LogisticRegression(solver=i)
                prediccion, metrica = train_predict_evaluate_model(classifier=modelLR,
                                                   train_features=train_features_,
                                                   train_labels=train_labels,
                                                   test_features=test_features_,
                                                   test_labels=test_labels)
                metrica['modelo']=f'{i} - Logistic Regression ' + datos
                resultados.append(prediccion)
                metricas2.append(metrica)

        if mo == 'Naive Bayes':
            # https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
            var_smo = [1e-1,1e-2,1e-3,1e-4,1e-5,1e-6,1e-7,1e-8,1e-9,1e-10,1e-11,1e-12,1e-13,1e-14,1e-15]

            for i in var_smo:
                modelNB = GaussianNB(var_smoothing=i)
                prediccion, metrica = train_predict_evaluate_model(classifier=modelNB,
                                                   train_features=train_features_,
                                                   train_labels=train_labels,
                                                   test_features=test_features_,
                                                   test_labels=test_labels)
                metrica['modelo']=f'{i} - Naive Bayes ' + datos
                resultados.append(prediccion)
                metricas2.append(metrica)

        if mo == 'Linear SVM':
            # https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
            loss = ["hinge", "log_loss", "log", "modified_huber", "squared_hinge", "perceptron", "squared_error", "huber", "epsilon_insensitive", "squared_epsilon_insensitive"]

            for i in loss:
                modelSVM = SGDClassifier(loss=i, max_iter=1000)
                prediccion, metrica = train_predict_evaluate_model(classifier=modelSVM,
                                                   train_features=train_features_,
                                                   train_labels=train_labels,
                                                   test_features=test_features_,
                                                   test_labels=test_labels)
                metrica['modelo']=f'{i} - Linear SVM ' + datos
                resultados.append(prediccion)
                metricas2.append(metrica)


        if mo == 'Gauss kernel SVM':
            # https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
            gamma = ["scale", "auto"]

            for i in gamma:
                modelRBFSVM = SVC(gamma=i, C=2)
                prediccion, metrica = train_predict_evaluate_model(classifier=modelRBFSVM,
                                                   train_features=train_features_,
                                                   train_labels=train_labels,
                                                   test_features=test_features_,
```

```
                                                    test_labels=test_labels)
                metrica['modelo']=f'{i} - Gauss kernel SVM ' + datos
                resultados.append(prediccion)
                metricas2.append(metrica)

```

Código 19: Variación de parámetros para comprobar mejora

Obtenemos los resultados de la segunda fase:

```
metricas3 = pd.DataFrame(metricas2)
metricas3 = metricas3.sort_values('Accuracy',ascending=False)
display(metricas3)
metricas3.head(1)

```

Código 20: Obtención de resultados de la segunda fase

Pamos a la tercera fase. Por el momento tenemos los modelos que mejor funcionan con el conjunto de datos que mejor se adapta a los modelos. Ya solo nos falta comprobar el rendiemiento de estos modelos en la prática. Seleccionamos los modelos:

```
modelos = metricas3['modelo'].tolist()

mejor = modelos[0]
segundo = modelos[1]
tercero = modelos[2]
# cuarto = modelos[3]

moa2 = [mejor, segundo, tercero] #, cuarto
print(moa2)


dat = []
mod = []
params = []

for i in moa2:
    m, d = separar_modelo(i)

    dat.append(d)


    p = m.split('-')

    if 'newton' in p:
        para = '-'.join(p[0:2])
        mdo = ' '.join(p[2:])

    else:
        para = p[0]
        mdo = ' '.join(p[1:])


    params.append(para.strip())
    mod.append(mdo.strip())


print(mod)
print(dat)
print(params)

```

Código 21: Selección de modelos para la tecera fase

Definimos una función para la puestra en producción de dichos modelos para que no haya que copiar y pegar los modelos sino que lo haga de formaautomática.

```
def produccion(mod, dat, params):
    """ Funcion para crear los tres mejores modelos en produccion
    - mod: lista con los nombres de modelos
    - dat: lista con los datos de los mejores modelos
    - params: lista de los parametros correspondientes a los modelos
```

```
    """

    resultados2 = []
    metricas4 = []
    clasificadores = []

    for datos, mo, param in zip(dat, mod, params):

        # DATOS
        if datos == 'Bow':
            train_features_ = bow_train_features.toarray()
            test_features_ = bow_test_features.toarray()

        if datos == 'tfidf':
            train_features_ = tfidf_train_features.toarray()
            test_features_ = tfidf_test_features.toarray()

        if datos == 'averaged':
            train_features_ = avg_wv_train_features
            test_features_ = avg_wv_test_features

        else: # tfidf wv
            train_features_ = tfidf_wv_train_features
            test_features_ = tfidf_wv_test_features




        # MODELOS

        if mo == 'Logistic Regression':
            # https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html


            modelLR = LogisticRegression(solver=param)
            prediccion, metrica = train_predict_evaluate_model(classifier=modelLR,
                                                train_features=train_features_,
                                                train_labels=train_labels,
                                                test_features=test_features_,
                                                test_labels=test_labels)
            metrica['modelo']=f'{param} - Logistic Regression'
            resultados2.append(prediccion)
            metricas4.append(metrica)
            clasificadores.append(modelLR)

        if mo == 'Naive Bayes':
            # https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

            modelNB = GaussianNB(var_smoothing=param)
            prediccion, metrica = train_predict_evaluate_model(classifier=modelNB,
                                                train_features=train_features_,
                                                train_labels=train_labels,
                                                test_features=test_features_,
                                                test_labels=test_labels)
            metrica['modelo']=f'{param} - Naive Bayes'
            resultados2.append(prediccion)
            metricas4.append(metrica)
            clasificadores.append(modelNB)

        if mo == 'Linear SVM':
            # https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

            modelSVM = SGDClassifier(loss=param, max_iter=1000)
            prediccion, metrica = train_predict_evaluate_model(classifier=modelSVM,
                                                train_features=train_features_,
                                                train_labels=train_labels,
                                                test_features=test_features_,
                                                test_labels=test_labels)
```

```
                metrica['modelo']=f'{param} - Linear SVM'
                resultados2.append(prediccion)
                metricas4.append(metrica)
                clasificadores.append(modelSVM)


            if mo == 'Gauss kernel SVM':
                # https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

                modelRBFSVM = SVC(gamma=param, C=2)
                prediccion, metrica = train_predict_evaluate_model(classifier=modelRBFSVM,
                                                   train_features=train_features_,
                                                   train_labels=train_labels,
                                                   test_features=test_features_,
                                                   test_labels=test_labels)
                metrica['modelo']=f'{param} - Gauss kernel SVM'
                resultados2.append(prediccion)
                metricas4.append(metrica)
                clasificadores.append(modelRBFSVM)


        return resultados2, metricas4, clasificadores

    resultados2, metricas4, clasificadores = produccion(mod, dat, params)


```

Código 22: Definición de los modelos en una tercera fase

Obtenemos los resultados la tercera fase y cargamos el registro de las anteriores iteraciones para almacenar los resuldatos de la iteración actual.

```
    metricas5 = pd.DataFrame(metricas4)
    mra = pd.read_csv(prediccion_r+'/metricas_produccion.csv')

    tot = pd.merge(mra, metricas5, how='outer')

    tot1 = tot.sort_values('Accuracy',ascending=False)
    display(tot1)

    predi = []
    vf = []
    labs = []

    for c, i in zip(moa2,resultados2):
        q = [c for n in range(len(i))]

        for f, g, r in zip(i,q, test_labels):
            predi.append(g)
            vf.append(f)
            labs.append(r)


    print(len(predi))
    print(len(vf))
    print(len(labs))


        res = pd.DataFrame()
    res['modelo'] = vf
    res['prediccion'] = predi
    res['category'] = labs

    display(res)

```

Código 23: Obtención de resultados de la tercera fase

Por último comprobamos el rendimiento de los tres mejores modelos para cada una de las categorías a partir de la matriz de confusión.

```
    for title, classifier in zip(moa2, clasificadores):
```

```
        disp = ConfusionMatrixDisplay.from_estimator(
            classifier,
            test_features_,
            test_labels,
            display_labels=categorias,
            cmap=plt.cm.Blues,
            normalize=None,
        )

        disp.ax_.set_title(title)

```

Código 24: Matriz de confusión de los tres mejores modelos

Referencias

[1] Christopher D. Manning, Prabhakar Raghavan e Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. DOI: 10.1017/CBO9780511809071.

[2] Anand V. Saurkar, Kedar G. Pathare y Shweta A. Gode. "An Overview on Web Scraping Techniques and Tools". En: 2018.