UNIVERSIDAD
POLITÉCNICA
DE MADRID

Clasificador documental con glosario

Alejandro Gabarre González
César García Cabeza

Curso académico 2020-21

Diciembre 2020

Asignatura:
Ingeniería Lingüística

Professor:
D. Jesús Cardeñosa Lera

Índice

1. Introducción

Los clasificadores documentales se basan en categorizar una serie de documentos en base a sus características. Atendiendo a esta definición, podemos distinguir fácilmente dos tipos de clasificadores documentales:

1. Clasificadores manuales: los textos se clasifican en una temática habiendo sido leídos por una persona y haciendo una clasificación lógica de los mismos. Es lo que ocurre, por ejemplo, en las bibliotecas físicas, donde los diversos documentos, los libros en este caso, están distribuidos en categorías de forma manual.
2. Clasificadores automáticos: los textos se clasifican en una temática sin una lectura y comprensión por parte de una persona. Es un programa informático el encargado de obtener una serie de características de los documentos y clasificarlos en base a estas características. Encontramos aquí dos tipos de clasificadores automáticos:
   a) Clasificadores documentales con glosario: parten de un conjunto de palabras clave, denominado glosario, como referencia.
   b) Clasificadores documentales sin glosario: agrupando los documentos en categorías en base a unas métricas de similitud.

1.1. Objetivos

El objetivo principal de esta práctica es el diseñar e implementar un clasificador documental con glosario que sea capaz de clasificar documentos en una de estas tres categorías: salud, deportes y política. Para lograr este objetivo, se plantean una serie de hitos los cuales vamos a ir consiguiendo hasta llegar al objetivo principal. Estos hitos son:

1. Obtención de documentos.
2. Creación de los glosarios.
3. Clasificar los documentos.
4. Almacenar los documentos clasificados en carpetas con el nombre de la temática predicha.
5. Ordenar los documentos presentes en cada una de estas carpetas según una métrica que indique con qué seguridad se ha clasificado en esa temática.

1.2. Material entregado

Para cumplir con el objetivo principal y con los hitos especificados, entregamos un fichero .zip con el nombre *Documents-Classifier.zip*, como se puede ver en la Figura 1.

- En la subcarpeta *documents* se encuentran los documentos de cada una de las temáticas, así como un fichero que contiene una lista de *stopwords* (Ver Sección 2.3.2).
- En la subcarpeta *keywords* vamos a guardar los glosarios generados para cada uno de los temas.
- En la última subcarpeta, *src*, se encuentran los scripts que se han utilizado para alcanzar los objetivos. Éstos son los dos ficheros que vemos con la extensión .ipynb.

- Finalmente, el documento gabarre_garcia_documents_classifier.pdf, documento que está leyendo actualmente.

Destacar que todo este material lo puede encontrar en este repositorio de Github.

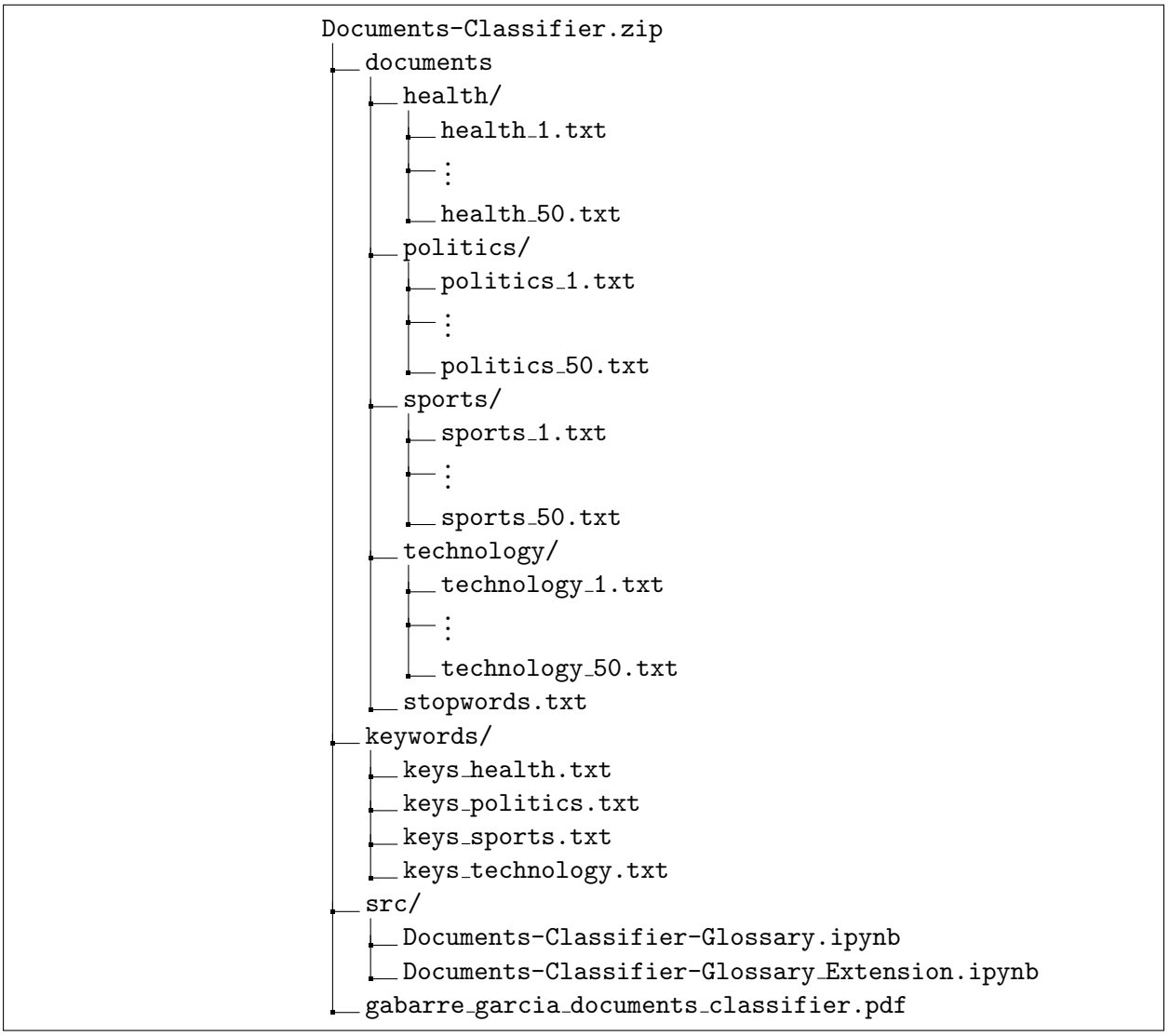

Figura 1: Estructura del zip.

1.3. Estructura

En primer lugar, en el Capítulo 2 explicaremos los aspectos llevados a cabo en el tratamiento de los documentos: recopilación, preprocesado, tokenización, etc. Seguidamente, en el Capítulo 3 explicaremos la metodología utilizada para la obtención de los glosarios así como el contenido de los mismos. En el Capítulo 4 hablaremos de los clasificadores de documentos que hemos utilizado, mostrando cómo los documentos se organizan en carpetas en base a su temática predicha y su probabilidad de pertenecer a la misma. Posteriormente, en el Capítulo 5 daremos un enfoque estadístico a los resultados obtenidos por los clasificadores para poder evaluarlos. En el capítulo 6 ampliaremos la práctica añadiendo una categoría más a nuestro clasificador y finalizaremos expresando las conclusiones obtenidas de la evaluación de los clasificadores en el Capítulo 7

Además, se incluye en el Anexo C un manual de uso que permite la ejecución de nuestro proyecto.

2. Dataset

Para lograr cumplir con los hitos indicados en el capítulo anterior, necesitamos adquirir los documentos necesarios, tratando de que se adecúen a nuestras necesidades. El objetivo principal de la práctica es saber indicar si un documento dado está más relacionado con la temática de política, deportes o salud. Para ello, necesitamos recopilar documentos clasificados en una de estas tres temáticas para construir el futuro glosario (Ver Capítulo 3).

2.1. Recopilación de documentos

Uno de los aspectos más importantes que tenemos que tener en cuenta es que nuestros documentos tienen que abarcar al máximo las posibilidades de cada temática. No podemos centrarnos en un conjunto de documentos específico. De esta forma, a la hora de buscar diferentes documentos de, por ejemplo, la temática deportes, no podemos centranos únicamente en documentos cuya temática principal sea el fútbol. Si hiciésemos esto, la variedad terminológica se limitaría únicamente a esta temática, y en el momento en el que se desee clasificar un documento cuya temática principal sea, por ejemplo, la caza, las probabilidades de que este nuevo documento sea similar a nuestros documentos previos de la temática deportes son muy bajas.

Sabiendo que debemos abarcar toda las temáticas con la mayor amplitud posible, vamos a asegurarnos de que obtenemos documentos que intenten cumplir con este requisito. Así, en los documentos obtenidos de deportes, encontramos documentos de fútbol, baloncesto, natación, balonmano, atletismo, motor, golf, caza...

También nos ha parecido interesante no centrarnos únicamente en una fuente de información de la que podamos obtener los documentos. Además de la variedad terminológica, que ya hemos visto cómo conseguir, vamos a intentar conseguir una mayor variabilidad en la manera en las que los diferentes documentos están escritos. Por lo tanto, vamos a obtener nuestros documentos de diversas publicaciones en periódicos online. A continuación se muestran, para cada temática, los sitios web de los cuáles se han obtenido diferentes documentos:

1. Salud
    a) Noticias en Salud
    b) Libertad Digital
    c) El Confidencial
    d) El Mundo
2. Deportes
    a) Marca
    b) Diario As
    c) Libertad Digital
    d) Federación Española de Tiro con Arco
    e) Portal Caza y Ocio
    f) OpenGolf
3. Política
    a) El País
    b) El Confidencial

c) Libertad Digital
d) La Vanguardia
e) El HuffPost
f) Periodista Digital

2.2. Organización de documentos

Para poder tener organizados los documentos de manera sencilla, hemos creado una carpeta para cada temática donde vamos a guardar todos los documentos asociados a la misma. Como veíamos en la Sección 1.2, cada una de estas carpetas va a contener 50 documentos, identificados cada uno como: *carpeta_nºdocumento.txt*.

2.3. Preprocesado

Los documentos que hemos obtenido y almacenado son documentos que podemos considerar documentos "crudos". Estos documentos poseen signos de puntuación o caracteres especiales, incluso algunos caracteres no habituales que pueden resultar del proceso de obtención de documentos al no obtener únicamente los caracteres.

Vamos a estar interesados en obtener, de los diferentes documentos, un conjunto de *tokens* que van a proporcionar una mejor información que la que puede proporcionar el documento completo. Estos tokens no están formados obligatoriamente por palabras individuales. Pongamos el siguiente ejemplo: si en nuestros documentos aparecen a menudo las palabras "código penal"de forma consecutiva, vamos a estar más interesados en poder guardar ambas palabras como un mismo token. Aquí es donde entran en juego los n-gramas. Podríamos tener en cuenta loa aparición de conjuntos de palabras como "Organización Mundial de la Salud", donde pueden llegar a aparecer hasta 5 términos de forma consecutiva que pueden ser inseparables. Sin embargo, hemos considerado que la aparición de conjuntos de más de 3 términos no van a afectar de manera positiva el objetivo de la práctica, por lo que vamos a trabajar únicamente con términos individuales (unigramas) y términos de dos palabras (bigramas).

2.3.1. Limpieza de carácteres especiales

En primer lugar, vamos a eliminar todos los símbolos de puntuación y posibles caracteres especiales inesperados de cada uno de los documentos. Para ello, vamos a hacer uso de las expresiones regulares.

Es frecuente que determinados caracteres se necesiten eliminar. Sin embargo, también aparecen caracteres que deben ser reemplazados por un espacio en blanco, como pueden ser los guiones. Vamos a hacer uso de la librería de expresiones regulares *re* de Python[15] para definir nuestra función que va a limpiar del texto todos los caracteres especiales.

```
REPLACE_NO_SPACE = re.compile(
    "(\&)|(\%)|(\$)|(\€)|(\.)|(\;)|(\:)|(\!)|\
    (\')|(\?)|(\,)|(\")|(\()|(\))|(\[)|(\])|(\d+)|(\)|(\•)|(\\')")
REPLACE_WITH_SPACE = re.compile("(<br\s*/><br\s*/>)|(\-)|(\/)")
NO_SPACE = ""
SPACE = " "

def preprocess_document(document):
    document = REPLACE_NO_SPACE.sub(NO_SPACE, document.lower())
    document = REPLACE_WITH_SPACE.sub(SPACE, document)
    return document
```

Ejemplo Veamos cómo actúa esta función con un ejemplo:

```
document = "Texto$ .con-caracteres€ especiale?s"
# Printeamos el resultado del preprocesado
preprocess_document(document)
# 'texto con caracteres especiales'
```

2.3.2. Palabras vacías y tokenización

Llegado a este punto, tenemos todos los documentos sin signos de puntuación ni caracteres especiales. El siguiente paso va a ser eliminar las *stopwords* de nuestros documentos. Entendemos por *stopwords* aquellas palabras que no tienen significado, como pueden ser artículos, pronombres, preposiciones, conjunciones... Además de estas, también caben destacar todas las formas verbales de verbos como haber, ser o poder. Crear una lista de *stopwords* propia puede llegar a ser una tarea complicada, por lo que hemos decidido incluir la siguiente lista de *stopwords* en castellano que contiene un total de 443 palabras .

Aprovechando que vamos a evaluar cada una de las palabras que conforman cada documento para determinar si es o no una *stopword*, vamos a obtener en este mismo paso nuestros unigramas, que van a ser aquellas palabras que no son stopwords.

Teniendo nuestra lista de *stopwords* y haciendo uso de la función *wordpunct_tokenize* de la librería nltk, en concreto del módulo tokenize, definimos la función encargada de borrar las stopwords de un documento y devolver sus tokens:

```
def delete_stop_words(doc):
    tokens = wordpunct_tokenize(doc)
    clean = [
        token for token in tokens if token not in STOP_WORDS and len(token) > 2]
    return clean
```

Ejemplo Podemos ver cómo funciona en el siguiente ejemplo.

```
delete_stop_words("texto con caracteres especiales")
# ['texto', 'caracteres', 'especiales']
```

2.3.3. Obtención de raíces o lemas. ¿Es buena idea?

Dos de las posibilidades que aparecen con mucha frecuencia en la obtención de tokens a partir de textos son las opciones de obtener el lema (lemmatization) o la raíz (stemming) de las palabras que aparecen en los mismos. Sin embargo, estas dos técnicas pueden resultar perjudiciales por diversos motivos.

Como consecuencia de los diferentes extractores terminológicos que utilizaremos en la Sección 3, vamos a obtener una serie de frecuencias o valores que pueden ser muy altos en una forma morfológica de una palabra pero muy bajos en otras formas. Si obtuviésemos el lema o la raíz del término, estaríamos agrupando ambas formas morfológicas en una, cuando realmente nos va a interesar aquella con un mayor valor y no sus formas derivadas.

Además, otro impedimento importante es la ausencia de librerías o programas que permitan realizar estos dos procesos con buenos resultados para el español, mientras que para textos en inglés es mucho más fácil encontrar otros proyectos en los que ya se hayan realizado.

Poniendo énfasis en el primero de los dos motivos, vamos a mantener nuestros tokens como resultado de la función *delete_stop_words()*, sin obtener de éstos las raices o los lemas.

2.3.4. Bigramas

Como bien indicábamos en la introducción de esta sección, vamos a trabajar tanto con unigramas como con bigramas. Siguiendo los pasos anteriores hemos podido llegar a la obtención de los primeros, por lo que a continuación explicaremos el proceso llevado a cabo para la obtención de los segundos.'

Previamente definíamos bigrama como secuencia de dos palabras y poníamos como ejemplo de bigrama "código penal". Ahora bien, en este conjunto de tokens nada nos debe impedir la aparición de bigramas como puede ser el nombre propio "Le Normand". En este caso, en nuestra función *delete_stop_words()* estaríamos borrando la palabra "Le". Lo que ocurre con los n-gramas que no son unitarios es que no podemos despreciar fácilmente las *stopwords*. Quizás este es un caso particular, pero pongamos el caso que nombrábamos previamente en la introducción del capítulo: "Organización Mundial de la Salud". Si borrásemos las *stopwords* no tendríamos forma posible de obtener este n-grama, por lo que para los bigramas vamos a aplicar la misma lógica y no vamos a limpiar el documento de *stopwords*.

Para generar los bigramas del total de documentos de cada temática, definimos:

```
def get_bigrams(documents, threshold):
    """
    Obtiene una serie de bigramas del conjunto de documentos
    """
    # Obtenemos unos tokens simplemente separando por espacios los documentos
    token_ = [doc.split(" ") for doc in documents]
    # Entrenamos el modelo Phrases:
    # --- sentences -> Conjunto de documentos
    # --- min_count -> Ignora palabras y bigramas con frecuencia menor a este valor
```

```
    # --- threshold -> Umbral que determina el número de frases. A más valor menos expresiones.
    bigram = Phrases(sentences=token_, min_count=1,
                     threshold=threshold, delimiter=b' ')
    # El modelo Phrases sólo está pensado para aprender los bigramas.
    # Necesitamos usar un objeto Phraser para poder procesarlo y obtener los bigramas.
    bigram_phraser = Phraser(bigram)
    bigram_token = []
    for sent in token_:
        for bigram in bigram_phraser[sent]:
            # comprobamos que realmente es un bigrama
            if len(bigram.split(" ")) > 1:
                bigram_token.append(bigram)
    return list(set(bigram_token))
```

Ejemplo Veamos un ejemplo de la obtención de bigramas mediante esta función:

```
documentos = ["El arbol es rojo","El arbol viejo"]

get_bigrams(documentos, 1)

# ['El arbol', 'El arbol']
```

De esta forma, para cada uno de los documentos podemos buscar qué bigramas de los correspondientes al total de documentos de una temática se encuentran en el documento o no.

```
def check_bigram(x, bigrams):
    """
    Devuelve los bigramas que aparecen en la cadena x
    """
    return [bigram for bigram in bigrams if x.find(bigram) != -1]
```

3. Creación del glosario

Un *glosario* es un "catálogo de palabras de una misma disciplina, de un mismo campo de estudio, de una misma obra, etc., definidas o comentadas." (Real Academia Española, definición 1) [11]. La creación de estos glosarios corresponde con el segundo de los hitos de esta práctica, descritos en la Sección 1.1.

Los glosarios son una parte fundamental de esta práctica ya que son los que nos van a permitir posteriormente clasificar cada documento en una categoría u otra. Por tanto, una correcta clasificación de los documentos dependerá de una buena creación de los glosarios.

3.1. Metodología

Desde el primer momento, nuestra intención ha sido la de automatizar al máximo el proceso de generación de los glosarios. Para conseguir esta automatización, hemos decidido extraer los 100 términos más representativos de cada clase de 3 formas distintas para después hacer la intersección dos a dos de estos conjuntos y realizar finalmente un filtrado manual de esta agregación para quedarnos con 30 palabras. Por tanto, conseguimos generar los glosarios en un proceso *semi automático*, todo esto realizado únicamente con 15 documentos de cada clase.

El proceso de creación de los glosarios es el siguiente:

1. Escoger al azar 15 documentos de cada clase.
2. Extracción de los 100 términos más representativos de cada clase de 3 formas distintas, eliminando los términos que aparecen en varias clases al no ser representativos únicamente de una clase.
   a) Método 1: TF-IDF[1].
   b) Método 2: extractor terminológico ya implementado.
   c) Método 3: K-Means.
3. Agregar estos *glosarios locales* en un único *glosario global*.
4. Filtrar manualmente este glosario global y quedarnos con los 30 términos que consideramos más representativos.

En los cuatro siguientes apartados describiremos más en detalle los pasos 2, 3 y 4.

3.2. Glosario local: TF-IDF

El primer enfoque escogido para la creación de los glosarios ha sido el de usar el esquema de ponderación de términos tf-idf. El tf-idf es muy interesante al mezclar en un solo valor lo frecuente que es un término con su *rareza*. Esta métrica incluye por una parte **la frecuencia de un término en el documento** y por otra **la frecuencia inversa del documento**, es decir, la inversa del número de documentos en los que aparece un término. Salton y Buclkey, 1988 [13] proponen diversas variaciones, siendo la más común [1]

$$(1 + log\ f_{i,j}) \times \log \frac{N}{n_i} \tag{1}$$

donde N es el número total de documentos, n_i el número de documentos en los que aparece el término i-ésimo y $f_{i,j}$ el número de veces que aparece el término i-ésimo en el documento j-ésimo.

[1] *Term Frequency - Inverse Document Frequency*

Ejemplo Para comprender mejor esta métrica haremos uso de un pequeño ejemplo. Supongamos que tenemos dos documentos

D1 = *El árbol es rojo.*
D2 = *El árbol es azul y viejo.*

y queremos calcular los tf-idf de cada palabra. En la Tabla 1 podemos ver cómo habría que hacer para calcularlos. Nótese que para calcular el tf-idf se ha seguido la variación descrita antes. Los valores con mayor tf-idf y por tanto más representativos, son *rojo*, *azul*, *y* y *viejo*.

Tabla 1: Ejemplo tf-idf

	Frecuencia			**IDF**	**TF-IDF**		
	D1	**D2**	**Total**		**D1**	**D2**	**Total**
El	1	1	2	1	0	0	0
árbol	1	1	2	1	0	0	0
es	1	1	2	1	0	0	0
rojo	1	-	1	2	0,30103	-	0,30103
azul	-	1	1	2	-	0,30103	0,30103
y	-	1	1	2	-	0,30103	0,30103
viejo	-	1	1	2	-	0,30103	0,30103

Como consecuencia, mediante el tf-idf podemos saber qué términos son los más representativos de cada documento y, de manera análoga, podemos calcular qué términos son los más representativos de cada clase y quedarnos con los 100 primeros.

Cada documento lo representaremos con los unigramas calculados en la Sección 2.3.2 y los bigramas calculados en la Sección 2.3.4, de tal forma que mediante este enfoque seremos capaces de extraer términos clave formados por una o dos palabras. Destacar que para la creación de este glosario local se ha utilizado la librería *Gensim* [12]. *Gensim* es una librería de Python especializada en NLP[2].

Finalmente, en la Tabla 2 ponemos una muestra de los primeros 5 términos clave extraídos con este método (si desea comprobar la lista completa puede hacerlo en el Anexo A).

Tabla 2: Keywords TF-IDF (5 primeras)

Salud	**Deportes**	**Política**
tuberculosis	gasol	marín
teléfonos	vaccaro	ayuntamiento
plasma	stéfano	vivienda
móviles	haaland	supremo
metabolismo	balonmano	sociedad

3.3. Glosario local: Gensim

Para la creación de este glosario local se ha utilizado de nuevo la librería *Gensim* y, en concreto, la función *keywords* del módulo *summarization*.

[2]Natural Language Processing

Usar este extractor terminológico es tan simple como pasarle a la función *keywords* todos los documentos unidos en formato *string* y te devuelve los términos más representativos, tal como se explica en [14]. En la Tabla 3 se pueden observar los primeros 5 términos obtenidos para cada clase usando este método. Nótese que este método tiene en cuenta términos compuestos, como "socialista pedro".

Tabla 3: Keywords Gensim (5 primeras)

Salud	**Deportes**	**Política**
enfermedad	equipo	gobierno
enfermedades	equipos	sanchez
caso	jugador	socialista pedro
estudio	jugadores	presupuestos
estudios	madrid	presupuesto

Para consultar la lista completa de los 100 términos obtenidos para cada categoría con este método, vaya a la Tabla 24 en el Anexo A.

3.4. Glosario local: K-Means

El algoritmo K-Means [5] es un algoritmo de *clustering* perteneciente a la familia del aprendizaje automático no supervisado.

El objetivo de este algoritmo es la división en k grupos de un conjunto de observaciones en base a unas características. Para ello se calculan k centros de tal manera que cada observación inicial queda asignada al centro más cercano. Éste es un proceso iterativo en el que se intenta minimizar la distancia de las observaciones al centro de su grupo asignado.
Tras este proceso iterativo, cada centro o centroide tendrá asignado un conjunto de observaciones. Estas observaciones son similares entre ellas si están asignadas al mismo grupo. Además, se puede saber qué características de las observaciones son las más relevantes para cada grupo.

Haciendo la analogía a nuestro problema:

- Las observaciones son los 15 documentos de entrenamiento.
- Las características de cada documento son las palabras representadas por su tf-idf.
- Los documentos pertenecen a 3 categorías distintas $\implies k = 3$.

Por lo tanto, podemos aplicar este algoritmo en nuestro objetivo de crear los glosarios, tal como se explica en [3]. Al "clasificar"los 15 documentos en 3 grupos en base a sus palabras, podemos obtener los términos más representativos de cada grupo creado, siendo precisamente ésto, por definición, un glosario. Recordemos que en nuestro caso obtenemos los 100 términos más representativos de cada clase.

Para implementar esta forma de generar los glosarios se ha utilizado la librería *Scikit-learn* [10] de Python, la cual ya tiene integrado el algoritmo *K-Means*. Como aclaración, se puede observar a continuación el código de esta forma de generar los glosarios:

```python
def get_k_kmeans_keywords(data, k):
    vectorizer = TfidfVectorizer(ngram_range=(1,2))
    # documentos representados por sus palabras en formato tf-idf
    X = vectorizer.fit_transform(k_means_data)

    model = KMeans(n_clusters=3, init='k-means++', max_iter=1000, n_init=1, random_state = 5,
                   algorithm="full")
    model.fit(X)
    # ordenamos las características de cada centroide por distancia
    order_centroids = model.cluster_centers_.argsort()[:, ::-1]
    # obtenemos las características (palabras) representativas de cada centroide
    terms = vectorizer.get_feature_names()

    # obtenemos los k términos más representativas de cada grupo
    keywords_kmeans_politics = [terms[ind] for ind in order_centroids[0, :k]]
    keywords_kmeans_health = [terms[ind] for ind in order_centroids[1, :k]]
    keywords_kmeans_sports = [terms[ind] for ind in order_centroids[2, :k]]

    return keywords_kmeans_politics, keywords_kmeans_health, keywords_kmeans_sports
```

En la Tabla 4 se pueden observar los 5 primeros términos de cada clase obtenidos mediante este método. Para ver la tabla completa vaya al Anexo A.

Tabla 4: Keywords KMeans (5 primeras)

Salud	Deportes	Política
día	equipo	erc
enfermedad	jugador	marín
covid	liga	ley
nacional	jordan	presupuestos
horas	nba	felipe

3.5. Glosario agregado

Tras calcular por separado estos tres glosarios locales (TF-IDF, Gensim, K-Means) hemos de, alguna forma, agregarlos para obtener un único glosario global.

Para lograr esta agregación se ha optado por hacer la intersección dos a dos de los glosarios locales para cada clase, quedándonos con las palabras que apareciesen al menos en dos de los glosarios locales. A continuación se puede ver un extracto de código donde se explica este procedimiento.

```python
def check_relevant_keywords(d1, d2, d3):
    # Hacemos la intersección dos a dos entre los tres glosarios locales
    i1 = set(d1) & set(d2)
    i2 = set(d1) & set(d3)
    i3 = set(d2) & set(d3)

```

```
    repeated = set(list(i1.union(i2).union(i3)))
    return repeated

# Nos quedamos con las palabras que aparecen al menos en dos glosario locales
relevant_keywords_politics = list(check_relevant_keywords(keywords_kmeans_politics,
                                          keywords_gensim_politics, keywords_tfidf_politics))
relevant_keywords_health = list(check_relevant_keywords(keywords_kmeans_health,
                                          keywords_gensim_health, keywords_tfidf_health))
relevant_keywords_sports = list(check_relevant_keywords(keywords_kmeans_sports,
                                          keywords_gensim_sports, keywords_tfidf_sports))

# Printeamos estas palabras para cada categoría
print("Politics ==> ", relevant_keywords_politics)
print("Sports ==> ", relevant_keywords_sports)
print("Health ==> ", relevant_keywords_health)

# Politics ==>  ['erc', 'presupuestos generales', 'instituciones', ... , 'alberto']
# Sports ==>  ['jugador', 'mercado', 'volver', 'escolta', ... , 'base']
# Health ==>  ['facebook', 'suicidio', 'riesgo', 'teléfonos móviles', ... , 'covid']
```

Las intersecciones completas de las *keywords* de cada clase se encuentran en la Tabla 26 del Anexo A. A partir de las palabras que aparecen en esa tabla se realiza un filtrado manual escogiendo las 30 palabras de cada categoría que entendemos más representativas. Esas palabras filtradas corresponden a los glosarios de cada clase, pudiéndose comprobar en la Tabla 5.

Tabla 5: Glosarios

Salud	Deportes	Política
plasma	firmar	desahucio
suicidios	juego	sociedad
suicidio	grupos	ley
sanidad	grupo	ayuntamiento
anticuerpos	rehabilitación	generales
fruta	temporadas	independentistas
infarto	temporada	pnv
muerte	mercado	presupuestos generales
alergia	equipo	ciudadanos
alergias	equipos	psoe
virus	estrella	pp
alimentos	jugadores	vox
teléfonos móviles	jugador	podemos
coronavirus	jugadoras	erc
expertos	entrenador	bildu
comida	balonmano	código penal
enfermedad	competir	regulación
pecho	competición	proyecto
sistema	historia	eutanasia
plato	gol	instituciones

Continúa en la próxima página

Tabla 5 – continuación de la página anterior

Salud	Deportes	Política
salud	pista	acuerdos
riesgo	franquicia	casa real
contagios	goles	vivienda
tratamiento	campeones	diputados
vacuna	campeonas	ministro
fallecidos	liga	informes
covid	lesión	congreso
ensayo	marca	eta
metabolismo	favorito	militares
sangre	nba	comunidades

4. Clasificador

Una vez creados los glosarios de cada categoria, podemos proceder a clasificar los documentos del conjunto de prueba en base a estos glosarios. Para ello, se necesita utilizar un clasificador documental. Multitud de algoritmos se han usado para esta tarea: desde los enfoques tradicionales que usaban Regresión Logística hasta enfoques más modernos que usan Redes Neuronales [4].

Es en esta sección donde analizaremos e implementaremos 3 clasificadores distintos comentando las ventajas y desventajas de cada uno, los motivos que nos llevaron a escogerlos y los resultados obtenidos. Con esto estaremos abordando el tercer hito propuesto en la Sección 1.1.

Finalmente, en la Sección 4.4 explicaremos el procedimiento seguido para mover los documentos una vez clasificados a sus carpetas correspondientes.

4.1. *Vector Space Model*

El *Vector Space Model* es uno de los tres modelos tradicionales en el campo de la Recuperación de la Información junto al Modelo Booleano y el Modelo Probabilístico. Este modelo presenta una mejora respecto al Booleano al permitir coincidencias parciales. En concreto, este modelo representa los documentos mediante vectores de pesos. En nuestro caso particular, usaremos la métrica tf-idf para los pesos de los vectores. Cómo se calcula esta métrica se puede comprobar en el ejemplo aportado en la Sección 3.2.

Este modelo hace uso de otros conceptos como el de **diccionario** y **bolsa de palabras** que describiremos en los dos próximos apartados. A su vez, para establecer las **similaridades** entre documentos se hace uso del coseno entre los vectores (se explicará en detalle en las próximas secciones).

Las principales **ventajas** y **desventajas** [1] [4] de este modelo y de usar el tf-idf como métrica son:

- **Ventajas**
  - Su esquema de ponderación de términos mejora la calidad del modelo.
  - Fácil y rápido de calcular.
  - Calcular la similaridad entre documentos se hace de forma sencilla.
- **Desventajas**
  - Los vectores son dispersos, es decir, la mayoría de posiciones son 0.
  - Asume que los términos son mutuamente independientes.
  - No captura la posición en el texto de las palabras (sintaxis).
  - No captura el significado en el texto de las palabras (semántica).
  - Si no se normaliza la métrica la longitud del documento afectará a los valores.

A pesar de lo simple y antiguo que es este modelo, sigue siendo uno de los más populares a día de hoy [1]. Es por ello que hemos decidido usarlo en nuestra tarea de clasificación de documentos. Lo que haremos será calcular la similaridad entre los documentos y cada glosario clasificando cada documento a la clase del glosario que más se asemeje. De nuevo, para la implementación de este modelo se ha usado la librería *Gensim*.

A continuación, explicaremos tres conceptos importantes de este modelo basándonos siempre en el siguiente ejemplo:

D1 = *El árbol es alto y verde.*
D2 = *El árbol es alto y viejo.*
D3 = *La planta es verde.*

4.1.1. Diccionario

El primer paso para ser capaces de representar cada documento como un vector es el de hallar un *mapping* único para cada palabra, es decir, que cada palabra tenga un único identificador. Este concepto se conoce también como creación del diccionario.

```
from gensim import corpora

d1 = "El árbol es alto y verde"
d2 = "El arbol es alto y viejo"
d3 = "La planta es verde"
documentos = [d1, d2, d3]

diccionario = corpora.Dictionary(doc for doc in documentos)

# Tiene nueve palabras únicas distintas
# diccionario = [(0, "El"), (1, "árbol"), (2, "es"), (3, "alto"), ..., (7, "La"), (8, "planta")]
```

4.1.2. Bolsa de palabras

El siguiente paso sería el de crear la bolsa de palabras o *bag of words*. Para ello lo que hacemos es convertir cada documento en un vector cambiando cada palabra por su clave en el diccionario y calculando la frecuencia en el documento. Podemos comprobar en el tercer documento cómo se pierde el orden de las palabras en el documento, como comentamos en las desventajas de este modelo.

```
# Asumiendo que continuamos el bloque de código anterior
# Clase auxiliar para calcular la bolsa de palabras
class MyCorpus:
    def __init__(self, docs, dictionary):
        self.docs = docs
        self.dict = dictionary
    def __iter__(self):
        for doc in self.docs:
            yield self.dict.doc2bow(doc)

bow = MyCorpus(documents, diccionario)
# Ahora bow tendría la siguiente forma:
# bow = [[(0,1), (1, 1), (2,1), (3, 1), (4, 1), (5, 1)],
#         ...
#        [(2,1), (5, 1), (7, 1), (8, 1)]]
```

4.1.3. Modelo

Una vez tenemos creada la bolsa de palabras y el diccionario, ya podemos crear nuestro modelo basado en la métrica tf-idf. Con esto lo que conseguiremos será expresar cada palabra en vez de con su frecuencia en el documento, con su tf-idf.

Se pueden usar diferentes variantes de esta métrica, pero como ya mostramos en la Fórmula 1, la más utilizada es

$$\text{tf-idf}_{i,j} = (1 + \log f_{i,j}) \text{ x } \log(\frac{N}{n_i})$$

A la hora de crear el modelo, *Gensim* nos permite configurar qué variante queremos usar. Por tanto, para conseguir la variante descrita arriba deberemos usar en el parámetro *smartirs* "lfc". Nótese que el último caracter hace referencia a la normalización del documento de tipo coseno, consiguiendo así eliminar el problema presentado en la última desventaja de este modelo.

```
from gensim.corpora import models
# Asumiendo que continuamos el bloque de código anterior
model_tfidf = models.TfidfModel(bow, smartirs = "lfc")

# IMPORTANTE: los valores que aparecen para el tf-idf no son reales, son inventados
# para este ejemplo
# Ahora model_tfidf tendría la siguiente forma:
# model_tfidf = [[(0, 0.2), (1, 0.2), (2, 0.1), (3, 0.2), (4, 0.2), (5, 0.2)],
#        ...
#        [(2, 0.4), (5, 0.4), (7, 0.1), (8, 0.2)]]
```

4.1.4. Semejanzas

Una vez que tenemos nuestros documentos en formato tf-idf, podemos proceder a calcular las semejanzas entre ellos. Este cálculo de semejanzas puede hacerse entre documentos pertenecientes al modelo o entre un documento perteneciente al modelo y una *query* externa. En nuestro caso estaremos interesados en la segunda forma de semejanza, siendo la *query* cada documento que queremos clasificar.

Este cálculo de semejanza se hace usando el coseno entre los dos vectores, de ahí su nombre *cosine similarity*. Tenemos un vector $\vec{q}$ en formato tf-idf que representa a la *query*; por otro lado, tenemos un vector $\vec{d}$ en formato tf-idf que representa al documento. Este esquema se puede ver en la Figura 2.

Por lo tanto, esta semejanza se calcularía con la siguiente fórmula

$$sim(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{\|\vec{q}\| \|\vec{d}\|} \tag{2}$$

donde el numerador representaría el producto escalar entre ambos vectores y el denominador es el producto de sus normas.

A modo de ejemplo, supongamos que queremos calcular la semejanza entre D1 y la query *El árbol es verde*. La representación vectorial del D1 ya la teníamos calculada arriba:

$$\vec{D1} = [(0, 0.2), (1, 0.2), (2, 0.1), (3, 0.2), (4, 0.2), (5, 0.2)]$$

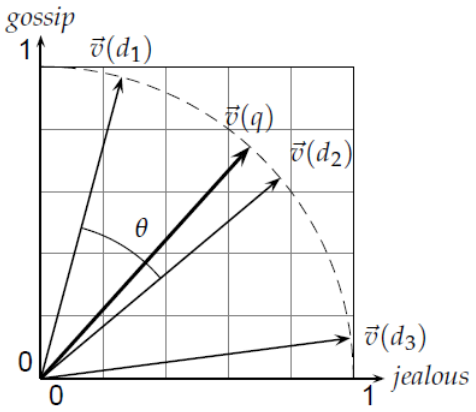

Figura 2: *Cosine similarity*: representación de los vectores. [6]

y supongamos que la *query* está representada por el vector

$$\vec{q} = [(0, 0.3), (1, 0.3), (2, 0.25), (5, 0.15)]$$

De tal forma, aplicando la Formula 2 obtendríamos de semejanza 0.74. Este proceso se hace automáticamente con la librería *Gensim*.

4.1.5. Clasificación

Una vez que sabemos cómo calcular la semejanza entre query y documentos, el proceso de clasificar documentos se simplifica. En este caso, los "documentos"serían los tres glosarios creados y las "queries"serían los documentos que queremos clasificar.

Por lo tanto, lo que debemos hacer es calcular la semejanza entre cada documento que queremos clasificar y cada glosario. Al documento se le asignará la clase correspondiente al glosario que más se asemeje.

Lo único más que necesitamos hacer sería transformar los valores de las semejanzas en probabilidades. Al tratarse las semejanzas de valores entre 0 y 1, este proceso es tan simple como dividir cada valor entre la suma de las semejanzas.

```
# supongamos que tenemos estas similaridades para un documento dado y
# los glosarios de health, sports y politics
similarities = [0.345, 0.23, 0]
probabilities = similarities / np.sum(similarities)
# probabilities = [0.652, 0.348, 0]
# por tanto se le asigna a la clase health con una probabilidad del 65%
```

Caso especial Tendríamos que prestar especial atención a estos dos casos, ya que en ambos hemos de tener cuidado con la clasificación:

- Un documento se asemeja por igual a los 3 glosarios: debemos clasificar este documento como *unknown*.
- Un documento tiene 50 % de probabilidad de ser asignado a 2 clases: debemos clasificar el documento como *unknown*.

Finalmente, en la Tabla 6, 7, 8 podemos observar las clasificaciones de los 5 primeros documentos de cada clase con este modelo. Para consultar las clasificaciones completas puede ir al Anexo B.1.

Tabla 6: Clasificación *Health* Vector Space Model (5 primeras)

doc_name	class	p_health	p_sports	p_politics	predicted_class
health_43.txt	health	**1.00**	0.00	0.00	health
health_7.txt	health	**1.00**	0.00	0.00	health
health_48.txt	health	**1.00**	0.00	0.00	health
health_3.txt	health	**1.00**	0.00	0.00	health
health_28.txt	health	**0.94**	0.06	0.00	health

Tabla 7: Clasificación Sports Vector Space Model (5 primeras)

doc_name	class	p_health	p_sports	p_politics	predicted_class
sports_49.txt	sports	0.00	**1.00**	0.00	sports
sports_30.txt	sports	0.00	**1.00**	0.00	sports
sports_43.txt	sports	0.00	**1.00**	0.00	sports
sports_24.txt	sports	0.00	**1.00**	0.00	sports
sports_37.txt	sports	0.00	**1.00**	0.00	sports

Tabla 8: Clasificación *Politics* Vector Space Model (5 primeras)

doc_name	class	p_health	p_sports	p_politics	predicted_class
politics_46.txt	politics	0.00	0.00	**1.00**	politics
politics_8.txt	politics	0.00	0.00	**1.00**	politics
politics_7.txt	politics	0.00	0.00	**1.00**	politics
politics_30.txt	politics	0.00	0.00	**1.00**	politics
politics_43.txt	politics	0.00	0.00	**1.00**	politics

4.2. Word2Vec

El segundo clasificador que decidimos probar es uno basado también en la representación de las palabras mediante vectores. Por lo tanto, todo el procedimiento (calcular similaridades, convertir en probabilidades, clasificar los documentos) será exactamente igual.

Sin embargo, esta vez no utilizaremos el modelo tf-idf para calcular los vectores. Como modelo utilizaremos el famoso *Word2Vec* creado por Mikolov et al. (2013) [8] [9]. Cómo calcula este modelo las representaciones vectoriales de las palabras lo explicaremos en la Sección 4.2.1.

Las principales ventajas y desventajas [4] de este modelo son las siguientes:

- **Ventajas**
  - Los vectores son densos.

  - Captura la posición de las palabras en el texto (sintaxis).
  - Captura el significado de las palabras (semántica).

- **Desventajas**
  - No captura el significado de las palabras en el texto (falla en la polisemia).
  - No captura palabras que se encuentran fuera del vocabulario del corpus.
  - Al ser de un modelo basado en aprendizaje profundo, necesita de un dataset de gran tamaño para funcionar bien.

Una vez expuestas las ventajas y desventajas de este modelo, suponemos que para nuestro caso particular no va a funcionar tan bien como otros clasificadores por varios motivos:

1. No tenemos un dataset de gran tamaño, necesario para entrenar este modelo.
2. Al tratarse de un clasificador con glosario, al entrenar nuestro modelo realmente no estamos capturando una relación real de las palabras; simplemente estamos capturando una relación entre palabras de una lista que normalmente no tienen por qué aparecer juntas.
3. Al no aparecer la mayoría de las palabras de los documentos en los glosarios, estamos introduciendo mucho ruido al calcular los *embeddings* (se explica en la siguiente sección).

A pesar de nuestras creencias, decidimos implementar este clasificador para comprobar si nuestra hipótesis inicial era cierta.

4.2.1. ¿Cómo funciona?

Como ya se ha comentado, este modelo también representa las palabras como vectores. Sin embargo, la forma de hacerlo es totalmente distinta a lo visto con el modelo tf-idf y fue una gran innovación en el campo del Procesamiento del Lenguaje Natural. Destacar que hemos usado la implementación de *Word2Vec* de la librería *Gensim*.

Arquitectura Para calcular las representaciones vectoriales de las palabras hace uso de una red neuronal de una capa oculta, que puede ser configurada con dos arquitecturas distintas: *Skip-gram* o *Continuous Bag of Words*. En nuestro caso, hemos decidido usar la arquitectura CBOW. Esta arquitectura trata de predecir la palabra actual dado su contexto, es decir, las palabras de a su alrededor (Véase Figura 3). El número de palabras a considerar como su contexto se puede controlar mediante el parámetro *window_size*.

Input Como input, tendrá palabras representadas en el formato *one-hot-encoding*. Es decir, cada palabra será un vector de tamaño V (siendo V el número total de palabras de nuestro vocabulario) formado por todo 0 salvo la posición en el vocabulario correspondiente a esa palabra que tendrá de valor 1.

Tamaño de los vectores Como comentamos antes, esta red neuronal solo tiene una capa oculta compuesta por D neuronas. El número de neuronas de la capa oculta marcará el tamaño de los vectores calculados.

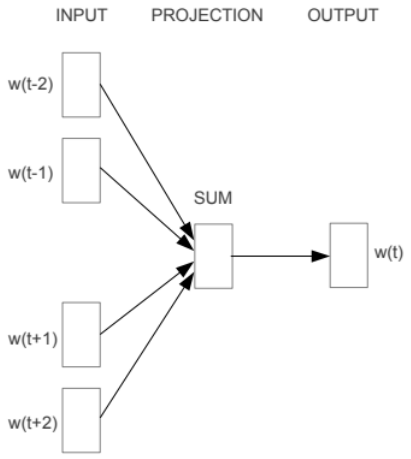

Figura 3: Arquitectura CBOW [8]

Embeddings La matriz de pesos de la capa oculta tendrá un tamaño de $V \times D$, siendo V el número de palabras de nuestro vocabulario y D el número de neuronas de la capa oculta. Si nos fijamos, con esta matriz de pesos tendríamos justo lo que queríamos: una representación vectorial de cada palabra. Por lo tanto, esta matriz de pesos de la red neuronal es lo que se conoce como *word embeddings* y será lo que utilizaremos para calcular las representaciones vectoriales de cada palabra.

Representación vectorial de un documento Una vez que tenemos calculados nuestros *word embeddings*, calcular la representación vectorial de un documento es una tarea trivial. El proceso sería el siguiente:

1. Para cada palabra que forma el documento:
   - Si esa palabra **se encuentra** en nuestro vocabulario, obtenemos directamente su representación vectorial de nuestros *embeddings*.
   - Si esa palabra **no se encuentra** es nuestro vocabulario, representamos esta palabra mediante un vector aleatorio (entre -1 y 1) de tamaño D.
2. Una vez que tenemos calculadas las representaciones vectoriales de todas las palabras del documento, la representación vectorial del documento sería la media de los vectores de sus palabras.

Cálculo de semejanzas Como comentamos al principio de esta sección, al tratarse las palabras y documentos como vectores el cálculo de semejanzas es exactamente igual que en la Sección 4.1.4.

4.2.2. Clasificación

Una vez que tenemos calculadas las semejanzas, el proceso de clasificación seguiría los mismos pasos que describimos en la Sección 4.1.5.

En la Tabla 9, 10 y 11 se puede ver la clasificación de los 5 primeros documentos de cada categoría. Las tablas completas de las clasificaciones las puede encontrar en el Anexo B.2.

Tabla 9: Clasificación *Health* Word2vec (5 primeras)

doc_name	class	p_health	p_sports	p_politics	predicted_class
health_44.txt	health	**0.92**	0.08	0.00	health
health_50.txt	health	**0.92**	0.08	0.00	health
health_29.txt	health	**0.89**	0.00	0.11	health
health_34.txt	health	**0.85**	0.15	0.00	health
health_3.txt	health	**0.82**	0.00	0.18	health

Tabla 10: Clasificación *Sports* Word2vec (5 primeras)

doc_name	class	p_health	p_sports	p_politics	predicted_class
sports_6.txt	sports	0.00	**0.95**	0.05	sports
sports_40.txt	sports	0.00	**0.94**	0.06	sports
sports_28.txt	sports	0.07	**0.93**	0.00	sports
sports_33.txt	sports	0.00	**0.88**	0.12	sports
sports_42.txt	sports	0.00	**0.85**	0.15	sports

Tabla 11: Clasificación *Politics* Word2vec (5 primeras)

doc_name	class	p_health	p_sports	p_politics	predicted_class
politics_26.txt	politics	0.04	0.00	**0.96**	politics
politics_43.txt	politics	0.00	0.12	**0.88**	politics
politics_40.txt	politics	0.17	0.00	**0.83**	politics
politics_45.txt	politics	0.00	0.18	**0.82**	politics
politics_4.txt	politics	0.00	0.18	**0.82**	politics

4.3. Bernoulli Naive Bayes

Dentro de los algoritmos de clasificación de aprendizaje automático, destacan en la clasificación de textos el uso de, entre otros, *Support Vector Machine*, *k-Nearest Neighbor* o *Naïve Bayes* [4]. Somos conscientes que para que un algoritmo de aprendizaje automático funcione bien y sea fiable es necesario tener un dataset de gran tamaño, del orden de a partir de los 10000 datos. Sin embargo, queríamos probar desde el principio cómo funcionaba un algoritmo de aprendizaje automático.

Optamos por usar un clasificador *Naïve Bayes*. Este tipo de clasificador se basa en la suposición ingenua de Bayes (o *Naïve Bayes assumption*), asumiendo que las variables predictoras -en nuestro caso las palabras- son independientes. Es decir, que la aparición de una determinada palabra en el texto no está relacionada con la aparición de cualquier otra palabra. Esta suposición está claro que es falsa en el dominio de la clasificación de textos, sin embargo, funciona muy bien en tareas de clasificación. Esta paradoja se explica en [2].

Estos clasificadores funcionan aprendiendo en el entrenamiento la probabilidad condicional de cada atributo (cada palabra) dada una clase (salud, deportes o política). Una vez entrenado, la clasificación se realiza aplicando la regla de Bayes para calcular la probabilidad de cada clase según las palabras de cada documento.

A modo de ejemplo, supongamos que queremos calcular la probabilidad de que un documento

$$\text{D1} = \textit{El Real Madrid es el mejor.}$$

pertenezca a la clase deportes. Para ello, simplemente habría que calcular la siguiente probabilidad condicionada

$$P(Deportes|"El", \ldots, "mejor") = \frac{P(Deportes) \times P("El"|Deportes) \times \cdots \times P("mejor"|Deportes)}{P("El", \ldots, "mejor")} \tag{3}$$

Nótese que todas estas probabilidades ya habrían sido aprendidas en la fase de entrenamiento por el clasificador, por lo que la predicción de la clase de un documento nuevo se resume a una multiplicación de probabilidades.

Otro aspecto a tener en cuenta es que los clasificadores bayesianos hacen uso de una distribución de probabilidad para estimar esas probabilidades anteriores. Dentro de la clasificación de textos las distribuciones más comunes son la Multinomial y la de Bernoulli. Además, se indica que para vocabularios pequeños (como es el nuestro) funciona mejor la distribución de Bernoulli [7]. Es por ello que decidimos usar esa distribución.

Por lo tanto las **ventajas** [4] [7] de usar este clasificador con esta distribución de probabilidad son:

- Funciona bien para vocabularios pequeños.
- Es fácil de implementar.
- Relativamente rápido en comparación con otros algoritmos.
- Requiere poca memoria.

Por otra parte, este método presentaría también las siguientes **desventajas**:

- Suposición de independencia.
- Como comentamos al principio, si no se tiene un dataset grande puede que no se obtengan buenos resultados.
- Es un buen clasificador, pero mal estimador. Es decir, clasifica en la clase correcta pero no estima bien las probabilidades de cada clase.

4.3.1. Entrenamiento

Como se comentó en la sección anterior, en el entrenamiento el clasificador aprende las probabilidades necesarias para poder predecir después la clase de nuevos documentos. Para la implementación de este modelo se ha usado la librería de Python *Sckikit-learn*. Entrenar este modelo es tan sencillo como se ve en el siguiente extracto de código donde hacemos uso de *TfidfVectorizer* para convertir cada documento en un vector y después entrenamos nuestro clasificador. Esta función y el modelo están integrados en una *Pipeline* por cuestiones de eficiencia.

```
1  from sklearn.feature_extraction.text import TfidfVectorizer
2  from sklearn.naive_bayes import BernoulliNB
3  from sklearn.pipeline import Pipeline
4
5  X_train = glossaries # [glosarioA, glosarioB, glosarioC]
6  y_train = [0,1,2]
7
8  clf = Pipeline([('tfidf', TfidfVectorizer()),
9                  ('clf', BernoulliNB(alpha=0.1))])
10
11 clf.fit(X_train, y_train)
```

4.3.2. Clasificación

Una vez entrenado nuestro modelo, podemos proceder a clasificar los documentos del conjunto de prueba. Para ello calculamos para cada documento la probabilidad de pertenecer a cada clase y lo clasificamos como la clase que tenga mayor probabilidad.

Al igual que en los métodos anteriores, en el caso de que un documento pudiese pertenecer a dos o más clases se le clasificaría como *unknown*.

En la Tabla 12, 13 y 14 se pueden ver las primeras 5 clasificaciones para cada categoría con este modelo. Para consultar las tablas completas vaya al Anexo B.3.

4.4. Reorganización de documentos

Una vez tenemos los documentos clasificados, sea con el modelo que sea, el proceso de reorganizar los documentos en base a su clase predicha es un proceso muy sencillo. De esta forma, estaríamos cumpliendo con el cuarto de los hitos propuestos.

Para realizar esta tarea se ha creado la siguiente función la cual nos permite dado un conjunto de documentos clasificados, conseguir el nombre del archivo y escribirlo en su fichero correspondiente.

Tabla 12: Clasificación *Health* Bayes (5 primeras)

doc_name	class	p_health	p_sports	p_politics	predicted_class
health_14.txt	health	**1.00**	0.00	0.00	health
health_17.txt	health	**1.00**	0.00	0.00	health
health_24.txt	health	**1.00**	0.00	0.00	health
health_23.txt	health	**1.00**	0.00	0.00	health
health_28.txt	health	**1.00**	0.00	0.00	health

Tabla 13: Clasificación *Sports* Bayes (5 primeras)

doc_name	class	p_health	p_sports	p_politics	predicted_class
sports_1.txt	sports	0.00	**1.00**	0.00	sports
sports_9.txt	sports	0.00	**1.00**	0.00	sports
sports_16.txt	sports	0.00	**1.00**	0.00	sports
sports_12.txt	sports	0.00	**1.00**	0.00	sports
sports_42.txt	sports	0.00	**1.00**	0.00	sports

Tabla 14: Clasificación *Politics* Bayes (5 primeras)

doc_name	class	p_health	p_sports	p_politics	predicted_class
politics_28.txt	politics	0.00	0.00	**1.00**	politics
politics_15.txt	politics	0.00	0.00	**1.00**	politics
politics_18.txt	politics	0.00	0.00	**1.00**	politics
politics_32.txt	politics	0.00	0.00	**1.00**	politics
politics_30.txt	politics	0.00	0.00	**1.00**	politics

```python
def move_files(data, tables=True):
    """
    Moves the files to their corresponding new directory after classification is done.
    """
    data = data.copy()
    classes = [[0, "p_health"], [1, "p_sports"], [2, "p_politics"]]

    for cl in classes:
        docs = data[data["current_class"] == cl[0]]
        docs = docs.sort_values(by=[cl[1]], ascending=False)
        docs["predicted_class_name"] = docs["predicted_class"].apply(lambda x: keys_dic[x])
        docs["confidence"] = docs[["p_health", "p_sports", "p_politics"]].max(axis=1)
        docs["file"] = docs.apply(lambda x: get_filename(x), axis=1)
        docs.apply(lambda row: write_file(row["file"], row["text"]), axis=1)
        if tables:
            # tabla para la memoria
            docs = docs[["doc_name", "class", "p_health",
                         "p_sports", "p_politics", "predicted_class_name"]]
            docs = docs.rename(
                columns={"predicted_class_name": "predicted_class"})
            print(docs.to_latex(bold_rows=True, float_format="%.2f",
                                column_format="llllll", index=False))
# Escribiría documentos con el siguiente formato:
# classification/health/0.500_health_40-True-health-health.txt
# classification/health/0.778_health_25-True-health-health.txt
# classification/politics/0.626_sports_32-False-sports-politics.txt
# classification/sports/0.639_sports_15-True-sports-sports.txt
# classification/unknown/0.333_sports_32-False-sports-unknown.txt
```

Nótese que al poner en primer lugar la probabilidad de clasificación ya cumplimos el quinto y último hito (ordenar los documentos por confianza de clasificación), cumpliendo así con el objetivo de la práctica. Un ejemplo de cómo quedaría la reorganización de documentos se puede ver en la Figura 4.

Figura 4: Reorganización de los documentos una vez clasificados.

5. Evaluación

Tras la implementación de los distintos clasificadores, vamos a evaluar cada uno de ellos para saber cuál es el que mejor clasifica los documentos en las tres categorías previamente nombradas. Para ello, vamos a utilizar técnicas y métodos que son habituales en la evaluación de modelos de *Machine Learning*.

5.1. Metodología

Disponemos de 50 documentos de cada una de las categorías. En la Sección 3.1 indicábamos cómo 15 de éstos eran elegidos para crear los glosarios, por lo que vamos a evaluar nuestros modelos con los 35 documentos restantes. Para realizar esta evaluación, nos hemos decido por utilizar las siguientes medidas:

Matriz de confusión Una matriz de confusión proporciona información sobre el desglose de casos evaluados correcta e incorrectamente para cada una de las clases. En nuestro caso, vamos a tener una matriz 3x3 donde podemos observar, de los documentos iniciales de cada clase que se representan en las filas, el número de documentos clasificados como cada una de las clases representadas en columnas.

Es más fácil verlo con un ejemplo, como el que se muestra en la Tabla 15.

Tabla 15: Ejemplo de matriz de confusión

	Salud	**Deportes**	**Política**
Salud	salud_como_salud	salud_como_deportes	salud_como_politica
Deportes	deporte_como_salud	deporte_como_deporte	deporte_como_politica
Política	politica_como_salud	politica_como_salud	politica_como_politica

Precisión y cobertura La precisión y la cobertura son dos medidas que se pueden utilizar para comprobar la calidad de los clasificadores. Para calcular estas dos métricas se usan los siguientes cuatro valores:

1. True Negative: Valores negativos predichos como negativos.
2. True Positive: Valores positivos predichos como positivos.
3. False Negative: Valores positivos predichos como negativos.
4. False Positive: Valores negativos predichos como positivos.

Definimos, utilizando estos 4 valores:

- **Precisión**: cuántos de los documentos clasificados en una categoría son realmente de esa categoría.

$$Precisión = \frac{TP}{TP + FP} \tag{4}$$

- **Cobertura**: cuántos de los documentos de una categoría son clasificados como esa categoría.

$$Cobertura = \frac{TP}{TP + FN} \tag{5}$$

Recordemos que en la Sección 4.1.5 exponíamos dos casos especiales en los que un documento se asemejaba por igual a los 3 glosarios o tenía la misma probabilidad de ser asignado a dos clases. Por lo tanto, los documentos que se encuentren en alguno de estos dos casos se clasificarán como *unknown* y el número total de documentos clasificados de esa categoría será menor que 35.

Hemos usado las siguientes funciones de la librería *Scikit-learn* para calcular las métricas mencionadas antes: *confusion_matrix()* y *classification_report()*

5.2. *Vector Space Model*

En la Tabla 16 podemos observar la matriz de confusión del clasificador basado en el *Vector Space Model* con tf-idf. Se puede comprobar cómo se han clasificado correctamente 30 documentos en la clase salud, 28 en deportes y 27 en política. Por otra parte, se han clasificado erróneamente 11 documentos en la clase salud (de los cuales 4 eran de deportes y 6 de política), 5 documentos en la clase deporte (de salud) y ninguno clasificado erróneamente en la clase política.

Tabla 16: Matriz de confusión: Vector Space Model

	Salud	**Deportes**	**Política**
Salud	30	5	0
Deportes	4	28	0
Política	6	0	28

A su vez, en la Figura 5 podemos observar la precisión y cobertura para cada una de las clases, obteniendo una **precisión media de 0.87** y una **cobertura media de 0.85**.

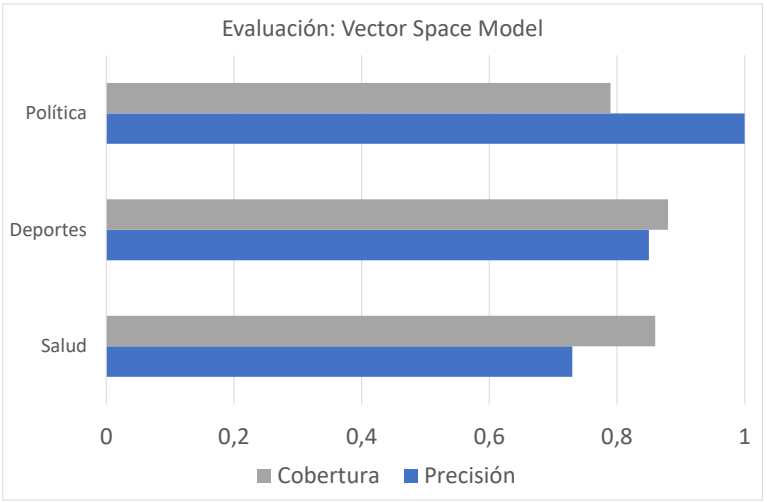

Figura 5: Precisión y cobertura: Vector Space Model

5.3. Word2Vec

En la Tabla 16 podemos observar la matriz de confusión del clasificador basado en *Word2Vec*. En este caso podemos comprobar el mal funcionamiento de este clasificador, clasificando 14 documentos de salud, 15 documentos de deportes y 14 documentos de política mal.

Tabla 17: Matriz de confusión: Word2Vec

	Salud	**Deportes**	**Política**
Salud	21	6	8
Deportes	8	20	7
Política	9	5	21

Estos malos resultados en la matriz de confusión se ven también en la gráfica de la precisión y la cobertura (Figura 6), obteniendo tan solo una **precisión media de 0.59** y una **cobertura media de 0.59**.

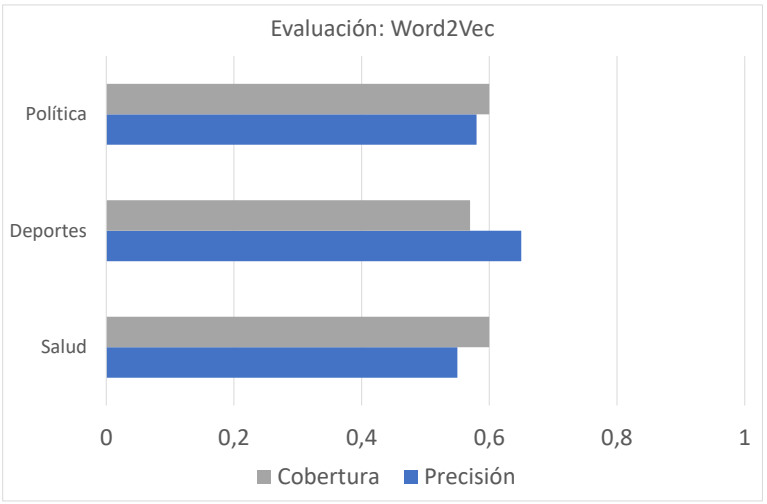

Figura 6: Precisión y cobertura: Vector Space Model

5.4. Multinomial Naive Bayes

En la Tabla 18 podemos observar la matriz de confusión del clasificador bayesiano. Este clasificador es el que funciona mejor de todos, pudiendo ver que tan solo clasifica mal 6 documentos de salud, 1 de deportes y 5 de política.

Tabla 18: Matriz de confusión: Bernoulli Naive Bayes

	Salud	**Deportes**	**Política**
Salud	29	5	1
Deportes	0	34	1
Política	5	0	27

También podemos ver en la Figura 7 cómo los valores de precisión y cobertura son muy altos para todas las clases, obteniendo una **precisión media de 0.89** y una **cobertura media de 0.88**, las mejores de los tres clasificadores probados.

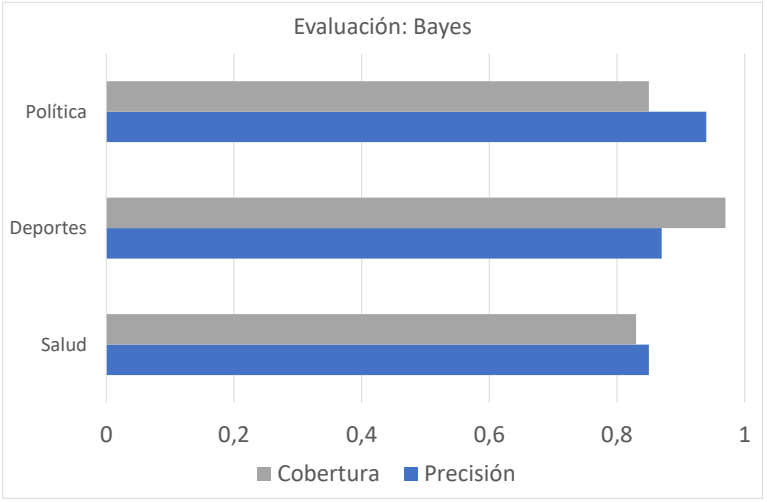

Figura 7: Precisión y cobertura: Bernoulli Naive Bayes

6. Ampliación

Viendo los prometedores resultados obtenidos de la clasificación anterior, especialmente con TF-IDF y Bayes, nos surgió la duda de cuan buenos podrían ser estos clasificadores enfrentándonos a situaciones algo más complicadas, ya que en un entorno real rara es la vez en el que las categorías de los documentos son tan disjuntas. Para ello, hemos decidido ampliar el número de documentos e introducir una nueva categoría: tecnología. Siguiendo el mismo proceso que hemos realizado para la clasificación con tres categorías, vamos a dar breves pinceladas sobre cambios que hemos realizado (voluntariamente) y cambios que hemos podido observar (consecuencia de la agregación de esta categoría).

Obtención de documentos Para la obtención de los documentos, además de algunos diarios digitales que mencionamos en la Sección 2.1 como pueden ser El Confidencial o El País, buscamos para esta nueva categoría sitos web especializados en tecnología, añadiendo las siguientes tres fuentes para la obtención de documentos:

1. Computer Hoy
2. Silicon.es
3. Byte TI

Creación de los glosarios A la hora de crear los glosarios mediante los métodos indicados en la Sección 3, nos topamos con un empeoramiento considerable en la calidad de las keywords generadas mediante el método K-Means, especialmente en el de salud. En la Tabla 19 se pueden observar los primeros 10 términos de cada clase obtenidos mediante el método K-Means para las cuatro categorías.

Tabla 19: Keywords KMeans - Ampliado (10 primeras)

Salud	**Deportes**	**Política**	**Tecnología**
redes	jugador	gobierno	nasa
facebook	liga	sánchez	spacex
sociales	atlético	españa	luna
mundo	jordan	erc	espacial
forma	nba	nacional	artemis
redes sociales	vaccaro	pedro sánchez	chinook
infarto	temporada	pedro	helicóptero
tuberculosis	gasol	presidente	superficie
usuarios	contrato	fiscal	modelo
contraseñas	lakers	ayuso	corazón

Glosario agregado El problema expresado en el párrafo anterior no va a tener realmente demasiada relevancia, ya que al agregar todas las keywords de los diferentes métodos mediante la intersección, aquellos que puedan ser considerados como keywords no relevantes de esa clase, como ocurría con las keywords *usuarios* y *contraseñas* en referencia a salud, no serán obtenidos como resultado de la intersección. Lo que sí que nos vimos obligados a hacer es aumentar el número de keywords obtenidas para cada una de las clases, ya que al hacer la intersección no

obteníamos suficientes keywords como para poder generar los glosarios de la misma forma que los generábamos en la Sección 3.5.

Clasificadores La clasificación de los documentos se va a hacer ahora teniendo en cuenta esta nueva clase. De esta forma, podemos observar cómo varia la evaluación de los clasificadores al haber añadido una 4ª categoría.

6.1. TFIDF

En la Tabla 20 podemos ver la matriz de confusión del clasificador basado en el *Vector Space Model* incluyendo la nueva categoría de tecnología.

Tabla 20: Ampliación - Matriz de confusión: Vector Space Model

	Salud	**Deportes**	**Política**	**Tecnología**
Salud	30	4	0	1
Deportes	4	27	1	0
Política	4	0	30	0
Tecnología	4	5	0	26

En este caso se obtiene una **precisión media de 0.85** y una **cobertura media de 0.83**.

6.2. Word2Vec

En la Tabla 21 podemos ver la matriz de confusión del clasificador basado en *Word2Vec* incluyendo la nueva categoría de tecnología. Se puede comprobar cómo también al incluir una nueva categoría sigue funcionando mal, incluso peor que con solo 3 categorías.

Tabla 21: Ampliación - Matriz de confusión: Word2Vec

	Salud	**Deportes**	**Política**	**Tecnología**
Salud	17	2	7	9
Deportes	8	15	7	5
Política	4	7	17	7
Tecnología	8	8	3	16

En este caso se obtiene una **precisión media de 0.47** y una **cobertura media de 0.46**.

6.3. Bernoulli Naive Bayes

En la Tabla 22 podemos ver la matriz de confusión del clasificador bayesiano incluyendo la nueva categoría de tecnología. Se puede ver cómo en este caso parece clasificar peor los documentos que el clasificador basado en el *Vector Space Model* con tf-idf.

Se obtiene una **precisión media de 0.80** y una **cobertura media de 0.79**.

Tabla 22: Ampliación - Matriz de confusión: Bernoulli Naive Bayes

	Salud	**Deportes**	**Política**	**Tecnología**
Salud	24	5	2	4
Deportes	0	28	2	5
Política	3	1	29	2
Tecnología	0	5	0	30

7. Conclusiones

Una vez presentados los resultados obtenidos con nuestros clasificadores, tanto para la versión original como para la ampliación, podemos sacar las siguientes conclusiones:

- Respecto a los **glosarios**, creemos que nuestro método presenta una forma rápida y semi automática de crearlos. Sería interesante contemplar la incorporación de determinadas palabras que los expertos supiesen que son representativas de cada clase, a cambio de reducir la automatización del proceso.
- Sobre **Word2Vec**, nuestra hipótesis de que no funcionaría tan bien parece ser cierta. Creemos que esto es debido fundamentalmente al hecho de que el clasificador sea con glosario. Sería muy interesante probar este modelo en un clasificador sin glosario para comprobar su funcionamiento y ver si realmente el problema era el usar un glosario.
- El **clasificador bayesiano** funciona muy bien a la hora de clasificar los documentos en 3 categorías a pesar del pequeño tamaño del dataset. Sin embargo, a la hora de ampliar el dataset con una cuarta categoría podemos ver cómo empeora su funcionamiento.
- Respecto al **VSM con tf-idf** podemos ver cómo es superado por el clasificador bayesiano por apenas unas centésimas a la hora de clasificar los documentos en 3 categorías. Sin embargo, al ampliar el dataset con la categoría tecnología funciona mejor que el clasificador bayesiano.
- Por lo tanto, podemos concluir que el **clasificador basado en el *Vector Space Model*** es el que mejor funciona de todos.
- También sería muy interesante poder probar estos métodos de clasificación en un sistema real con miles de documentos, donde pudiésemos ver el funcionamiento real de éstos.

Referencias

[1] R. Baeza-Yates y B. Ribeiro-Neto, "Modern information retrieval: the concepts and technology behind search," *Choice Reviews Online*, vol. 48, n.º 12, págs. 48–6950–48-6950, 2011, ISSN: 0009-4978. DOI: 10.5860/choice.48-6950.

[2] N. Friedman, D. Geiger y M. Goldszmidt, "Bayesian Network Classifiers," *Machine Learning*, vol. 29, n.º 2-3, págs. 131-163, 1997, ISSN: 08856125. DOI: 10.1023/a:1007465528199. dirección: https://link.springer.com/article/10.1023/A:1007465528199.

[3] V. Goel, *Applying Machine Learning to classify an unsupervised text document*, https://towardsdatascience.com/applying-machine-learning-to-classify-an-unsupervised-text-document-e7bb6265f52, 2018. (visitado 14-12-2020).

[4] K. Kowsari, K. J. Meimandi, M. Heidarysafa, S. Mendu, L. Barnes y D. Brown, *Text classification algorithms: A survey*, 2019. DOI: 10.3390/info10040150. arXiv: 1904.08067.

[5] J. MacQueen, "Some methods for classification and analysis of multivariate observations," inf. téc., 1967, págs. 281-296.

[6] C. D. Manning, P. Raghavan y H. Schütze, *Introduction to information retrieval*. Cambridge University Press, 2008, pág. 482, ISBN: 9780511809071.

[7] A. McCallum, K. Nigam y col., "A comparison of event models for naive bayes text classification," en *AAAI-98 workshop on learning for text categorization*, Citeseer, vol. 752, 1998, págs. 41-48.

[8] T. Mikolov, K. Chen, G. Corrado y J. Dean, "Efficient estimation of word representations in vector space," inf. téc., 2013. arXiv: 1301.3781. dirección: http://ronan.collobert.com/senna/.

[9] T. Mikolov, I. Sutskever, K. Chen, G. Corrado y J. Dean, "Distributed representations of-words and phrases and their compositionality," *Advances in Neural Information Processing Systems*, oct. de 2013, ISSN: 10495258. arXiv: 1310.4546. dirección: http://arxiv.org/abs/1310.4546.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot y E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, págs. 2825-2830, 2011.

[11] Real Academia Española, *Diccionario de la lengua española*, 2020. dirección: https://dle.rae.es.

[12] R. Řehůřek y P. Sojka, "Software Framework for Topic Modelling with Large Corpora," English, en *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta: ELRA, mayo de 2010, págs. 45-50.

[13] G. Salton y C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing and Management*, vol. 24, n.º 5, págs. 513-523, ene. de 1988, ISSN: 03064573. DOI: 10.1016/0306-4573(88)90021-0.

[14] N. Saxena, *Extracting Keyphrases from Text: RAKE and Gensim in Python*, https://towardsdatascience.com/extracting-keyphrases-from-text-rake-and-gensim-in-python-eefd0fad582f, 2020. (visitado 14-12-2020).

[15] G. Van Rossum y F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009, ISBN: 1441412697.

A. Keywords

Tabla 23: Keywords TF-IDF

Salud	Deportes	Política
tuberculosis	gasol	marín
teléfonos	vaccaro	ayuntamiento
plasma	stéfano	vivienda
móviles	haaland	supremo
metabolismo	balonmano	sociedad
miocardio	alfredo	dsn
teléfonos móviles	di stéfano	informes
moderna	candidato	manifestaciones
infarto	favorito	notas
facebook	jornet	seguridad nacional
pecho	pista	ministro
contagios	curry	desahucio
anticuerpos	thompson	felipe gonzález
las redes	warriors	territorio
redes	lakers	armonización fiscal
alergia	nike	mal
magra	bla	feministas
masa	campazzo	las manifestaciones
masa magra	carro	este jueves
hiperinmune	nacional	utrera
al día	campeones	propuesto
angina	marc	gonzález
plato	bernabéu	eutanasia
voluntarios	atlético	interés para
personalidad	dortmund	navidades
castilla	ha sido	información
fallecidos	barça	pnv
mes	grupo	europa
senegal	rehabilitación	exterior
quizá	tipo	azurmendi
donar	michael jordan	becerril
suicidio	bolt	jiménez becerril
suicidios	del mundo	callar
metabólica	competir	iceta
salud mental	duro	don juan
sociales	ricky	elena
algún tipo	facu	rey juan
tasa	jode	los hechos
sistema	laso	rodríguez
por ejemplo	me jode	convocatoria
león	abadía	gobierno central
enfermera	jesús	isabel díaz

Continúa en la próxima página

Tabla 23 – continuación de la página anterior

Salud	Deportes	Política
ensayo	obstaculista	armadas
inyección	rojo	defensa nacional
placebo	récord nacional	fuerzas armadas
se produce	kilómetro	loyola
desayuno	abdi	militares
apostar	ademar	antúnez
apostar por	leonés	municipal
hambre	marcador	estas navidades
cardíaca	adrian	planes
fritas	adrian wojnarowski	profesionales
fruta	orleans	verano
donación	wojnarowski	código
donantes	dinamarca	código penal
belleza	federación	injurias
tinción	grupos	ha explicado
utilización	herning	madrileños
escuela	kolding	atención
idealizada	principal	casa real
usuarios	ronda principal	ha advertido
comenzado	sede	armonización
dakar	equipos	regulación
desconocida	mcgee	texto
las autoridades	pasada temporada	pedro sánchez
ocho días	huella	interés
inmunitario	leyenda	acuerdos con
sistema inmunitario	real madrid	eh bildu
sueño	saunders	pablo iglesias
tu sistema	situaciones	alberto
microbios	wolves	ascen
patógenos	gol	presupuestos generales
se encuentran	gran pregunta	enfrentar
sistema inmune	distancias	haga
sin embargo	reto	haga falta
aumento del	lesión	ir mejor
padres	hablar	mejorar
preguntas	entrenador	puede enfrentar
energía	all star	recordado
tu tasa	ha promediado	fiesta
tu cuerpo	han llegado	fiesta nacional
alergias	ni siquiera	tensión
alimentaria	por debajo	decisiones pesc
científicos	como obligación	delcy
acumulada	obligación	delcy rodríguez
asturias	pelearla	las obligaciones
baleares	sobre todo	obligaciones
canarias	tiro	pesc

Continúa en la próxima página

Tabla 23 – continuación de la página anterior

Salud	Deportes	Política
incidencia acumulada	dellavedova	política exterior
notificados	detroit	querellas
sanidad	espn	territorio español
última semana	esta manera	tribunal supremo
clínicos	ha informado	penal
dosis	heat	tribunal
miami	informado	decisiones
padre	nueva orleans	activó
participar	pelicans	arcas
pfizer	pistons	financiero
expertos	NaN	NaN
parte del	NaN	NaN

Tabla 24: Keywords Gensim

Salud	Deportes	Política
enfermedad	equipo	gobierno
enfermedades	equipos	sanchez
caso	jugador	socialista pedro
estudio	jugadores	presupuestos
estudios	madrid	presupuesto
comidas	temporada	comunidad
comida	temporadas	comunidades
forma	grupos	nacional
formas	por	nacion
salud	marca	familia
saludable	marcas	familias
saludables	base	coronavirus madrid
cuerpo	liga	psoe
telefono	atletico	erc
telefonos	goles	politica
frente	carrera	politicas
infarto	carreras	euros horas
infartos	dificil	vox
sangre	dificiles	ley
muerte	llegado nba	acuerdo
muertes	juego	acuerdos
efectos	record	socialistas
efecto	records	rey felipe
covid	pivot	ayuntamiento
puedes	pivots	vivienda
puede	pivote	congreso
sistema	historia	cuentas caso
sistemas	final	diputados

Continúa en la próxima página

Tabla 24 – continuación de la página anterior

Salud	Deportes	Política
virus	jordan	diputado
vacuna	stefano	vicepresidente pablo iglesias
importantes vacunas	futbol	politico
importante	contrato millones	politicos
alimentos	mercado	este
alimento	lesion	espanol
pandemias	lesiones	espanoles
tuberculosis	escolta	vidas
riesgo	escoltas	sanitarias
por sobre	finales champions	sanitaria
datos	noruega	comision
dato	noruegas	hijos
anticuerpos	haaland	hijo
medicos	kilometros	independentistas
medico	kilometro	independentista
vida	gol minuto jugar	reales
plasma	jornet	ejecutivo
corazon	registro	pnv
grasas	registros	pais vida institucional
grasa	competicion	informes
partes	pregunta	informe
parte	preguntas	claro
numero	vaccaro	andalucia
sintomas	distancia	jueves
proceso	rondas	covid momentos
procesos	minutos	ministros
coronavirus	entrenamientos distancias	ministro
unidos	franquicias	vicepresidentes
unido	franquicia	desahucio
fruta	estrella	desahucios
frutas	estrellas	impuestos
tipo	entrenador	impuesto
tipos	gasol	instituciones
pacientes	bajas gente	institucion
paciente	debajo	derecho
moderna grupos	situaciones	derechos
movil	volver	fiscal
moviles	mundo	fiscales
autentica	unico	militar
autenticas	unicos	militares
mucho	primeras	proyecto
muchos	primera	bildu
produce	club	madrilenos
expertos	balonmano	madrileno
experto	seleccion ronda	noticias
mundo dietas	pasa	noticia

Continúa en la próxima página

Tabla 24 – continuación de la página anterior

Salud	Deportes	Política
ejemplo	pistas entrenamiento	generales
ejemplos	agente	notas
jovenes	agentes	nota
joven	lakers	seguridad
metabolismo	recien salida	padre
paises	jornada	padres
contagio	jornadas	moncloa
contagios	montana	unidas
tratamiento	puesto	isabel
tratamientos	esta	social
bacterias	prueba	sociales
bacteria	pruebas	sanidad
pais	cambio	casado
probable	cambios	jose
cantidad	maxima	diaz ayuso
cantidades	maximas	informaciones
modernas	alfredo	informacion
centro	titulos	instalaciones
centros	titulo	instalacion
dieta	relacion	eta
realidad	hombre	delitos
plato	pista	delito
platos	hablando	presidenta
gente	realmente	dsn
problemas	firmar	momento
problema	convirtio	fuerza
general	segunda	fuerzas
arterias	ritmo	fuentes
arteria	todas	fuente
cifra	davis	sentido asegurado
cifras	selecciones	defensa
ensayos	fichajes	navidades
ensayo	fichaje	navidad
investigacion	esfuerzo	ciudadanos
investigaciones	esfuerzos	jimenez
revista	metros	zonas
revistas	queda	zona
opcion	noruego	sanitario
opciones	deporte	sanitarios
alergia	atletas noruegos	muy
alergias	baja	NaN
mas	atleta	NaN
tasas	NaN	NaN
tasa	NaN	NaN
manos	NaN	NaN
mano	NaN	NaN

Continúa en la próxima página

Tabla 24 – continuación de la página anterior

Salud	Deportes	Política
cardiaco	NaN	NaN
suicidio	NaN	NaN
suicidios	NaN	NaN
facebook	NaN	NaN
musculo	NaN	NaN
musculos	NaN	NaN
hospital	NaN	NaN
sal	NaN	NaN
estomago	NaN	NaN
malos	NaN	NaN
malo	NaN	NaN
vitamina	NaN	NaN
vitaminas	NaN	NaN
investigadores	NaN	NaN
demasiado	NaN	NaN
elementos	NaN	NaN
elemento	NaN	NaN
periodo	NaN	NaN
inmune	NaN	NaN
ejercicios	NaN	NaN
ejercicio	NaN	NaN

Tabla 25: Keywords KMeans

Salud	Deportes	Política
día	equipo	erc
enfermedad	jugador	marín
covid	liga	ley
nacional	jordan	presupuestos
horas	nba	felipe
pandemia	temporada	bildu
personas	goles	gonzález
tuberculosis	vaccaro	proyecto
grupo	gasol	sistema
días	contrato	jueves
teléfonos	base	presupuestos generales
coronavirus	millones	generales
casos	metabolismo	felipe gonzález
ayuso	juego	erc bildu
comunidad	lakers	acuerdos
virus	haaland	andalucía
récord	facebook	sociedad
plasma	cuerpo	independentistas
comida	mercado	congreso

Continúa en la próxima página

Tabla 25 – continuación de la página anterior

Salud	Deportes	Política
móviles	auténtica	vicepresidente
sangre	año	euros
contagios	entrenador	indicado
madrileños	jugar	claro
número	hombre	socialista
tipo	escolta	iglesias
semana	competir	comunidades
vacuna	duro	españoles
seguridad	atlético	vox
stéfano	puntos	eutanasia
anticuerpos	masa	eta
salud	volver	quizá
ola	pívot	diputados
país	competición	autónomas
vivienda	nike	comunidades autónomas
ayuntamiento	magra	armonización fiscal
teléfonos móviles	masa magra	armonización
defensa	gente	presidente gobierno
díaz	raptors	callar
sanidad	lesión	iceta
marca	redes sociales	financiación
europa	temporadas	pablo
balonmano	curry	proetarras
familia	warriors	azurmendi
centro	thompson	jiménez becerril
riesgo	jugadores	becerril
caso	franquicia	mal
alergia	gol	pablo iglesias
díaz ayuso	suns	pge
importante	realmente	pnv
situación	redes	jiménez
miocardio	investigadores	izquierda
corazón	energía	jefe
política	recién	inmunitario
frente	champions	sueño
moderna	campazzo	sistema inmunitario
principal	ricky	ciudadanos
supremo	the	código
muertes	marc	injurias
jóvenes	campeones	código penal
juan	llegado	ascen
juan carlos	convirtió	regulación
estudios	tasa	malo
comisión	dólares	atención
media	vida	junta
alfredo	bla	separatistas

Continúa en la próxima página

Tabla 25 – continuación de la página anterior

Salud	Deportes	Política
impuestos	metabólica	manda callar
pecho	tasa metabólica	afecta
alimentos	sociales	generales pge
síntomas	tantos	casa
tratamiento	pasar	esquerra
ministro	personalidad	republicana
jornet	dortmund	esquerra republicana
pista	hablando	ocasiones
decisión	difícil	lengua
medidas	fútbol	penal
noviembre	sal	banderas
seguridad nacional	michael jordan	casa real
dsn	pasa	obligado
informes	draft	cosa
expertos	firmar	medicamentos
fallecidos	partidos	alberto
ejecutivo	historia	barrios
países	acuerdo	líder

Tabla 26: Keywords Agregadas

Salud	Deportes	Política
vacuna	alfredo	marín
tipo	mercado	azurmendi
contagios	grupos	navidades
frente	jornet	independentistas
anticuerpos	goles	ayuntamiento
alimentos	lesión	vivienda
moderna	campeones	regulación
importante	temporada	penal
riesgo	estrella	comunidades
alergias	bla	código penal
sanidad	historia	vox
estudios	pasa	dsn
plato	barça	notas
muertes	campazzo	acuerdos
plasma	hombre	desahucio
tuberculosis	escolta	ley
expertos	temporadas	felipe gonzález
suicidios	liga	congreso
comida	realmente	diputados
centro	equipos	callar
teléfonos móviles	vaccaro	casa real
enfermedad	gasol	sociedad

Continúa en la próxima página

Tabla 26 – continuación de la página anterior

Salud	Deportes	Política
suicidio	lakers	eta
fallecidos	entrenador	ciudadanos
coronavirus	pista	erc
infarto	gol	texto
pecho	equipo	ascen
facebook	hablando	proyecto
virus	marc	alberto
tasa	michael jordan	ministro
teléfonos	volver	militares
fruta	bolt	becerril
caso	thompson	iceta
tratamiento	duro	presupuestos generales
ensayo	base	pablo iglesias
covid	franquicia	informes
miocardio	haaland	injurias
móviles	curry	jueves
metabolismo	firmar	armonización
sangre	jugadores	atención
salud	dortmund	armonización fiscal
sistema	competir	claro
alergia	juego	gonzález

B. Tablas de clasificación

B.1. TF-IDF

Tabla 27: Clasificación *Health* Vector Space Model

doc_name	class	p_health	p_sports	p_politics	predicted_class
health_43.txt	health	**1.00**	0.00	0.00	health
health_7.txt	health	**1.00**	0.00	0.00	health
health_48.txt	health	**1.00**	0.00	0.00	health
health_3.txt	health	**1.00**	0.00	0.00	health
health_28.txt	health	**0.94**	0.06	0.00	health
health_24.txt	health	**0.90**	0.07	0.03	health
health_47.txt	health	**0.88**	0.12	0.00	health
health_49.txt	health	**0.88**	0.00	0.12	health
health_41.txt	health	**0.88**	0.12	0.00	health
health_6.txt	health	**0.88**	0.12	0.00	health
health_35.txt	health	**0.87**	0.13	0.00	health
health_45.txt	health	**0.85**	0.15	0.00	health
health_23.txt	health	**0.85**	0.15	0.00	health
health_34.txt	health	**0.84**	0.16	0.00	health
health_46.txt	health	**0.82**	0.00	0.18	health
health_8.txt	health	**0.81**	0.00	0.19	health
health_25.txt	health	**0.78**	0.11	0.11	health
health_39.txt	health	**0.77**	0.23	0.00	health
health_32.txt	health	**0.76**	0.24	0.00	health
health_9.txt	health	**0.72**	0.28	0.00	health
health_30.txt	health	**0.71**	0.03	0.26	health
health_36.txt	health	**0.71**	0.00	0.29	health
health_42.txt	health	**0.70**	0.30	0.00	health
health_31.txt	health	**0.69**	0.10	0.21	health
health_50.txt	health	**0.67**	0.33	0.00	health
health_26.txt	health	**0.67**	0.00	0.33	health
health_44.txt	health	**0.67**	0.33	0.00	health
health_5.txt	health	**0.62**	0.09	0.28	health
health_29.txt	health	**0.57**	0.25	0.18	health
health_40.txt	health	**0.50**	0.00	0.50	health
health_27.txt	health	0.47	**0.53**	0.00	sports
health_4.txt	health	0.25	**0.50**	0.25	sports
health_38.txt	health	0.22	**0.56**	0.22	sports
health_37.txt	health	0.00	**1.00**	0.00	sports
health_33.txt	health	0.00	**1.00**	0.00	sports

Tabla 28: Clasificación *Sports* TF-IDF

doc_name	class	p_health	p_sports	p_politics	predicted_class
sports_49.txt	sports	0.00	**1.00**	0.00	sports
sports_30.txt	sports	0.00	**1.00**	0.00	sports
sports_43.txt	sports	0.00	**1.00**	0.00	sports
sports_24.txt	sports	0.00	**1.00**	0.00	sports
sports_37.txt	sports	0.00	**1.00**	0.00	sports
sports_34.txt	sports	0.00	**1.00**	0.00	sports
sports_33.txt	sports	0.00	**1.00**	0.00	sports
sports_5.txt	sports	0.00	**1.00**	0.00	sports
sports_41.txt	sports	0.00	**1.00**	0.00	sports
sports_3.txt	sports	0.00	**1.00**	0.00	sports
sports_6.txt	sports	0.00	**1.00**	0.00	sports
sports_28.txt	sports	0.00	**1.00**	0.00	sports
sports_7.txt	sports	0.00	**1.00**	0.00	sports
sports_26.txt	sports	0.00	**1.00**	0.00	sports
sports_8.txt	sports	0.00	**1.00**	0.00	sports
sports_47.txt	sports	0.03	**0.97**	0.00	sports
sports_9.txt	sports	0.08	**0.92**	0.00	sports
sports_46.txt	sports	0.08	**0.92**	0.00	sports
sports_42.txt	sports	0.04	**0.88**	0.08	sports
sports_40.txt	sports	0.13	**0.87**	0.00	sports
sports_44.txt	sports	0.17	**0.83**	0.00	sports
sports_23.txt	sports	0.20	**0.80**	0.00	sports
sports_4.txt	sports	0.20	**0.80**	0.00	sports
sports_29.txt	sports	0.20	**0.80**	0.00	sports
sports_45.txt	sports	0.25	**0.75**	0.00	sports
sports_38.txt	sports	0.00	**0.67**	0.33	sports
sports_50.txt	sports	0.36	**0.64**	0.00	sports
sports_36.txt	sports	0.50	0.50	0.00	health
sports_48.txt	sports	**0.53**	0.47	0.00	health
sports_31.txt	sports	**0.53**	0.47	0.00	health
sports_32.txt	sports	0.33	0.33	0.33	unknown
sports_27.txt	sports	0.33	0.33	0.33	unknown
sports_39.txt	sports	0.33	0.33	0.33	unknown
sports_35.txt	sports	0.00	0.00	**1.00**	politics
sports_25.txt	sports	**1.00**	0.00	0.00	health

Tabla 29: Clasificación *Politics* TF-IDF

doc_name	class	p_health	p_sports	p_politics	predicted_class
politics_46.txt	politics	0.00	0.00	**1.00**	politics
politics_8.txt	politics	0.00	0.00	**1.00**	politics
politics_7.txt	politics	0.00	0.00	**1.00**	politics
politics_30.txt	politics	0.00	0.00	**1.00**	politics
politics_43.txt	politics	0.00	0.00	**1.00**	politics
politics_45.txt	politics	0.00	0.00	**1.00**	politics
politics_24.txt	politics	0.04	0.00	**0.96**	politics
politics_27.txt	politics	0.05	0.00	**0.95**	politics
politics_4.txt	politics	0.00	0.10	**0.90**	politics
politics_28.txt	politics	0.00	0.10	**0.90**	politics
politics_32.txt	politics	0.13	0.00	**0.87**	politics
politics_26.txt	politics	0.08	0.08	**0.84**	politics
politics_5.txt	politics	0.20	0.00	**0.80**	politics
politics_48.txt	politics	0.04	0.18	**0.78**	politics
politics_41.txt	politics	0.00	0.22	**0.78**	politics
politics_23.txt	politics	0.24	0.00	**0.76**	politics
politics_33.txt	politics	0.03	0.21	**0.76**	politics
politics_36.txt	politics	0.04	0.27	**0.69**	politics
politics_34.txt	politics	0.17	0.14	**0.69**	politics
politics_39.txt	politics	0.22	0.11	**0.67**	politics
politics_47.txt	politics	0.17	0.17	**0.67**	politics
politics_25.txt	politics	0.17	0.17	**0.67**	politics
politics_9.txt	politics	0.26	0.09	**0.66**	politics
politics_37.txt	politics	0.00	0.35	**0.65**	politics
politics_40.txt	politics	0.06	0.30	**0.64**	politics
politics_42.txt	politics	0.00	0.43	**0.57**	politics
politics_35.txt	politics	0.00	0.45	**0.55**	politics
politics_38.txt	politics	0.29	0.21	**0.51**	politics
politics_29.txt	politics	0.35	0.17	**0.48**	politics
politics_44.txt	politics	0.36	0.17	**0.48**	politics
politics_50.txt	politics	**0.53**	0.00	0.47	health
politics_3.txt	politics	**0.56**	0.00	0.44	health
politics_49.txt	politics	0.33	0.33	0.33	unknown
politics_31.txt	politics	**0.71**	0.00	0.29	health
politics_6.txt	politics	**0.70**	0.05	0.25	health

B.2. Word2vec

Tabla 30: Clasificación *Health* Word2vec

doc_name	class	p_health	p_sports	p_politics	predicted_class
health_44.txt	health	**0.92**	0.08	0.00	health
health_50.txt	health	**0.92**	0.08	0.00	health
health_29.txt	health	**0.89**	0.00	0.11	health
health_34.txt	health	**0.85**	0.15	0.00	health
health_3.txt	health	**0.82**	0.00	0.18	health
health_24.txt	health	**0.79**	0.00	0.21	health
health_35.txt	health	**0.78**	0.22	0.00	health
health_6.txt	health	**0.76**	0.24	0.00	health
health_23.txt	health	**0.74**	0.00	0.26	health
health_41.txt	health	**0.74**	0.26	0.00	health
health_42.txt	health	**0.73**	0.27	0.00	health
health_46.txt	health	**0.69**	0.31	0.00	health
health_30.txt	health	**0.68**	0.32	0.00	health
health_9.txt	health	**0.68**	0.32	0.00	health
health_36.txt	health	**0.67**	0.00	0.33	health
health_28.txt	health	**0.62**	0.00	0.38	health
health_33.txt	health	**0.56**	0.00	0.44	health
health_49.txt	health	**0.56**	0.00	0.44	health
health_39.txt	health	**0.54**	0.00	0.46	health
health_43.txt	health	**0.53**	0.00	0.47	health
health_4.txt	health	0.47	**0.53**	0.00	sports
health_7.txt	health	0.43	0.00	**0.57**	politics
health_25.txt	health	0.41	**0.59**	0.00	sports
health_5.txt	health	0.39	**0.61**	0.00	sports
health_48.txt	health	0.32	**0.68**	0.00	sports
health_40.txt	health	0.29	**0.71**	0.00	sports
health_8.txt	health	0.23	**0.77**	0.00	sports
health_32.txt	health	0.19	0.00	**0.81**	politics
health_31.txt	health	0.00	**0.92**	0.08	sports
health_37.txt	health	0.00	0.43	**0.57**	politics
health_27.txt	health	0.00	**0.81**	0.19	sports
health_47.txt	health	0.00	0.04	**0.96**	politics
health_38.txt	health	0.00	0.32	**0.68**	politics
health_26.txt	health	0.00	0.48	**0.52**	politics
health_45.txt	health	0.00	0.35	**0.65**	politics

Tabla 31: Clasificación *Sports* Word2vec

doc_name	class	p_health	p_sports	p_politics	predicted_class
sports_6.txt	sports	0.00	**0.95**	0.05	sports
sports_40.txt	sports	0.00	**0.94**	0.06	sports
sports_28.txt	sports	0.07	**0.93**	0.00	sports
sports_33.txt	sports	0.00	**0.88**	0.12	sports
sports_42.txt	sports	0.00	**0.85**	0.15	sports
sports_23.txt	sports	0.00	**0.83**	0.17	sports
sports_29.txt	sports	0.00	**0.81**	0.19	sports
sports_34.txt	sports	0.00	**0.80**	0.20	sports
sports_50.txt	sports	0.20	**0.80**	0.00	sports
sports_8.txt	sports	0.20	**0.80**	0.00	sports
sports_5.txt	sports	0.21	**0.79**	0.00	sports
sports_48.txt	sports	0.21	**0.79**	0.00	sports
sports_49.txt	sports	0.26	**0.74**	0.00	sports
sports_46.txt	sports	0.30	**0.70**	0.00	sports
sports_43.txt	sports	0.00	**0.68**	0.32	sports
sports_47.txt	sports	0.00	**0.66**	0.34	sports
sports_9.txt	sports	0.00	**0.66**	0.34	sports
sports_27.txt	sports	0.37	**0.63**	0.00	sports
sports_30.txt	sports	0.00	**0.62**	0.38	sports
sports_26.txt	sports	0.00	**0.56**	0.44	sports
sports_24.txt	sports	0.00	**0.54**	0.46	sports
sports_45.txt	sports	0.46	**0.54**	0.00	sports
sports_4.txt	sports	0.00	**0.53**	0.47	sports
sports_44.txt	sports	0.00	**0.51**	0.49	sports
sports_7.txt	sports	0.00	0.26	**0.74**	politics
sports_25.txt	sports	0.00	0.22	**0.78**	politics
sports_31.txt	sports	**0.81**	0.19	0.00	health
sports_38.txt	sports	0.00	0.03	**0.97**	politics
sports_3.txt	sports	**0.79**	0.00	0.21	health
sports_32.txt	sports	0.25	0.00	**0.75**	politics
sports_35.txt	sports	0.42	0.00	**0.58**	politics
sports_36.txt	sports	0.33	0.00	**0.67**	politics
sports_37.txt	sports	0.39	0.00	**0.61**	politics
sports_41.txt	sports	0.10	0.00	**0.90**	politics
sports_39.txt	sports	0.20	0.00	**0.80**	politics

Tabla 32: Clasificación *Politics* Word2vec

doc_name	class	p_health	p_sports	p_politics	predicted_class
politics_26.txt	politics	0.04	0.00	**0.96**	politics
politics_43.txt	politics	0.00	0.12	**0.88**	politics
politics_40.txt	politics	0.17	0.00	**0.83**	politics
politics_45.txt	politics	0.00	0.18	**0.82**	politics
politics_4.txt	politics	0.00	0.18	**0.82**	politics
politics_3.txt	politics	0.00	0.19	**0.81**	politics
politics_46.txt	politics	0.00	0.21	**0.79**	politics
politics_27.txt	politics	0.00	0.22	**0.78**	politics
politics_38.txt	politics	0.22	0.00	**0.78**	politics
politics_24.txt	politics	0.24	0.00	**0.76**	politics
politics_34.txt	politics	0.25	0.00	**0.75**	politics
politics_8.txt	politics	0.00	0.25	**0.75**	politics
politics_36.txt	politics	0.00	0.27	**0.73**	politics
politics_28.txt	politics	0.00	0.30	**0.70**	politics
politics_41.txt	politics	0.30	0.00	**0.70**	politics
politics_33.txt	politics	0.30	0.00	**0.70**	politics
politics_7.txt	politics	0.00	0.31	**0.69**	politics
politics_32.txt	politics	0.39	0.00	**0.61**	politics
politics_23.txt	politics	0.00	0.40	**0.60**	politics
politics_42.txt	politics	0.00	0.40	**0.60**	politics
politics_35.txt	politics	0.40	0.00	**0.60**	politics
politics_6.txt	politics	0.45	0.00	**0.55**	politics
politics_9.txt	politics	0.00	0.48	**0.52**	politics
politics_25.txt	politics	0.00	0.50	0.50	politics
politics_49.txt	politics	0.00	**0.51**	0.49	sports
politics_50.txt	politics	**0.52**	0.00	0.48	health
politics_44.txt	politics	**0.53**	0.00	0.47	health
politics_39.txt	politics	**0.54**	0.00	0.46	health
politics_30.txt	politics	0.00	**0.55**	0.45	sports
politics_29.txt	politics	**0.56**	0.00	0.44	health
politics_37.txt	politics	**0.60**	0.00	0.40	health
politics_48.txt	politics	**0.60**	0.00	0.40	health
politics_31.txt	politics	**0.65**	0.00	0.35	health
politics_47.txt	politics	**0.94**	0.06	0.00	health
politics_5.txt	politics	**0.83**	0.17	0.00	health

B.3. Bayes

Tabla 33: Clasificación *Health* Bayes

doc_name	class	p_health	p_sports	p_politics	predicted_class
health_14.txt	health	**1.00**	0.00	0.00	health
health_17.txt	health	**1.00**	0.00	0.00	health
health_24.txt	health	**1.00**	0.00	0.00	health
health_23.txt	health	**1.00**	0.00	0.00	health
health_28.txt	health	**1.00**	0.00	0.00	health
health_2.txt	health	**1.00**	0.00	0.00	health
health_45.txt	health	**1.00**	0.00	0.00	health
health_10.txt	health	**1.00**	0.00	0.00	health
health_16.txt	health	**1.00**	0.00	0.00	health
health_49.txt	health	**1.00**	0.00	0.00	health
health_1.txt	health	**1.00**	0.00	0.00	health
health_3.txt	health	**1.00**	0.00	0.00	health
health_7.txt	health	**1.00**	0.00	0.00	health
health_43.txt	health	**1.00**	0.00	0.00	health
health_13.txt	health	**1.00**	0.00	0.00	health
health_35.txt	health	**1.00**	0.00	0.00	health
health_6.txt	health	**1.00**	0.00	0.00	health
health_25.txt	health	**1.00**	0.00	0.00	health
health_9.txt	health	**1.00**	0.00	0.00	health
health_47.txt	health	**1.00**	0.00	0.00	health
health_29.txt	health	**1.00**	0.00	0.00	health
health_48.txt	health	**1.00**	0.00	0.00	health
health_31.txt	health	**1.00**	0.00	0.00	health
health_46.txt	health	**1.00**	0.00	0.00	health
health_34.txt	health	**1.00**	0.00	0.00	health
health_42.txt	health	**0.92**	0.08	0.00	health
health_27.txt	health	**0.92**	0.08	0.00	health
health_44.txt	health	**0.92**	0.08	0.00	health
health_26.txt	health	**0.92**	0.00	0.08	health
health_39.txt	health	0.08	**0.91**	0.01	sports
health_38.txt	health	0.08	**0.91**	0.01	sports
health_40.txt	health	0.08	0.01	**0.91**	politics
health_33.txt	health	0.00	**1.00**	0.00	sports
health_4.txt	health	0.00	**0.99**	0.01	sports
health_37.txt	health	0.00	**1.00**	0.00	sports

Tabla 34: Clasificación *Sports* Bayes

doc_name	class	p_health	p_sports	p_politics	predicted_class
sports_1.txt	sports	0.00	**1.00**	0.00	sports
sports_9.txt	sports	0.00	**1.00**	0.00	sports
sports_16.txt	sports	0.00	**1.00**	0.00	sports
sports_12.txt	sports	0.00	**1.00**	0.00	sports
sports_42.txt	sports	0.00	**1.00**	0.00	sports
sports_14.txt	sports	0.00	**1.00**	0.00	sports
sports_5.txt	sports	0.00	**1.00**	0.00	sports
sports_13.txt	sports	0.00	**1.00**	0.00	sports
sports_15.txt	sports	0.00	**1.00**	0.00	sports
sports_6.txt	sports	0.00	**1.00**	0.00	sports
sports_34.txt	sports	0.00	**1.00**	0.00	sports
sports_41.txt	sports	0.00	**1.00**	0.00	sports
sports_8.txt	sports	0.00	**1.00**	0.00	sports
sports_24.txt	sports	0.00	**1.00**	0.00	sports
sports_11.txt	sports	0.00	**1.00**	0.00	sports
sports_10.txt	sports	0.00	**1.00**	0.00	sports
sports_49.txt	sports	0.00	**1.00**	0.00	sports
sports_7.txt	sports	0.00	**1.00**	0.00	sports
sports_19.txt	sports	0.00	**1.00**	0.00	sports
sports_29.txt	sports	0.00	**1.00**	0.00	sports
sports_22.txt	sports	0.00	**1.00**	0.00	sports
sports_44.txt	sports	0.00	**1.00**	0.00	sports
sports_45.txt	sports	0.00	**1.00**	0.00	sports
sports_48.txt	sports	0.00	**1.00**	0.00	sports
sports_26.txt	sports	0.00	**1.00**	0.00	sports
sports_28.txt	sports	0.00	**1.00**	0.00	sports
sports_4.txt	sports	0.00	**1.00**	0.00	sports
sports_31.txt	sports	0.00	**1.00**	0.00	sports
sports_17.txt	sports	0.00	**1.00**	0.00	sports
sports_50.txt	sports	0.08	**0.92**	0.00	sports
sports_36.txt	sports	0.08	**0.92**	0.00	sports
sports_27.txt	sports	0.08	**0.91**	0.01	sports
sports_39.txt	sports	0.08	**0.91**	0.01	sports
sports_32.txt	sports	0.08	**0.91**	0.01	sports
sports_25.txt	sports	0.08	0.01	**0.91**	politics

Tabla 35: Clasificación *Politics* Bayes

doc_name	class	p_health	p_sports	p_politics	predicted_class
politics_28.txt	politics	0.00	0.00	**1.00**	politics
politics_15.txt	politics	0.00	0.00	**1.00**	politics
politics_18.txt	politics	0.00	0.00	**1.00**	politics
politics_32.txt	politics	0.00	0.00	**1.00**	politics
politics_30.txt	politics	0.00	0.00	**1.00**	politics
politics_4.txt	politics	0.00	0.00	**1.00**	politics
politics_43.txt	politics	0.00	0.00	**1.00**	politics
politics_33.txt	politics	0.00	0.00	**1.00**	politics
politics_19.txt	politics	0.00	0.00	**1.00**	politics
politics_14.txt	politics	0.00	0.00	**1.00**	politics
politics_27.txt	politics	0.00	0.00	**1.00**	politics
politics_40.txt	politics	0.00	0.00	**1.00**	politics
politics_46.txt	politics	0.00	0.00	**1.00**	politics
politics_34.txt	politics	0.00	0.00	**1.00**	politics
politics_26.txt	politics	0.00	0.00	**1.00**	politics
politics_8.txt	politics	0.00	0.00	**1.00**	politics
politics_9.txt	politics	0.00	0.00	**1.00**	politics
politics_17.txt	politics	0.00	0.00	**1.00**	politics
politics_47.txt	politics	0.00	0.00	**1.00**	politics
politics_44.txt	politics	0.00	0.00	**1.00**	politics
politics_38.txt	politics	0.00	0.00	**1.00**	politics
politics_1.txt	politics	0.00	0.00	**1.00**	politics
politics_10.txt	politics	0.00	0.00	**1.00**	politics
politics_25.txt	politics	0.00	0.00	**1.00**	politics
politics_50.txt	politics	0.00	0.00	**1.00**	politics
politics_7.txt	politics	0.00	0.01	**0.99**	politics
politics_11.txt	politics	0.08	0.00	**0.92**	politics
politics_37.txt	politics	0.00	0.50	0.50	unknown
politics_42.txt	politics	0.00	0.50	0.50	unknown
politics_49.txt	politics	0.04	0.48	0.48	unknown
politics_31.txt	politics	**0.92**	0.00	0.08	health
politics_3.txt	politics	**1.00**	0.00	0.00	health
politics_2.txt	politics	**1.00**	0.00	0.00	health
politics_6.txt	politics	**1.00**	0.00	0.00	health
politics_21.txt	politics	**1.00**	0.00	0.00	health

C. Manual de uso

A continuación, procederemos a detallar los pasos necesarios para poder ejecutar nuestro proyecto haciendo uso de la distribución Anaconda de Python.

1. Nosotros le recomendamos instalar la distribución Anaconda, al instalar también por defecto Jupyter Notebook. En este enlace puede encontrar una guía detallada de cómo instalarlo en Windows.
2. Una vez acabe la instalación, abra Anaconda Prompt.
3. Antes de abrir el proyecto, deberá instalar las siguientes librerías mediante el comando **pip install *librería***:
   - numpy
   - pandas
   - nltk
   - joblib
   - gensim
   - scikit-learn
4. Entre en la carpeta que le hemos entregado.
5. Ejecute ***jupyter notebook***.
6. Le aparecerá en la parte de abajo una url. Cópiela y péguela en el navegador. Se le mostrará este contenido en pantalla.

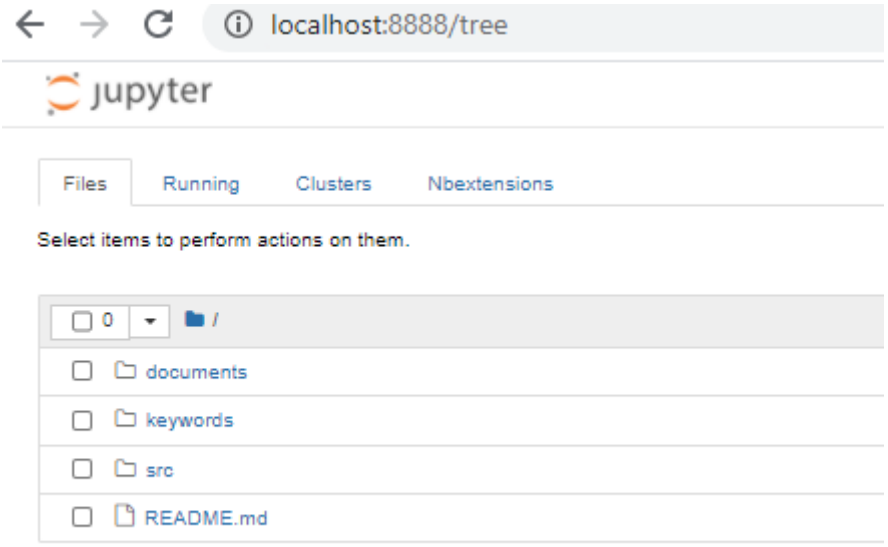

Figura 8: Estructura

Si no le aparecen estas carpetas es muy probable que no se encuentre en la carpeta correspondiente al zip entregado. Vuelva al Punto 4.

7. Entre en la carpeta **src**. Encontrará dos ficheros: uno el de la primera versión y otro el de versión extendida. Abra el que usted quiera.
8. Para ejecutar el notebook, en el menú de arriba, haga click en **Cell** y después en **Run all**.
9. A usted lo que realmente le interesa son los puntos **4.2** y **5**.
10. En el punto **4.2**, es donde se llama a la función que clasifica a los documentos. Por favor, descomente solo la que quiera probar.
11. En el punto **5**, encontrará la evaluación de los tres modelos probados.

Si tiene cualquier problema o duda a la hora de seguir este manual, póngase en contacto con nosotros mediante los siguientes correos electrónicos: *cesar.garcia.cabeza@alumnos.upm.es* o *a.gabarre@alumnos.upm.es*.