Clustering de noticias

Recuperación de información y minería de texto

Ángeles Blanco Fernández y Ana Rodríguez Redondo

Extracto

En esta práctica se ha abordado el problema de implementar un programa de procesado de noticias en la Red, mediante las librerías de procesamiento de lenguaje natural spaCy y NLTK, además de una posterior clasificación de las mismas según su temática utilizando un algoritmo de clustering jerárquico aglomerativo provisto por la librería scikit-learn.

Índice

1. Introducción

A modo de pequeña introducción teórica al área general de conocimiento de esta práctica, la minería de texto se define como el proceso de examinar colecciones de escritos con el objetivo de generar nueva información**.** Se utiliza para descubrir información relevante transformando el texto en datos que puedan ser utilizados para posterior análisis. Para conseguir extraer esta nueva información, entra en juego uno de los campos de la Inteligencia Artificial, el Procesamiento de Lenguaje Natural, o NLP (Natural Language Processing), por sus siglas en inglés. [1]

El NLP permite el tratamiento computacional del lenguaje humano. Esto requiere un proceso de modelización matemática para que un ordenador, que solo entiende bytes y dígitos, comprenda el lenguaje humano. El NLP es, por tanto, un conjunto de herramientas lingüísticas, modelos de machine learning y deep learning, así como arquitecturas software que permiten combinar en tiempo real los componentes anteriores. Así, el NLP podría definirse como un tipo de análisis lingüístico que ayuda a las máquinas a "leer" utilizando variedad de metodologías para descifrar las ambigüedades del lenguaje humano y convertirlo en idioma computacional. [1]

Algunas de las mencionadas herramientas lingüísticas del NLP son el análisis morfológico o léxico (que consiste en el análisis interno de las palabras que forman oraciones para extraer lemas, rasgos flexivos o unidades léxica compuestas), el análisis sintáctico (consistente en el análisis de la estructura gramatical de las oraciones), el análisis semántico (que proporciona la interpretación de las oraciones) o el análisis pragmático (que incorpora el análisis del contexto de uso a la interpretación final; incluyéndose aquí el tratamiento del lenguaje figurado). [2]

En la práctica aquí descrita se hará uso de las dos primeras herramientas (análisis morfosintáctico), siendo éstas las más básicas, pero suficientes para los objetivos propuestos que se relatan a continuación.

1.1 Objetivos

El objetivo principal de esta práctica es realizar un programa que efectúe un procesamiento adecuado del texto de 22 noticias internacionales (y en dos idiomas) que se encuentran en formato HTML, utilizando distintas aproximaciones para generar una clasificación de los textos a tratar por su temática, donde la medida del nivel de exactitud de dicha clasificación es caracterizada por el índice de Rand ajustado, que se explicará más adelante (sección 2.4.2).

2. Experimentación y resultados

En esta sección se irán detallando las diferentes aproximaciones utilizadas, tanto para la obtención de los datos, como su procesamiento y posterior clasificación mediante clustering, y se irán definiendo las bases teóricas de los conceptos utilizados según se vayan utilizando. En la § 2.1 se resumirán las principales características de las librerías en código Python utilizadas para el NLP de esta práctica. En la § 2.2 se relatarán las técnicas para la extracción del texto deseado de las páginas HTML. En la § 2.3 se expondrá un primer procesamiento de los textos, la traducción de algunos de ellos para conseguir una uniformidad lingüística; su guardado en formato TXT y los métodos utilizados para su lectura. Después, en la § 2.4 se mostrarán los diferentes procesos y

transformaciones efectuadas, así como una comparación de índices obtenidos. En la § 2.5, se proporcionará una explicación más pormenorizada de la parte encargada de clasificar los textos y medir su exactitud, y se explicará la preparación de los datos para su utilización en el algoritmo de clustering.

2.1 Librerías de NLP

El procesamiento de los textos se llevará a cabo mediante dos de las librerías en lenguaje Python más potentes y más utilizadas hoy en día en NLP, como son spaCy [3] y NLTK [4], con licencia de código abierto por el MIT (Massachusetts Institute of Technology) y por Apache 2.0, respectivamente.

Se ha usado en mayor medida la librería spaCy, principalmente por su sencillez de utilización y velocidad a la hora de procesar textos. El tratamiento de texto de ambas librerías es diferente. Mientras que NLTK procesa strings, spaCy se encuentra orientada a objetos. NLTK contiene una serie de funciones para distintas tareas de NLP que el usuario aplica a conveniencia, y cuyo input y output es en general un string o una lista de strings. Por otro lado, con spaCy, el texto, también en forma de string, ha de pasar a través de un pipeline predefinido (pero modificable) de procesamiento específico al lenguaje del texto, tras el cual se devuelve un objeto de tipo Doc. Cada objeto de este tipo almacena elementos con atributos, métodos y propiedades que dan acceso a la información lingüística deseada. [5] Se han utilizado ambas librerías porque sus funcionalidades se pueden complementar y enriquecen la propuesta de programa aquí expuesta.

Por otro lado, se ha utilizado la librería TextBlob [6] para realizar la detección del lenguaje de los textos y la posterior traducción de algunos de ellos. TextBlob ofrece estas funcionalidades mediante la API de Google Translate [7].

2.2 Obtención de datos

Como se ha mencionado anteriormente, las noticias a clasificar se encuentran en formato HTML, por lo que ha sido necesario extraer el texto principal. Para ello, se ha utilizado Beautiful Soup, en lenguaje Python, conocida librería que se usa para scrappear de manera sencilla el contenido de páginas web. Beautiful Soup crea un árbol de análisis con el contenido de cada página web parseada, pudiendo acceder directamente a dicho contenido a través de la etiqueta del bloque en el que se encuentra.

Para obtener una extracción más fiable del cuerpo de la noticia, y cerciorarse de que no se están cogiendo elementos de texto no deseados, se ha optado por seleccionar todas las etiquetas <p> de la página (especificando éstas un párrafo de texto), únicamente cuando no exista ninguna otra etiqueta o atributo en todo el árbol HTML que especifique que se trata del cuerpo del artículo (p. ej. itemprop = ‘articleBody’). De esta manera, se evitan futuros problemas de clasificación por parte del algoritmo a causa de la presencia de otros titulares presentes en la página, contenidos bajo etiquetas <p>. Puede verse en la parte inferior de la Figura 1, un fragmento del árbol de una noticia sobre un tiroteo en Texas, EEUU; en el cual se encuentra también el titular de una noticia sobre Sánchez, presidente actual de España, que podría dar lugar a un menor rendimiento en la clasificación, pues en esta práctica se procesan otras noticias principales sobre este sujeto.

Figura 1. Fragmento de árbol HTML de una noticia.

```python
def extractText(folder, listing):
    """
    Function for extracting text from HTML news articles in folder
    within the same parent directory.

    Parameters
    ----------
    folder: str
        Folder in which the documents are.
    listing: list
        List of documents.
    """
    if file.endswith(".html"):
        url = folder+"/"+file
    try:
        f = open(url,encoding="utf-8")
        raw = f.read()
    except UnicodeDecodeError:
        f = open(url,encoding="latin-1")
        raw = f.read()
    f.close()
    soup = BeautifulSoup(raw,'html.parser')
    text = ""

    for node in soup.find_all('p',itemprop='articleBody'):
        text = text + node.text
    if text == '':
        for node in soup.find_all('p',class_='article-body'):
            text = text + node.text
    if text == '':
        for node in soup.find_all('p'):
            text = text + node.text

    text = text.replace('\n','')
    return text
```

Figura 2. Función para extraer el texto de las noticias.

Se han eliminado además los caracteres representativos de saltos de línea del texto extraído, pues carecen de importancia en el análisis realizado. El código utilizado para esta parte consiste básicamente en una función que realiza las tareas anteriormente descritas, cuyo contenido se muestra en la Figura 2.

2.3 Lectura de datos

```python
def translateText(text, file, folder, format_file, from_lang, to_lang):
    """
    Function for translating and saving text to given format and folder
    within the same parent directory.

    Parameters
    ----------
    text: str
        Text for translating.
    file: str
        Name of the document.
    folder: str
        Desired folder in which to save the newly translated document.
    format_file: str
        Desired format.
    from_lang: str
        Original language.
    to_lang: str
        Desired language.
    """
    # Detecting and translating language with TextBlob:
    text = TextBlob(text)

    if text.detect_language() == from_lang:
        text = text.translate(from_lang=from_lang, to=to_lang)
    text = str(text)

    # Removing non-ascii characters.
    text = re.sub('[^\x00-\x7F]+', '', text)

    # Saving text:
    filename = (folder, file.split('.')[0], format_file)
    with open("./{folder}/{name}.{format_file}".format(folder=filename[0],\
              name=filename[1],format_file=filename[2]), "w") as file:
        file.write(text)
    file.close()

    return text
```

Figura 3. Función para detectar el idioma del texto, traducir al idioma deseado y guardar en formato TXT.

Tras la extracción, se ha procedido a la detección del lenguaje en el que se encuentran escritas las noticias. Para ello, como se adelantó en la § 2.1, se ha utilizado la librería TextBlob. Se ha detectado un total de 13 documentos en español y 9 documentos en inglés. Para conseguir uniformidad lingüística y obtener mejores resultados, se ha realizado la traducción al inglés de los 13 documentos de habla española, pues el desarrollo de las librerías de NLP es más avanzado para el idioma anglosajón. Posteriormente, tras efectuar la traducción, se ha realizado un guardado de cada noticia en formato TXT para una mayor libertad y rapidez en el manejo del texto, y se ha procedido a leer los

datos desde ese formato. Los nuevos documentos se han ido guardando en una lista de strings sobre la que poder iterar y realizar transformaciones. No obstante, todo el procesamiento se ha hecho durante las iteraciones de lectura de cada documento, como se puede comprobar en el código adjunto a esta memoria.

2.4 Procesamiento de los textos

Como se adelantó en la introducción, es necesario convertir el lenguaje a un tipo de estructura que un ordenador pueda interpretar, y para ello, se debe comenzar por las piezas fundamentales. Una pieza de texto es, esencialmente, una composición de caracteres, series de palabras y símbolos con un orden determinado. Cada palabra o símbolo representan un conjunto de características gramaticales que los dotan de significado y sentido en la composición. Entonces, de manera natural, antes de realizar cualquier procesamiento de texto, éste debe ser segmentado en unidades lingüísticas con significado: palabras, puntuación, números... A esta segmentación, desde el punto de vista del análisis léxico, se la conoce como tokenización. En cierto sentido, la tokenización es un método para simplificar contenido en unidades básicas antes de efectuar cualquier tipo de procesamiento. Se reemplaza cierto input de texto conocido con una serie de tokens que representan el significado de dicho input.

En inglés, al igual que en muchos otros idiomas, las palabras se encuentran generalmente separadas de una a otra por espacios en blanco, pero no todos los espacios en blanco son iguales. Por ejemplo, tanto “Los Angeles” como “rock 'n' roll” son piezas de significado individual, a pesar del hecho de que contengan varias palabras y espacios. Es importante, entonces, identificar y segmentar correctamente las unidades básicas de significado o tokens para poder realizar cualquier análisis. Webster y Kit [8] sugieren que encontrar tokens con significancia depende de la habilidad de reconocer patrones que muestren una colocación significativa. En lugar de simplemente confiar en que un string está separado por ciertos delimitadores a cada extremo, la segmentación en tokens significativos se basa en una especie reconocimiento de patrones. [9] Por todo esto, se pondrá especial énfasis en esta memoria en el funcionamiento interno de la tokenización por parte de spaCy.

En las subsecciones que siguen a continuación, se relatará el proceso de identificación, segmentación y selección de tokens representativos de los textos que serán utilizados en la § 2.5 para la clasificación temática mediante clustering de las 22 noticias. El procesamiento del texto se ha llevado a cabo mediante la creación de una clase, denominada TextClassifier, que contiene métodos para la eliminación de puntuación, caracteres no alfabéticos, stopwords (palabras que no aportan significado importante); selección de términos de interés morfológico, y ejecución de lematización (sustitución de palabras por su lema) y stemming (sustitución de palabras por una forma base). Toda esta nueva terminología se introducirá en detalle en cada subsección correspondiente.

2.4.1 Tokenización con spaCy [3]

La tokenización en spaCy se efectúa aplicando normas específicas para cada lenguaje. Cada objeto Doc consta de tokens individuales, y se puede iterar sobre ellos. Primero, el texto en bruto se divide

en caracteres mediante espacios en blanco, similar a text.split(' '). Luego, el tokenizador procesa el texto de izquierda a derecha. En cada subcadena, realiza dos verificaciones:

1. ¿Coincide la subcadena con una regla de excepción del tokenizador? Por ejemplo, "don’t" no contiene espacios en blanco, pero debe dividirse en dos tokens, "do" y "n't", mientras que "U.K." siempre debe ser un token.
2. ¿Se puede dividir un prefijo, un sufijo o un infijo? Por ejemplo, la puntuación como comas, puntos, guiones o comillas.

Si hay una coincidencia, se aplica la regla y el tokenizador continúa su ciclo, comenzando con las subcadenas recién divididas. De esta manera, spaCy puede dividir tokens anidados y complejos, como combinaciones de abreviaturas y múltiples signos de puntuación. Cabe destacar que si bien las reglas de puntuación suelen ser bastante generales, las excepciones del tokenizador dependen en gran medida de las características específicas del idioma individual; spaCy posee modelos concretos de entrenamiento de algoritmos para cada idioma.

Después de la tokenización, spaCy puede analizar y etiquetar un Doc determinado (es importante recalcar que la información del texto original siempre está disponible; de esta manera, no se pierde información al procesar el texto). Aquí es donde entra el modelo estadístico, que permite a spaCy hacer una predicción de qué etiqueta o etiqueta es más probable que se aplique en este contexto. Un modelo consta de datos binarios y se produce al mostrarle a un sistema suficientes ejemplos para que realice predicciones que se generalizan en todo el idioma; por ejemplo, una palabra que sigue a “the” en inglés es probablemente un sustantivo, y, como tal, recibirá una etiqueta POS (Part-Of-Speech) de ‘NOUN’. Todas las anotaciones lingüísticas se encuentran disponibles, como se mencionó en la § 2.1, como atributos de cada token.

Tras esta pequeña introducción sobre la tokenización en spaCy, se detallan ahora los procesos realizados a los textos.

2.4.2 Eliminación de símbolos y puntuación

Teniendo en cuenta todo lo relatado en la introducción de esta sección, se podría establecer entonces que un token es:

1. Lingüísticamente significativo.
2. Metodológicamente útil.

Partiendo de las dos premisas anteriores, se pueden descartar directamente todos los signos de puntuación, pues, aunque posean un significado lingüístico, para realizar los objetivos de esta práctica, y quedarse con los tokens representativos de un texto, manifiestamente, un signo de puntuación no aportará ninguna característica esencial. En la misma línea, para la tarea que concierne, tampoco son de importancia números demás simbología no alfabética, y por ello también se eliminan. No obstante, aunque no carezca de relevancia para el objetivo en cuestión y dada la temática de las noticias, como se puede ver en la función de la Figura 4, se dan dos ejemplos de lo que se puede hacer con determinados símbolos: traducirlos a la palabra que representan.

Para la eliminación de los signos de puntuación se ha hecho uso de la lista de signos que provee la librería string ('!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~') en conjunto con las etiquetas de POS (Part-Of-Speech) establecidas por el modelo de spaCy sobre los tokens del objeto Doc que se introdujo en apartados anteriores. Además, se ha utilizado una expresión regular para quedarse únicamente con caracteres propios del alfabeto romano; de esta forma, cualquier símbolo, signo de puntuación o caracter no alfabético es eliminado por completo. Sin embargo, antes de esto, se ha dividido el texto mediante puntos (.) y comillas (”), para separar algunas sentencias que se econtraban mal extraídas (p. ej. ‘that pocket.In announcing’). En este caso, no supone ningún tipo de peligro el realizar esta primera acción porque las siglas o acrónimos que pueda haber en los textos (p. ej. U.K.), no poseen gran relevancia sobre el objetivo de la clasificación, pero en otros contextos, es muy probable que este tipo de modificaciones se debiese manejar con más cuidado.

```
def removeSymbols(self):
    """ Removing punctuation and symbols."""

    # Initial variable declaration:
    text = self.text
    words = []
    punctuation = list(string.punctuation)
    pos_not_wanted = ['PUNCT','SPACE']

    # Translating some symbols.
    text = re.sub('\$', " dollar ", text)
    text = re.sub('\%', " percent ", text)

    # Dividing words with a dot or quotation marks
    # in between them:
    text = text.split('.')
    text = ' '.join(text)
    text = text.split('”')
    text = ' '.join(text)

    # Transforming to spacy.doc.Doc object for easier processing.
    doc = nlp(text)

    # Removing punctuation
    # (double way with pos_tags and list of punctuation),
    # and the rest of symbols and also numbers.

    for word in [token for token in doc
                 if token.pos_ not in pos_not_wanted
                 and re.search('[a-zA-Z]', token.text)]:
        if word.text not in punctuation:
            words.append(word.text)

    text = ' '.join(words)
    return text
```

Figura 4. Función para eliminar signos de puntuación y caracteres no alfabéticos.

Llegado este punto, es hora de enseñar una primera ejecución del código y ver cuán de acertadas son las asunciones tomadas. La conversión a un lenguaje matemático de los tokens que pasarán al algoritmo para su utilización en la clasificación se hace únicamente con el valor de la frecuencia de estos tokens en cada documento. Ésta será la estructura interpretable por el algoritmo de clustering, mencionada al comienzo de esta sección. En la siguiente sección (2.5) se entrará en detalle sobre cómo se efectúa esta conversión matemática, cuál es el algoritmo de clustering utilizado, y como se miden las similitudes entre las estructuras a la hora de generarse las temáticas.

El índice de Rand ajustado, sin efectuar ningún tipo de procesamiento y una vez cambiado el patrón de referencia Gold Standard (pues el patrón de referencia dado en la primera versión de la práctica era erróneo y una vez realizadas diversas ejecuciones, tras efectuar un procesamiento con sentido y una investigación primaria de los textos a analizar, fue sencillo descubrir el verdadero patrón de clasificación correcta), genera un valor de -0.09[1]. Pero, ¿qué significa este índice y qué significa el valor obtenido? Este índice es, sencillamente, una medida de la similitud entre dos resultados de clustering. Y la especificación de ‘ajustado’ se debe a la versión corregida para el azar del índice de Rand. Dicha corrección para el azar establece una base mediante el uso de una similitud esperada (referencia Gold Standard) de todas las comparaciones por pares entre todas las comparaciones posibles especificadas por un modelo aleatorio. Pese a que el índice de Rand solo genera valores entre 0 y 1, el índice de Rand ajustado puede devolver valores negativos si el índice

[1] La ejecución en esta fase es muy lenta. Es necesaria la adición de caches y mejora de rendimiento del código para ejecutarse en tiempos óptimos. No obstante, como cabe esperar, en posteriores fases donde el número de tokens únicos es reducido, el tiempo de ejecución sí es adecuado.

comparado es menor que el valor esperado. La formulación matemática de esta medida se puede consultar en [10].

Tras efectuar este primer procesado eliminando puntuación y símbolos, la lista de tokens únicos se reduce en 1000 términos y el índice de Rand, en adelante, ARI (Adjusted Rand Index), aumenta hasta llegar a un valor de 0.76[2]. Esto significa que las asunciones, por el momento, son correctas.

2.4.3 Eliminación de stopwords y selección de términos

El siguiente paso en la búsqueda de los mejores tokens representativos de cada texto, pasa por una eliminación de stopwords (véase la siguiente Figura), denominación de uso general a palabras que no aportan ningún significado importante sobre el texto. Para sustraer estas palabras se ha utilizado la lista de stopwords en lengua inglesa proporcionada por la librería NLTK, en conjunto con las etiquetas internas de los tokens de spaCy generadas tras su paso a través del pipeline (token.is_stop). Por otro lado, se eliminan aquí también morfemas como determinantes y preposiciones, palabras como adverbios o subclases de adverbios que existen en inglés como los que comienzan por wh- (con POS ADV), que no aportan tampoco un significado relevante sobre el texto. Además, se utiliza la distribución de las longitudes de los tokens encontrados para eliminar también outliers, o longitudes poco comunes. De esta forma, por ejemplo, el percentil 95 de la distribución de los tokens únicos es de 13 letras, y se estarían eliminando partes del texto extraído como 'Columbus388TodayMorndayJuevesFriday' en la segunda noticia, con una longitud de 35 caracteres, o 'AdvertisementSupported', en la tercera noticia, con una longitud de 22 caracteres. Es simplemente otra forma de perfeccionar el texto extraído de las páginas HTML y reducir la cantidad de términos sin importancia o que se encuentran fuera de la materia en cuestión. Por otro lado, con el percentil 5, se eliminan también pronombres como 'I', 'you' o 'me' que carecen de un significado de relevancia fuera de la oración en la que se encuentran, pues sus referentes son contextuales. En otro tipo de ejercicios es probable que la eliminación de palabras mediante percentiles no fuese recomendable, sobre todo en problemas donde tuviese importancia alguna palabra muy corta o donde no hubiese composiciones de palabras sin significado lógico o lingüístico como las vistas en líneas anteriores debidas a la forma de extracción del texto.

```
def removeStopWords(self, nouns = True):
    """ Removing stopwords and more non-useful words.
    Parameters
    ----------
    nouns: bool
        If True, removes all words but noun words.
    """


    text = self.text
    stopwords_en = nltk.corpus.stopwords.words("english")
    pos_not_wanted = ['DET','SPACE','ADV','ADP','PART']
    words = []

    # Transforming to spacy.doc.Doc object for easier processing.
    doc = nlp(text)

    # Double removing stop_words with Spacy and NLTK.
    tokens = [token for token in doc
              if token.pos_ not in pos_not_wanted
              and not token.is_stop]
    if nouns == True:
        tokens = [token for token in tokens if token.pos_ == 'NOUN']

    lengths = [len(token) for token in tokens]

    for word in tokens:
        if word.text not in stopwords_en:
            # Removing outliers in word-length:
            if (len(word.text) >= numpy.percentile(lengths,5)) &\
               (len(word.text) <= numpy.percentile(lengths,95)):
                   words.append(word.text)

    text = ' '.join(words)
    return text
```

Figura 5. Función para eliminar palabras no deseadas.

Ejecutando el código con estas nuevas funcionalidades añadidas, los términos únicos se reducen en 600 unidades y el ARI alcanza un valor de 0.85[3]. No obstante, cabe pararse a reflexionar. El número de términos únicos sigue siendo muy alto (aproximadamente 3000), pero, ¿realmente son

[2] Véase [1].
[3] Véase [1].

necesarios tantos términos para determinar la temática intrínseca de un texto? ¿Cuáles son las unidades lingüísticas que dan nombre e identifican a todo lo conocido? Si el lector estaba pensando en los sustantivos, se encuentra en lo cierto. En la función de la Figura 5, se muestra que ésta contiene un argumento con la opción para quedarse únicamente con los sustantivos del texto. Ejecutando el código bajo esta condición, el número de términos únicos se reduce en más de un 50%, y el ARI aumenta hasta alcanzar el valor máximo de 1.0.

```
Prepared  22  documents.
They can be accessed using or_texts[n], being n an integer from 0 to 21.
Distribution of documents by language after translation:  {'en': 22}
Unique terms found:  1445
Vectors created.
Test:  [2 5 1 1 1 4 1 1 1 0 2 2 4 4 0 1 4 2 0 0 0 3]
Reference:  [0, 5, 2, 2, 2, 3, 2, 2, 2, 4, 0, 0, 3, 3, 4, 2, 3, 0, 4, 4, 4, 1]
Adjusted Rand Index:  1.0
```

Figura 6. Valores de salida tras ejecutar el programa con sustantivos como términos únicos.

Esto significa que las asunciones realizadas hasta el momento han tenido éxito. La clasificación de los textos es en este instante es totalmente correspondida por la clasificación referente. No obstante, aunque la clasificación sea técnicamente perfecta en este punto según la medida utilizada, en las siguientes subsecciones se mostrarán más formas de procesamiento que se pueden realizar, y que aumentan además el rendimiento del programa reduciendo la cantidad de términos.

2.4.4 Lematización y stemming

Por razones gramaticales, los documentos van a utilizar diferentes formas de una palabra, como organizar, organizas y organizando. Además, hay familias de palabras relacionadas de forma derivada con significados similares, como democracia, democrático y democratización. En muchas situaciones, parece que sería útil para una búsqueda de una de estas palabras devolver documentos que contengan otra palabra en el conjunto. El objetivo tanto de la lematización como del stemming es reducir las formas de inflexión y, a veces, las formas relacionadas de una palabra a una base común. [11]

Por lo general, el stemming se refiere a un proceso heurístico que corta los extremos de las palabras y, a menudo, incluye la eliminación de los afijos (prefijos, sufijos e infijos) derivados. La lematización generalmente se refiere al uso de un vocabulario y un análisis morfológico de las palabras, normalmente con el objetivo de eliminar solo los morfemas flexivos y devolver la forma de base o de diccionario de una palabra, que se conoce como el lema. Por ejemplo, el lema de la palabra ‘trophies’ es ‘trophy’ (véase la Figura 7), tal cual se buscaría en un diccionario, sin morfema flexivo de plural, pero su stem es ‘trophi’, que en este caso coincide con su raíz. Por lo general, un stem es la composición de la raíz de la palabra (unidad mínima de significado léxico) y morfemas derivativos. Otro ejemplo, la palabra ‘walking’ es la forma derivada de ‘walk’, que también es su lema y además es su stem. [12]

```
nlp('walking')[0].lemma_
'walk'

stemming.stem('walking')
'walk'

nlp('trophies')[0].lemma_
'trophy'

stemming.stem('trophies')
'trophi'
```

Figura 7. Ejemplos de lematización y stemming realizados con spaCy y NLTK.

Para probar esta fase del procesamiento, se volverá a coger todos los términos devueltos tras pasar por la función vista anteriormente de removeStopWords() con la condición del argumento noun = False, con el objetivo de observar satisfactoriamente el efecto de este proceso partiendo desde un índice de 0.85.

Efectuando primero una lematización, el número de términos únicos continúa en torno a los 3000, y el índice sube a un 0.89, lo cual tiene sentido, pues muchas palabras habrán sido reducidas a una forma común. Efectuando ahora una prueba ejecutando únicamente la fase de stemming (tal y como se ha realizado el código, tanto para realizar lematización como para stemming, se trata de una única función que realizará una opción u otra, o ambas, según se indique en sus argumentos, como se puede ver en la Figura 8), el número de términos es aproximadamente igual al obtenido tras lematizar, aunque ligeramente menor, pero el índice de Rand llega otra vez hasta un valor máximo de 1.0. Esto también tiene sentido, pues las formas base del stemming, abarcan muchas más palabras que un lema.

```
def rootText(self, lem = True, stem = False):
    """ Perform lemmatization and stemming.

    Parameters
    ----------
    lem: bool
        True by default. Returns token lemma.
    stem: bool
        False by default. Returns token stem.

    """
    text = self.text
    words = []

    # Transforming to spacy.doc.Doc object for easier processing.
    doc = nlp(text)

    # Performing lematization and/or stemming to non-proper nouns:
    if lem == True:
        words = [str(token.lemma_) for token in doc if not token.pos_ == 'PROPN']
        words.extend([token.text for token in doc if token.pos_ == 'PROPN'])
        if stem == True:
            raise ValueError("Arguments cannot be True nor False simultaneously.")
        elif stem == False:
            return ' '.join(words)
    else:
        if stem == True:
            stemming = nltk.SnowballStemmer('english')
            words = [stemming.stem(word.text) for word in
                     [token for token in doc if not token.pos_ == 'PROPN']]
            words.extend([token.text for token in doc if token.pos_ == 'PROPN'])
            return ' '.join(words)
        else:
            raise ValueError("Arguments cannot be True nor False simultaneously.")
```

Figura 8. Función que lematiza y efectúa stemming.

2.4.5 Obtención de entidades nombradas

El Reconocimiento de Entidades Nombradas (NER) es probablemente el primer paso hacia la extracción de información que busca ubicar y clasificar las entidades nombradas en el texto en categorías predefinidas, como los nombres de personas, organizaciones, ubicaciones, expresiones de tiempos, cantidades, valores monetarios, porcentajes, etc. La NER se usa en muchos campos del procesamiento del lenguaje natural, y puede ayudar a responder muchas preguntas del mundo real, como: ¿Qué empresas fueron mencionadas en el reportaje? ¿Se mencionaron los productos especificados en las quejas o comentarios? ¿El tweet contiene el nombre de una persona? ¿El tweet contiene la ubicación de esta persona? [13] En la práctica objeto de esta memoria, el reconocimiento de entidades nombradas es un paso muy importante, pues otorgan todavía más información que los sustantivos a la hora de clasificar las noticias. Las noticias en cuestión están construidas y desarrolladas mayormente alrededor de entidades nombradas, y su temática puede derivarse muy fácilmente a partir de ellas.

Entonces, ¿qué es una entidad nombrada? Una entidad nombrada, según la propia documentación de spaCy, es un "objeto del mundo real" al que se le asigna un nombre, por ejemplo, una persona, un país, un producto o un título de libro. spaCy puede reconocer varios tipos de entidades nombradas en un documento, solicitando al modelo una predicción. Debido a que los modelos son estadísticos y dependen en gran medida de los ejemplos en los que fueron entrenados, esto no

siempre funciona a la perfección y puede necesitar algunos ajustes posteriores, dependiendo del caso de uso. Las entidades nombradas están disponibles como la propiedad ents de un objeto Doc.

```
'Richard Pérez Peña',
'Tiger Woods',
'Sura',
'Read Next Fairfax',
'Mecca Medina',
'Ted Cruz',
'Junts Catalunya',
'Trickle',
'Veronika Velez',
'Lindsey Kildow',
'Mariace Close',
'Mohamed Emwazi Jihadi',
'Rupert Murdoch',
'Alcott Vonn',
```

La funcionalidad de reconocimiento de entidades nombradas de spaCy ha sido entrenada con el corpus de OntoNotes 5 [14], y ofrece una gran cantidad de tipos de entidades. No obstante, para esta práctica, las entidades realmente interesantes son aquellas catalogadas bajo la etiqueta de 'PERSON'. Pueden verse algunos ejemplos de entidades de personas extraídas de los documentos en la figura de la derecha. La extracción de entidades en el código propuesto se ha relizado mediante la sencilla función que sigue, donde filtering es el argumento sobre el cual se especifica el tipo de entidad nombrada deseada:

```
def getNamedEntities(self, filtering):
    """ NER """
    text = self.text
    doc = nlp(text)
    return [ent.text for ent in doc.ents if ent.label_ in filtering]
```

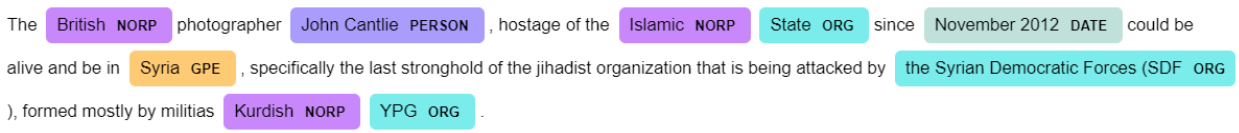
Figura 9. Función para extraer entidades nombradas de los documentos.

Tras ejecutar el código quedándose únicamente con la lista de entidades nombradas y matcheando frente a los textos procesados tras pasar por la función de removeStopWords, los términos únicos, es decir, las entidades, son un total de 176, y el Índice de Rand ajustado vuelve a obtener un valor máximo de 1.0. Pero no solo esto es lo mejor de utilizar las entidades nombradas, sino que el rendimiento del programa y su velocidad de ejecución aumentan enormemente, pues la cantidad de términos a procesar es un orden de magnitud menor. Se puede concluir que la utilización de entidades nombradas es la elección más eficaz y eficiente en esta práctica.

```
Prepared  22  documents.
They can be accessed using or_texts[n], being n an integer from 0 to 21.
Distribution of documents by language after translation:  {'en': 22}
Unique terms found:  3538
Named entities found:  176
Vectors created.
Test:  [4 5 0 0 0 2 0 0 0 1 4 4 2 2 1 0 2 4 1 1 1 3]
Reference:  [0, 5, 2, 2, 2, 3, 2, 2, 2, 4, 0, 0, 3, 3, 4, 2, 3, 0, 4, 4,
4, 1]
Adjusted Rand Index:  1.0
```

Figura 10. Resultados tras la utilización de entidades nombradas en el programa.

A modo de ejemplo, se muestra también una pequeña parte del poder de visualización de spaCy y el sentido de utilizar entidades nombradas en la clasificación, con la funcionalidad de displaCy, ejecutando el script del programa en un notebook de Jupyter (que se entrega adjunto a esta memoria y al código), para que el lector vea la variedad de entidades nombradas que spaCy es capaz de detectar:



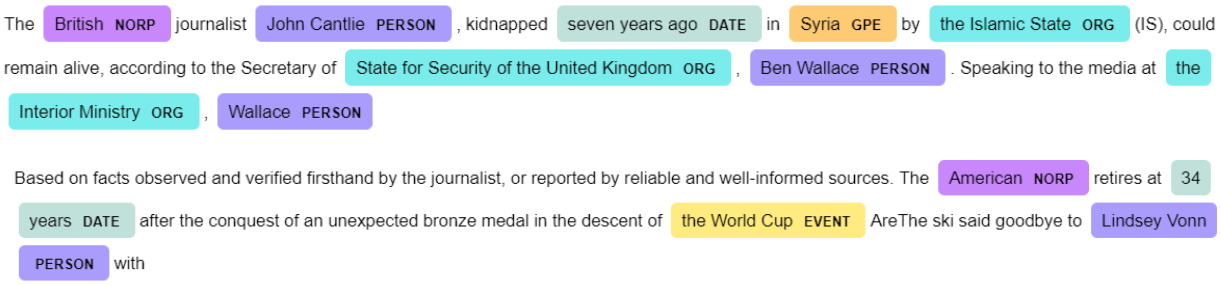


Figura 11. Detección de entidades nombradas por spaCy con sus categorías para tres párrafos de los documentos analizados; los dos primeros provenientes de las noticias sobre la posible vida del periodista John Cantlie, y el último sobre la retirada de la esquiadora Lindsey Vonn.

En los documentos de la figura anterior, donde se comparan las entidades nombradas obtenidas, se ve clara la similitud en los dos primeros en la repetición de dichas entidades por poseer la misma temática, y la clave de su utilización, diferenciándose con respecto al tercer ejemplo, de temática totalmente diferente, y entidades nombradas también completamente distintas.

A modo de añadidura, antes de concluir y pasar a la siguiente sección, cabe destacar que podrían utilizarse otras maneras de quedarse con pocos términos únicos pero efectivos, como por ejemplo, utilizando las funcionalidades extra que ofrece la librería textaCy, pero que no sea han probado para esta práctica. Según la propia documentación, textaCy es una librería de Python para realizar tareas de procesamiento de lenguaje natural (NLP) de alto nivel, desarrollada sobre spaCy. Con los conceptos básicos: tokenización, etiquetado de POS, análisis... descargados a otra librería, textaCy se enfoca en tareas facilitadas por la disponibilidad de texto tokenizado, POS etiquetado y analizado: extracción de palabras clave, estadísticas y mucho más. Teniendo esto en mente, podría utilizarse esta funcionalidad de identificar palabras clave en un documento, y seguramente obtener también buenos resultados.

2.5 Clustering de los textos

En esta sección, como se ha ido adelantando a lo largo del documento, se detallará la parte del código encargada de realizar específicamente la clasificación de los textos. Esto se ha realizado mediante la creación de una clase denominada TextClassifier, con métodos para la realización de TF-IDF (Term Frequency-Inverse Document Frequency), vectorizar, y aplicar el algoritmo de clustering utilizando el cálculo de una distancia concreta. No obstante, la mayoría de estos términos no se han introducido todavía, así que antes de continuar, es necesario definirlos.

Se mencionó en secciones anteriores que es necesario traducir la lista de términos a un lenguaje matemático interpretable por un algoritmo, y que todas las pruebas anteriores se han efectuado transformando los términos únicos en una lista de valores de frecuencias de aparición de las palabras en los documentos analizados. Estas frecuencias de aparición es lo que se conoce como Term Frequency o TF.

TF-IDF es un estadístico que pretende reflejar la importancia de una palabra para un documento en una colección o corpus. A menudo se utiliza como factor de ponderación en búsquedas de recuperación de información, extracción de texto y modelado de usuarios. El valor de TF-IDF

aumenta proporcionalmente al número de veces que aparece una palabra en el documento y se compensa con la cantidad de documentos en el cuerpo que contiene la palabra, lo que ayuda a ajustar el hecho de que algunas palabras aparecen con más frecuencia en general. Además, TF-IDF es uno de los esquemas de ponderación de términos más populares en la actualidad. La parte de Inverse Document Frequency, o IDF, se incorpora a TF en forma de un factor de frecuencia de documento inverso que disminuye el peso de los términos que aparecen con mucha frecuencia en el conjunto de documentos y aumenta el peso de los términos que ocurren raramente. Esto es así para restar importancia a palabras que aparecen mucho en un documento, como determinantes o preposiciones, y que no son palabras clave adecuadas para distinguir la relevancia de un documento o que no son representativas del mismo, como se ha visto en secciones anteriores.

```
def TFIDF(self, doc, idf = False, tf = True):
    """
    Term Frequency-Inverse Document Frequency, numerical statistic
    intended to reflect how important a word is to a document
    in a collection.

    Parameters
    ----------
    doc: spacy.doc.Doc
    tf: bool
        Default value is True.
    idf: bool
        Default value is False.

    Reference
    ---------
    For the calculus of the tf and idf terms, the following code,
    from the NLTK library, has been adapted:

    Natural Language Toolkit: Texts
    Copyright (C) 2001-2018 NLTK Project
    """
    terms = self.terms
    docs = self.docs
    word_tf = []

    for term in terms:
        if (idf == False) and (tf == True):
            # The frequency of the term in text.
            tf_ = doc.text.count(term)/len(doc)
            word_tf.append(tf_)
        elif (idf == True) and (tf == False):
            # The inverse document frequency.
            matches = len([True for doc in docs if term in doc.text])
            idf_ = (numpy.log10(len(docs) / matches) if matches else 0.0)
            word_tf.append(idf_)
        elif (idf == True) and (tf == True):
            # TF & IDF.
            tf_ = doc.text.count(term)/len(doc)
            matches = len([True for doc in docs if term in doc.text])
            idf_ = (numpy.log10(len(docs) / matches) if matches else 0.0)

            # Dot product of tf and idf.
            word_tf.append(tf_*idf_)
        else:
            raise ValueError('At least one argument must be True.')
    return word_tf
```

Figura 12. Función para realizar TF-IDF.

La aplicación de este estadístico en el código, se ha llevado a cabo a partir de una modificación de los métodos internos de la clase de TextCollection de la librería de NLTK. La función creada a partir de los mismos, permite escoger mediante argumentos de la misma, si se desea realizar únicamente TF, IDF o ambos sobre los términos y el documento de elección. Los términos se pasan al método a través del constructor de la clase.

El efecto del estadístico de TF ya se ha observado a lo largo de todo el documento, pero lo que no se ha mostrado es el efecto de IDF por separado, o de ambos combinados, que no es más que el producto de las dos cantidades estadísticas. Para observar el efecto, se han realizado pruebas con alguna de las fases relatadas anteriormente que no resultaban en un ARI con valor de 1.0, como por ejemplo, los términos y el procesado obtenidos tras la lematización de los tokens (recuérdese, el ARI era de un 0.89), y también con la fase en la que se utilizaron únicamente entidades nombradas, con un ARI de 1.0, pero el resultado obtenido no supera el 0.1, por lo que se deduce que utilizar IDF no es una buena opción en esta práctica. Esto podría ser debido a que la penalización sobre palabras frecuentes en el texto no es estrictamente necesaria y es posible que se estén penalizando palabras clave que aparezcan con frecuencia (como entidades nombradas, por ejemplo); pero esto es solo una conjetura.

Se concluye, por tanto, que la ejecución de TF es la más adecuada en el ejercicio aquí descrito. No obstante, ¿qué se obtiene tras la aplicación de este estadístico? Tras la utilización de la función de la Figura 12, se obtiene un vector con las frecuencias de cada término por cada documento. Posteriormente, en otro método denominado vectorText(), se obtiene la matriz de dos dimensiones

con todos las frecuencias de aparición de los términos únicos en todos los textos; es decir, se concatenan todos los vectores anteriormente obtenidos para cada documento, para formar la matriz de la siguiente figura:

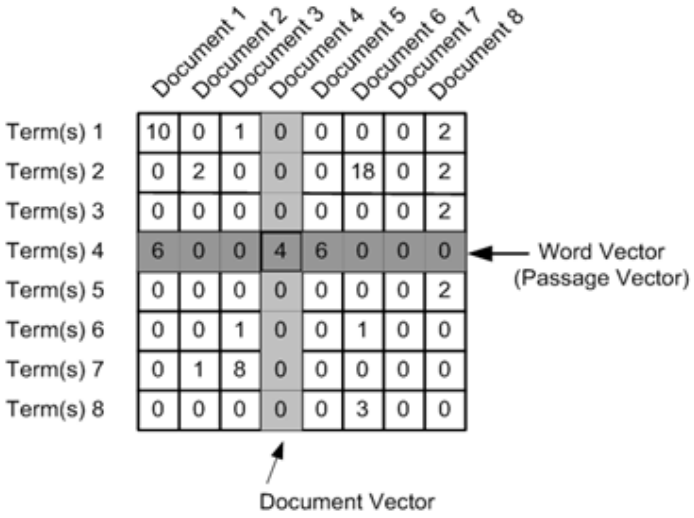


Figura 13. Ejemplo de matriz TF generada por el método de vectorText(). [15]

Finalmente, esta matriz de dos dimensiones se lee en el método clusterTexts() de la Figura 14, donde se calcula la similitud entre vectores de distintos documentos midiendo la distancia entre los mismos, y se agrupan en clústers siguiendo un criterio de minimización de esta distancia, es decir, los vectores de documentos que posean la menor distancia se agruparán conjuntamente, y que posean distancias mayores se desecharán hacia otros clústers. El algoritmo que realiza este ejercicio de búsqueda de la mínima distancia y agrupamiento se trata de un algoritmo de clustering jerárquico aglomerativo provisto por la librería de scikit-learn.

```
def clusterTexts(self, clustersNumber, distance):
    """
    Method for classifying the array of vectors generated in the constructor
    with an Agglomerative Clustering algorithm.

    Parameters
    ----------
    clustersNumber: int
        Number of clusters for the algorithm to make.
    distance: str
        Distance type for calculations. Values within 'cosine' and 'euclidean'.
    """
    vectors = self.vectorText()
    #print(vectors)

    # Initializing the clusterer:
    clusterer = AgglomerativeClustering(n_clusters=clustersNumber,
                                        linkage="average", affinity=distanceFunction)
    clusters = clusterer.fit_predict(vectors)

    return clusters
```

Figura 14. Función para realizar el clustering de los textos.

El cálculo de la distancia durante todas las pruebas mostradas en esta memoria se ha realizado con la distancia coseno. Pero antes de explicar qué significa, recuérdese primero la conocida distancia euclídea, cuya formulación es

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \quad ,$$

donde x e y son dos vectores. La formulación de la distancia coseno, es la siguiente:

$$\frac{x \bullet y}{\sqrt{x \bullet x}\sqrt{y \bullet y}}$$

donde x e y son también dos vectores. El resultado obtenido cambiando la distancia a la distancia euclídea, y utilizando las entidades como términos únicos, es de un ARI de -0.01, como se puede ver en la Figura 15.

```
Prepared  22  documents.
They can be accessed using or_texts[n], being n an integer from 0 to 21.
Distribution of documents by language after translation:  {'en': 22}
Unique terms found:  3538
Named entities found:  176
Vectors created.
Test:  [1 3 1 0 0 1 5 4 1 1 1 1 1 1 2 1 1 1 1 1 1]
Reference:  [0, 5, 2, 2, 2, 3, 2, 2, 2, 4, 0, 0, 3, 3, 4, 2, 3, 0, 4, 4,
4, 1]
Adjusted Rand Index:  -0.00880705
```

Figura 15. Resultados tras prueba de términos con entidades y distancia euclídea.

¿Por qué este cambio tan abismal con respecto al cálculo con la distancia coseno? Considérese la siguiente imagen:

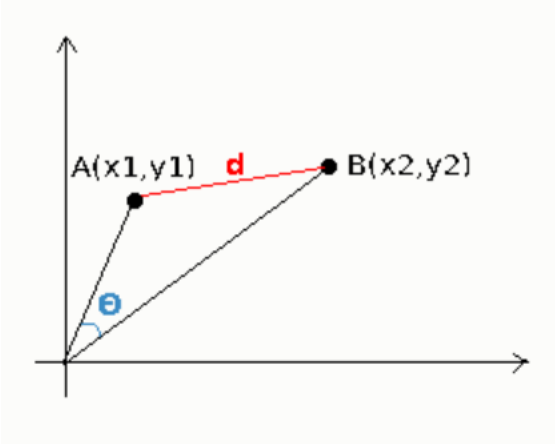


Figura 16. Representación visual de la distancia euclídea y la similitud de coseno.

Arriba se muestra una representación visual de la distancia euclídea (d) y la similitud de coseno (θ). Mientras que el coseno identifica el ángulo entre los vectores (por lo tanto, no teniendo en cuenta su peso o longitud), la distancia euclídea es similar a usar una regla para medir realmente la distancia lineal. Dos puntos A y B como los de la imagen pueden estar muy separados en el plano euclídeo y por tanto obtener una distancia euclídea muy alta y como consecuencia repartirse en clústers diferentes, pero obtener una distancia coseno muy baja porque son casi paralelos entre sí, lo que indicaría su similitud. Entonces, utilizando la distancia euclídea estos vectores de documentos estarían mal clasificados, al contrario que al utilizar la similitud de coseno.

Por otro lado, la similitud de coseno se usa generalmente como una métrica para medir la distancia cuando la longitud de los vectores no importa. Esto sucede, por ejemplo, cuando se trabaja con datos de texto representados por conteos de palabras, en lugar de una frecuencia (conteo corregido con la longitud del documento, como en TF). Se podría suponer que cuando aparece una palabra

(por ejemplo, 'ciencia') más frecuente en el documento 1 que en el documento 2, ese documento 1 está más relacionado con el tema de la ciencia. Sin embargo, también podría ser el caso de que se esté trabajando con documentos de longitud desigual (artículos de Wikipedia, por ejemplo). Entonces, la ciencia probablemente ocurrió más en el documento 1 solo porque era mucho más largo que el documento 2. La similitud de coseno corrige esto. [16]

Por otro lado, otros algoritmos para la determinación de la temática podrían ser también utilizados, como por ejemplo el LDA, o Latent Dirichlet Allocation, que es un modelo probabilístico que asume que los documentos son una mezcla de temas y que cada palabra en el documento es atribuible a los temas del documento. [15]

3. Conclusiones

Se ha creado un programa mediante clases y funciones para realizar el procesado y clasificado por su temática de 22 noticias internacionales mediante un algoritmo de clustering aglomerativo, el uso de un estadístico TF-IDF y diversas fases de procesamiento de los textos, que incluyen la tradicional y lógica eliminación de signos de puntuación y stopwords, y demás nuevas propuestas. Se ha observado cómo la elección de términos clave representativos del texto es esencial para un buen ejercicio de clustering, y cómo, antes de todo ello, es fundamental una correcta determinación estos términos mediante tokenización. Se ha visto, además, que la utilización de únicamente los sustantivos del texto o de las entidades nombradas de personas ha proporcionado un índice de Rand ajustado con valor máximo de 1, y se ha comprobado también cómo la reducción de términos mediante lematización y stemming (sobre todo esta última) ha mejorado también los resultados. Por otro lado, se ha visto la importancia de la utilización de la métrica de coseno en el contexto de la minería de texto.

Referencias

[1] Universidad de Alcalá (UAH), "Text Mining vs. Natural Language Processing" (Octubre, 2018).
[2] Instituto de Ingeniería del Conocimiento (IIC), "Procesamiento del Lenguaje Natural, ¿qué es?" (Octubre, 2017).
[3] SpaCy, spacy.io
[4] NLTK, www.nltk.org
[5] P. Olivier, "Simple NLP Tasks Tutorial", www.ekino.com
[6] TextBlob, textblob.readthedocs.io
[7] Google Translate API, cloud.google.com/translate
[8] Webster, Jonathan., Kit, Chunyu., "Tokenization as the initial phase in NLP", City Polytechnic of Hong Kong (1992)
[9] Craig Trim, "The Art of Tokenization", www.ibm.com (Enero, 2013)
[10] Wikipedia, "Rand Index"
[11] University of Stanford, "Stemming and Lematization", nlp.stanford.edu
[12] Wikipedia, "Lemmatisation"
[13] Susan Li, "Named Entity Recognition with NLTK and SpaCy", towardsdatascience.com
[14] OntoNotes 5, catalog.ldc.upenn.edu/LDC2013T19
[15] brandonrose.org, "Document Clustering with Python"
[16] Chris Emmery, "Euclidean vs. Cosine Distance", cmry.github.io (Marzo, 2017)