CLUSTERING DE NOTICIAS

Recuperación de Información y Minería de Texto

DESCRIPCIÓN BREVE

Poner en práctica diferentes conceptos aprendidos durante el curso y relacionados con la Recuperación de Información y el procesamiento de texto.

Remedios Blázquez Martin y Candela Vidal Rodríguez

Máster en Data Science 2017-2018

Índice

1. Introducción

Nos proporcionan una recopilación de noticias de diferentes fuentes de información, en inglés y español. Se trata de una colección pequeña, en total 22 documentos en formato .html que es necesario procesar hasta obtener un listado de tokens lo más limpio y agrupado posible para obtener una puntuación alta al aplicar un algoritmo de Clustering Aglomerativo.

Antes de comenzar con el detalle del procesamiento de los documentos y su investigación, detallamos las librerías más comunes y más interesantes para hacerlo:

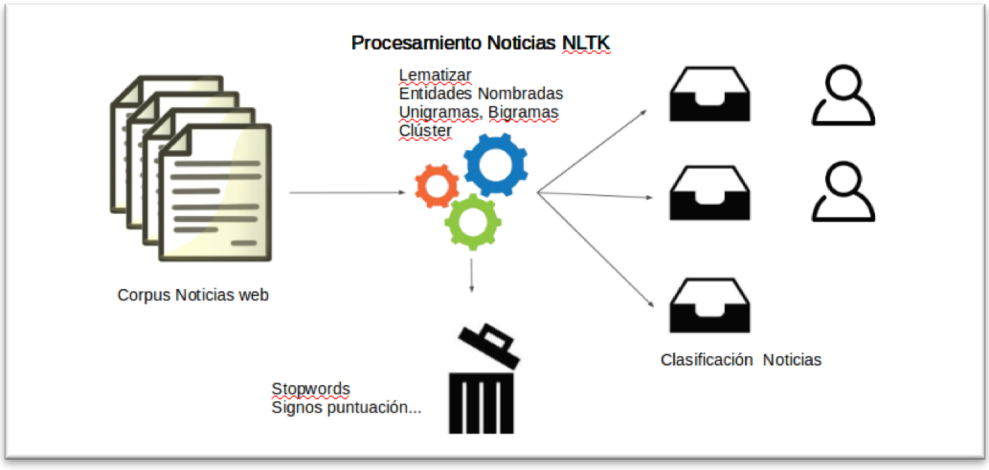
NLTK: Librería principal para el procesamiento de texto en Python. Dispone de un manual online. Posee una interfaz sencilla y cubre la mayor parte de las operaciones. Es una de las más utilizadas. https://www.nltk.org/

Textblob: De nuevo cumple bastantes de las operaciones de procesamiento del lenguaje, está construida sobre NLTK y pretende ser más intuitiva que la ésta. Incluye un módulo de traducción y detección del idioma. http://textblob.readthedocs.io/en/dev/

SpaCy: es la librería más reciente, su interfaz es muy fácil de utilizar. https://spacy.io/

Disponemos de un código fuente inicial en Python, donde se realiza un procesamiento de webs en formato txt y agrupamiento mediante un algoritmo de clustering aglomerativo, se utiliza la librería de Machine Learning “Scikit-Learn” junto con la librería “NLTK”.

A continuación, realizaremos una serie de experimentos sobre el código proporcionado con el único objetivo de limpiar los textos de noticias para obtener el menor número de términos únicos que faciliten la implementación del algoritmo de clustering.



2. Propuestas con experimentos y resultados

• Proceso de conversión de ficheros .html a .txt

Inicialmente debemos convertir los documentos html a txt. Para cada uno de los ficheros que abrimos en modo lectura, nos ayudamos de la librería BeatifulSoup para obtener todos los párrafos y ensamblarlos como un texto. Finalmente, el fichero de texto es generado en forma de escritura y será guardado en la carpeta correspondiente (CorpusNoticiasPractica1718) con la extensión adecuada (.txt).

Este paso no se trata de un experimento con el que se busque aumentar el rand score, sino que es un paso previo necesario sin el cual no es posible ejecutar el código inicial propuesto.

El código de la función utilizada es el siguiente:

```
def text_to_html(folder):

    filenames = os.listdir(folder)
    for file in filenames:

        if file.endswith('.html'):

            file_in = open("./"+folder+ "/" + file, 'rb')

            # Pasamos el contenido HTML a un objeto BeautifulSoup
            html = BeautifulSoup(file_in)
            file_in.close()

            # Obtenemos todos los párrafos para capturar el texto.
            parrafos = html.find_all('p')

            # Generamos el texto a procesar uniendo los párrafos.
            text = ""
            for parrafo in parrafos:
                text = text + parrafo.getText() + "\n"

            if not os.path.isdir('./CorpusNoticiasPractica1718/'):
                os.mkdir('./CorpusNoticiasPractica1718/')

            # Generamos el fichero de CorpusNoticiasPractica1718
            file_out = open("./CorpusNoticiasPractica1718/" \
                        + file.strip(".html") + ".txt", "w")
            file_out.write(text)
            file_out.close()
```

Con la extracción de la etiqueta </p> de los .html se ha obtenido el cuerpo de cada una de las 22 noticias proporcionadas.

Uno de los obstáculos que nos encontramos en este paso fue el uso incorrecto de los argumentos que es necesario pasar a la función BeautifulSoup() del paquete bs4. Inicialmente la llamábamos pasándole como argumentos el texto del fichero original y 'html.parser', esto hacía que los textos resultantes fueran más sucios y con ello empeoraban el funcionamiento del algoritmo de clustering.

- Ejecución del código inicial proporcionado

Se parte de la carpeta CorpusNoticiasPractica1718 donde están almacenados los documentos.txt, se leen los ficheros iterativamente, para cada fichero se hacen las siguientes acciones: se abre con encoding = "latin-1", se lee el fichero y se cierra.

Se tokeniza el texto en palabras, después con nltk.Text pasamos una lista de los tokens a evaluar, siendo el token la unidad básica de información.

Después se llama a la función: cluster_texts, donde le pasamos como parámetro de entrada el número de grupos en los que queremos agrupar los documentos y también le pasamos la medida de similitud, en este caso es la distancia Coseno.

La medida de la similaridad es fundamental en la definición de los Clústeres. Habrá que escogerla cuidadosamente, ya que los resultados dependerán de la medida que hayamos tomado, se escogerá una u otra medida en función de los tipos de datos que estemos tratando.

Se crea una colección con todos los términos, se extraen también los términos únicos y se calcula un array con todos los vectores, después llama a una función TF, donde crea un vector por cada documento, donde aparece la frecuencia del término único en ese documento.

A posteriori, se ejecuta toda la parte correspondiente a la clusterización por temática de las 22 noticias propuestas, obteniendo la siguiente salida:

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 19157 terms.
Unique terms found:  4722
Vectors created.
test:  [0 1 0 0 4 0 0 0 0 1 0 3 1 4 0 0 1 2 0 0 1 1]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  0.02914572864321609

Process finished with exit code 0
```

Observamos que se han procesado los 22 documentos, con 19157 términos, de los cuales 4722 son únicos y con un rand_score: 0.02914572864321609

Utilizando la medida de similaridad del Coseno, la cual varía entre: [-1, 1], donde 1 sería donde los documentos son más similares y –1 más distintos.

También hemos detectado que no comienza a leer por el primero fichero que se encuentra en la carpeta o alfabéticamente como debería ser el caso para que la comparación con el Gold Standar sea efectiva y obtengamos así una puntuación adecuada.

Los experimentos probados a partir de este punto se realizan de manera incremental, es decir que en cada paso mantenemos las modificaciones introducidas en el paso previo, en caso contrario se indicará.

- ## Ordenación adecuada de ficheros

Para procesar los ficheros según la ordenación Gold Standard procedemos a ordenar alfabéticamente la lista que contiene el listado de ficheros .txt de la carpeta CorpusNoticiasPractica1718. Para ello utilizamos la función sorted() de Python aplicable a iterables (en nuestro caso al listado que contiene la relación de los 22 textos de noticias a analizar)

Modificamos nuestro código como se muestra a continuación:

```
listing = sorted(os.listdir(folder))
for file in listing:
```

Como se puede observar, ahora la lista que recorremos para procesar cada uno de los 22 textos estará ordenada alfabéticamente. Con ello, el rand_score que compara los arrays de noticias “test” y “reference” estará siendo calculado adecuadamente.

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 19157 terms.
Unique terms found:  4722
Vectors created.
test:  [4 3 4 1 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 2 0 1]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  0.008040201005025135

Process finished with exit code 0
```

Observamos que el rand_score ha disminuido. Esto va en contra de la definición del algoritmo pues si los textos no se procesan en el orden correcto la comparación de los vectores “test” y “reference” no es lo que se está buscando.

Este experimento se ha realizado ejecutando con la función de distancia coseno (los objetos se consideran vectores, su similaridad se mide por el ángulo que los separa usando el coseno). A continuación, se probará con otra medida de distancia para determinar si con ella se puede obtener una puntuación mayor.

- ## Función de distancia

Si procesamos con la distancia euclídea:

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 19157 terms.
Unique terms found:  4722
Vectors created.
test:  [0 3 0 1 0 0 0 0 1 4 0 0 0 4 0 0 1 0 0 2 0 4]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  -0.00915331807780317

Process finished with exit code 0
```

El rand_score nos está indicando que el ajuste entre el vector observado y el predicho por nuestro clúster está mucho más distante, con este cambio efectuado nos estamos alejando del objetivo, de esta manera pensamos que el utilizar la función Euclídea no nos favorece.

La distancia euclídea representa lo diferentes que son los grupos o clústeres y tiene otro rango de medida que la medida del coseno.

Para esta fuente de información de datos que nos están aportando, como es un corpus de texto, la función de distancia óptima a utilizar es la distancia coseno, por ello seguimos nuestros experimentos con ella.

- ## Traducción de textos

Tal y como hemos visto en el análisis previo de los textos, contamos con noticias en dos idiomas diferentes. Esto creemos penalizará el score a la hora de aplicar el algoritmo de clustering pues habrá términos que, aun significando lo mismo, serán considerados como palabras diferentes por el hecho de estar en idiomas distintos.

Por ello optamos por implementar una traducción previa de los textos. De esta manera, a la hora de agrupar por unidades de lenguaje o tokens, pensamos que éstos serán más uniformes.

El código de la función utilizada es el siguiente:

```
def translate_text(text):

    if TextBlob(text).detect_language() != 'en':
        return TextBlob(text).translate(to = "en").string
    else:
        return text
```

Utilizamos la librería TextBlob para traducir cada texto completo en el caso de que su idioma sea distinto de inglés. Esta librería ofrece un servicio de traducción automática que además también facilita la detección del idioma del texto en cuestión.

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 19138 terms.
Unique terms found:  4048
Vectors created.
test:  [4 2 4 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 3 0 0]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  0.046215139442231094

Process finished with exit code 0
```

Como cabía de esperar los términos únicos han disminuido y el rand_score ha aumentado. Ésta métrica todavía es demasiado cercana a cero, y dicho valor nos indica que debemos seguir aplicando métodos de procesamiento de texto con el fin de limpiarlo al máximo.

- ## Encoding de lectura de ficheros

Hemos visto que en los textos aparecen caracteres extraños no legibles. Creemos que esto puede ser debido a la codificación con la que son leídos los textos. Por ello probamos a cambiar el enconding de apertura de los ficheros de “latin-1” a “utf-8”.

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 19535 terms.
Unique terms found:  3880
Vectors created.
test:  [4 2 4 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 3 0 0]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  0.046215139442231094

Process finished with exit code 0
```

No observamos ningún cambio en el rand_score, pero vemos que los términos únicos han disminuido considerablemente y ya no apreciamos caracteres extraños. Esto a la hora de vectorizar nuestros documentos nos discrimina bastante, con lo cual pensamos que para el algoritmo de clustering utilizado será más óptimo.

- ## Limpieza de Stopwords

Las Stopwords son palabras que no tienen ningún significado útil y que no nos aportan nada al análisis por temática de los textos. Palabras como determinantes, preposiciones, pronombres ..., se pueden considerar unidades gramaticales que no contribuyen al significado del mensaje, sino que son soportes gramaticales del lenguaje.

La propia librería NLTK ofrece un listado de dichas palabras según el idioma con el que se esté tratando. En nuestro caso, al trabajar con textos en inglés, declaramos las stopwords en dicho idioma y las guardamos en una variable lista que utilizaremos para depurar el texto.

Implementamos una función en la cual recorremos cada uno de los tokens del texto y nos quedamos solo con aquellos que no se encuentren en la lista de "stop_words" en inglés definida en la librería NLTK. El código de dicha función es el siguiente:

```
def remove_stopwords(tokens):

    not_stopwords_tokens = []
    for token in tokens:
        token = token.lower()
        if token not in set(stopwords.words('english')):
            not_stopwords_tokens.append(token)

    return not_stopwords_tokens
```

El resultado obtenido se muestra a continuación:

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 12504 terms.
Unique terms found:  3751
Vectors created.
test:  [0 2 3 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 4 0 0]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  0.014048059149722743

Process finished with exit code 0
```

Observamos que, aunque han disminuido los términos únicos, el rand_score también ha disminuido. Esto no implica necesariamente que la eliminación de stop_words no sea una parte clave en el procesamiento de los textos, sino que es necesario utilizarla en conjunto con otros procesos de normalización de textos.

Esto nos lleva a seguir implementando nuevos algoritmos unificación de los tokens de las noticias.

- ## Limpieza de puntuación, dígitos y caracteres no ascii o distintos de letras.

Pensamos que cuanto más limpios y normalizados estén los textos, más óptimo será el algoritmo de clustering. Por ello creamos una función en la cual nuevamente recorremos cada uno de los tokens de los textos y nos quedamos solo con aquellos que cumplen las siguientes condiciones:

- No contienen dígitos
- No contienen signos de puntuación
- No contienen caracteres que no sean lentras del alfabeto de la “a” a la “z”.

```
def clean_others(tokens):
    others = list(string.digits + string.punctuation)

    no_others = []
    for token in tokens:
        if token not in others and re.search('[a-zA-Z]', token):
            no_others.append(token)

    return no_others
```

Obtenemos los resultados siguientes:

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 10030 terms.
Unique terms found:  3625
Vectors created.
test:  [4 0 4 4 2 1 1 1 2 0 0 0 0 0 0 0 0 2 3 1 0]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  0.5671309192200557

Process finished with exit code 0
```

Observamos un aumento considerable del rand _score, no así una reducción de términos únicos. Esto puede ser debido a que con este experimento hemos realizado una limpieza más exhaustiva de la unidad básica de información, también conocido como token.

- ## Lematizar

Proceso sintáctico que consiste en obtener el lema correspondiente de una palabra, es decir, en obtener el término común que representaría a la entrada de dicha palabra en el diccionario, a la vez que reduce el tamaño del vocabulario y mejora el

tiempo de procesado de los documentos o noticias, a veces lematizar no agrupa palabras que deberían estarlo y o al contrario mostrar iguales palabras cuando no lo son. No obstante, vamos a probar el algoritmo y ver que resultados tiene.

Nuevamente haciendo uso de la librería NLTK, es este caso del módulo WordNetLemmatizer, definimos una serie de funciones para sacar los lemas correspondientes de las palabras tratadas y observar si con esta práctica mejora la puntuación del algoritmo de clustering.

Utilizamos las siguientes funciones para obtener los lemas de cada uno de los tokens que componen los textos:

```
def wordnet_value(value):

    if value.startswith('J'):
        return wordnet.ADJ
    elif value.startswith('V'):
        return wordnet.VERB
    elif value.startswith('N'):
        return wordnet.NOUN
    elif value.startswith('R'):
        return wordnet.ADV
    else:
        return ''

def lemmatize(tokens):
    tokens = nltk.pos_tag(tokens)
    wn_lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = []

    for token in tokens:
        if len(token) > 0:
            pos = wordnet_value(token[1])

            if pos != '':
                lemma = wn_lemmatizer.lemmatize(str(token[0]).lower(),
pos=pos)
                lemmatized_tokens.append(lemma)

    return lemmatized_tokens
```

Se implementan una función de lematización, en la cual inicialmente lo que hacemos es taggear la palabra (consiste en ponerle una etiqueta a cada token, clasificando la palabra como Verbo, Determinante, Adverbio...), para posteriormente quedarnos con el segundo elemento de la tupla y descartar aquellas etiquetas que no nos interesa, nos vamos a quedar con: Nombres, Adverbios, Verbos y Adjetivos.

Los tokens lematizados los guardamos en una lista para posteriormente procesarlos bajo el algoritmo de clusterización.

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 9098 terms.
Unique terms found:  2813
Vectors created.
test:  [4 0 4 4 2 3 3 3 2 0 0 0 0 0 1 1 0 1 2 1 3 0]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  0.6964520367936924

Process finished with exit code 0
```

Se obtiene otra disminución de términos únicos o palabras claves y aumento del rand_score que nos indica que el ajuste entre los observado y los predicho por nuestro algoritmo se va aproximando, apreciando divergencia en 2 o 3 noticias solamente.

- Stemming

Esta práctica consiste en reducir la palabra hasta su raíz o stem. Con ello se obtiene la mínima expresión con significado que favorecerá la agrupación de palabras y con ello los términos únicos, aumentando así el rand_score.

Existen varios algoritmos de stemming implementados en la librería NLTK, nosotras hemos optado por el algoritmo de Porter por ser el más comúnmente utilizado.

El código de la función encargada de realizar esta tarea en nuestro programa es el siguiente:

```
def stemming(tokens):
    stemmer = PorterStemmer()
    stemmer_token = []
    for token in tokens:
        stems = stemmer.stem(token)
        stemmer_token.append(stems)
    print("los steams:", stemmer_token , "\n", len(stemmer_token))
    tokens = stemmer_token
    return tokens
```

Implementamos una función que lo que hace es de los tokens que ya hemos lematizados anteriormente los reduce a su raíz, y los guardamos en una lista, la cual será luego pasada como argumento para el procesamiento del clúster.

Resultado:

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 9099 terms.
Unique terms found:  2565
Vectors created.
test:  [4 2 4 4 1 3 3 3 1 2 2 2 2 2 0 0 0 0 1 0 3 2]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  0.815913688469319

Process finished with exit code 0
```

Observamos que ha aumentado claramente el rand_score y han disminuido los términos únicos. A continuación, ofrecemos una explicación más detallada.

- TF y TF_IDF

En este código aportado previamente, vamos a clasificar (algoritmo de clúster) nuestras noticias mediante el espacio vectorial, donde cada noticia se representa por vectores de términos únicos.

Para el cálculo de los vectores es fundamental dar un peso a cada término, el cual representa la importancia en el documento y/o corpus (conjunto de noticias).

Existen varias métricas fundamentales para obtener dicho peso:

Booleano: Si la palabra aparece o no en la noticia (1 ó 0).

Frecuencia del término (TF): Número de veces que aparece el término en la noticia, se puede decir que cuántas más veces aparezca el término en la noticia más importante será.

Frecuencia del término-Frecuencia Inversa del Documento (TF-IDF): Combina la frecuencia del término en la noticia con la frecuencia de este término en el resto de documentos de la colección. Ejemplo: el término "Merkel" será representativo de la noticia: "'Mini Merkel' given crucial...", pero siempre y cuando el término "Merkel" no tenga una frecuencia alta en las otras noticias del corpus, será un término muy discriminatorio para la clasificación del algoritmo.

Medidas de similitud entre documentos

La similitud entre dos objetos, en nuestro caso dos vectores que representan noticias, es una cantidad numérica que determina el grado de semejanza de estos objetos (documentos). La similitud aumentará cuando más parecida sean las noticias evaluadas.

En nuestro código nos aportan dos medidas:

Coseno, es una medida de similitud entre dos vectores. Esta medida varía entre: [-1 ,1]. En minería de textos se aplica la similitud coseno con el objeto de establecer una métrica de semejanzas entre textos, se emplea como indicador de cohesión de clusters de textos.

Euclídea, es una medida de disimilitud, es decir de distancia. Cuanto más bajo sea el valor más parecido son los objetos y al contrario cuanto mayor sea el valor más diferente serán los objetos. No puede ser negativa.

Con todo ello, haremos varios experimentos:

a. Coseno y TF.

Este experimento es el que se ha venido desarrollando a lo largo de los puntos anteriores, desarrollamos su explicación aquí:

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 9099 terms.
Unique terms found:  2565
Vectors created.
test:  [4 2 4 4 1 3 3 3 1 2 2 2 2 2 0 0 0 0 1 0 3 2]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  0.815913688469319

Process finished with exit code 0
```

El rand_score nos indica que la similaridad de noticias es muy próxima (a pesar de que no se ha llegado a la similaridad total igual a 1) y que los diferentes clústeres son muy distintos.

Esto también ha sido provocado por el peso de cada uno de los términos en cada noticia, teniendo en cuenta que para el peso del término en cada noticia tomamos la métrica de la frecuencia del término en el documento (TF), podemos pensar que existen términos con alta frecuencia en cada noticia que provocan que la clasificación dentro de los grupos sea más homogénea y muy diferente entre ellos, llegando a clasificar noticias muy similares dentro de cada grupo.

Observamos que los clústeres se han originado de esta manera:

- Clúster 0: agrupa noticias referentes a Oxfam.
- Clúster 1: agrupa noticias referentes al Vaticano.
- Clúster 2: agrupa noticias sobre el avión siniestrado en Irán.
- Clúster 3: agrupa noticias sobre la Independencia de Cataluña.
- Clúster 4: agrupa noticias sobre Angela Merkel.

¿Qué ocurre si esta lista de noticias no estuviera ordenada, a pesar de que le hemos implementado todos los anteriores experimentos?

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 9099 terms.
Unique terms found:  2565
Vectors created.
test:  [2 2 0 0 4 3 3 2 1 0 1 2 2 4 3 2 4 0 3 0 2 1]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  0.022926500337154435

Process finished with exit code 0
```

Para empezar el rand_score está muy próximo a 0, nuestro clasificador ha hecho diferentes grupos en los cuales se observa que dentro de cada grupo las noticias son distintas y dispares entre sí.

La ordenación previa en las noticias tiene un impacto importante a la hora de clasificarlas.

b. Coseno y TF-IDF.

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 9099 terms.
Unique terms found:  2565
Vectors created.
test:  [4 3 4 4 1 2 2 2 1 3 3 3 3 0 0 0 0 0 1 0 2 3]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  0.6571428571428571

Process finished with exit code 0
```

Aquí seguimos hablando de similaridad entre noticias, pues estamos utilizando la medida del Coseno, también se aprecia que no se llega a una similaridad total entre las noticias, pero sigue siendo fuerte llegando a un 0.65714.

Hemos utilizado para el peso del término la métrica TF-IDF, en este caso los términos más discriminatorios serán aquellos que tengan un peso mayor y son aquellos que tienen frecuencias altas para una determinada noticia y frecuencia baja en el resto de documentos, aquí nos da a pensar que hay un volumen de términos que tienen frecuencias altas en una determinada noticia, pero a la vez frecuencia alta en el resto de noticias, provocando que los grupos no sean tan homogéneos dentro de ellos, como es el caso del clúster 0 en este caso concreto.

Clúster 0 agrupa noticias:

- Mueren 66 personas al estrellarse un avıвn comercial en Iran.
- Oxfam Haiti scandal_ Suspects 'physically threatened' witnesses - BBC News.
- Oxfam Says Workers in Haiti Threatened Witness to Misconduct - The New YorkTimes.
- Oxfam tendrá una comisión independiente para evitar los abusos.txt
- Oxfam was warned a decade ago about the crisis in sex abuse among world's aid workers

¿Qué ocurre si esta lista de noticias no estuviera ordenada, pero manteniendo todos los experimentos anteriormente implementados?

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 9098 terms.
Unique terms found:  2565
Vectors created.
test:  [3 3 0 0 4 2 2 3 1 0 1 3 3 4 2 3 4 0 2 0 0 1]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  0.0

Process finished with exit code 0
```

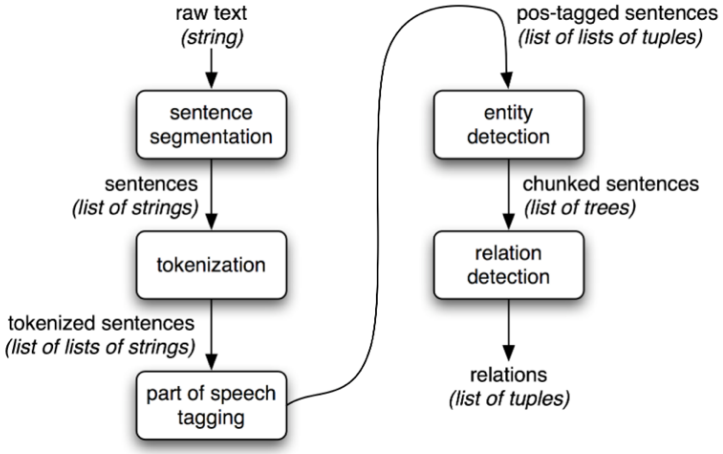
Se confirma de nuevo que la ordenación es significativa para obtener una mejor clasificación de las noticias.

¿Qué tendríamos que hacer para mejorar el score? Nos interesaría reducir el peso de los términos que son comunes en el conjunto de noticias frente a las palabras claves que caracterizan cada noticia.

Para ello vamos a implementar el algoritmo de Entidades Nombradas únicas.

• Entidades Nombradas

Con el fin de obtener programáticamente los sujetos de un texto (en este caso los temas de las noticias a analizar), exploramos la posibilidad de la obtención de Entidades Nombradas únicas de cada uno de los textos.



La extracción de Entidades nombradas consiste en buscar menciones de entidades potencialmente interesante en cada una de las frases.

Primero se segmenta y etiqueta las entidades que pueden formar parte de conjuntos relacionados entre sí. Por lo general, estas palabras candidatas a entidades nombradas suelen ser sustantivos o nombres propios. También puede ser interesante definir chunks de nombres indefinidos que no tienen por qué referirse a entidades directamente.

Por último, buscamos relaciones entre parejas de entidades que se den próximas dentro del texto, y usamos el patrón para crear tuplas que guarden la relación entre dichas entidades.

A continuación, mostramos el código de la función utilizada para recorrer el árbol creado para cada una de las frases del texto. Para generar dicho árbol, hacemos uso del clasificador ya entrenado que nos ofrece NLTK "nltk.ne_chunck_sents", donde definiremos el parámetro binary = True, para que las entidades nombradas sean etiquetadas únicamente como "NE", si no se utilizarían etiquetas de categorías más concretas.

```
def named_entities(tree):
    ne = []
    if hasattr(tree, 'label') and tree.label:
        if tree.label() == 'NE':
            for child in tree:
                ne.append(' '.join(child[0]))
        else:
            for child in tree:
                ne.extend(named_entities(child))
    return ne
```

El resultado obtenido, únicamente ordenando adecuadamente los textos, traduciéndolos todos a inglés y extrayendo sólo las entidades nombradas es el que se muestra en la siguiente captura:

```
Prepared  22  documents...
They can be accessed using texts[0] - texts[21]
Created a collection of 1469 terms.
Unique terms found:  525
Vectors created.
test:  [1 4 1 1 0 2 2 2 0 4 4 4 4 4 3 3 3 3 0 1 2 4]
reference:  [0, 5, 0, 0, 0, 2, 2, 2, 3, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 0, 2, 5]
rand_score:  0.9142857142857143
```

Observamos como los términos únicos han disminuido considerablemente hasta los 525 y que el rand score ha aumentado hasta 0.91, alcanzando casi la unidad. Estos resultados nos hacen pensar que probablemente este sea el camino a seguir cuando lo que se busca es procesar textos en bruto para poder aplicar sobre ellos algoritmos de clsutering.

Comparando el vector de test con el de reference podemos ver que la única noticia que el algoritmo no ha sabido clasificar adecuadamente es la quinta de la lista “As Merkel calls on Pope Francis, is a partnership in the Works”. Leyendo la noticia, nos damos cuenta de que ésta puede resultar confusa para el algoritmo de clustering ya que contiene dos de los 5 temas posibles, en lugar de poder encasillarla claramente en uno de esos cinco temas.

El código del programa principal que llama a cada una de las funciones correspondientes con los experimentos detallados en la memoria es el que sigue:

```
# Traducir a ingles
text = translate_text(raw)

# Tokenizamos por frase
sentences = nltk.sent_tokenize(text)

# Tokenizamos por palabra
tokenized_sentences = [nltk.word_tokenize(sentence) for sentence in sentences]

final_tokens = []

# Si utilizamos entidades nombradas

if named_ent:
    tagged_sentences = [nltk.pos_tag(sentence) for sentence in tokenized_sentences]
    chunked_sentences = nltk.ne_chunk_sents(tagged_sentences, binary=True)

    for sentence in chunked_sentences:
        final_tokens.extend(named_entities(sentence))

# Si no utilizamos entidades nombradas
else:

    for tokens in tokenized_sentences:

        tokens = remove_stopwords(tokens)
        tokens = clean_others(tokens)
        tokens = lemmatize(tokens)
        tokens = stemming(tokens)
        final_tokens.extend(tokens)

text = nltk.Text(final_tokens)
texts.append(text)
```

Para cada una de las noticias comenzamos traduciendo el texto y seguidamente tokenizamos por frase y por palabra. A continuación, existen dos posibles caminos dependiendo de si queremos utilizar entidades nombradas o no:

- En caso afirmativo, tageamos los tokens de cada frase y generamos los chunks para después recorrer el árbol para cada una de las frases y generar los tokens finales.

- Si no buscamos obtener entidades nombradas únicas, simplemente por cada conjunto de tokens eliminamos stopwords, limpiamos puntuación y caracteres no deseados, lematizamos y aplicamos stemming. Por último, también obtenemos los tokens finales con los que alimentar el algoritmo de clustering.

3. Conclusiones

A continuación, resumimos el Rand Score y número de términos únicos obtenidos tras la implementación de cada uno de los experimentos descritos anteriormente:

Propuestas	Términos Únicos	Rand Score
Código Inicial	4722	0.02914572864321609
Ordenación adecuada	4722	0.008040201005025135
Función de distancia Euclídea	4722	-0.00915331807780317
Traducción de textos a inglés	4048	0.046215139442231094
Encoding	3880	0.046215139442231094
Limpieza de Stopwords	3751	0.014048059149722743
Limpieza de puntuación y caracteres no ascii	3625	0.5671309192200557
Lematizar	2813	0.6964520367936924
Stemming + Coseno + TF	2565	0.815913688469319
Coseno (TF_IDF)	2813	0.6571428571428571
Entidades Nombradas	525	0.9142857142857143

A la vista de los experimentos realizados, llegamos a esta batería de conclusiones:

1. La ordenación previa en las noticias tiene un impacto importante a la hora de clasificarlas. De no realizar este paso, es inútil realizar la comparación con el Gold Standard y con ello la obtención del Rand Score, pues no tendría sentido.
2. El algoritmo de Lemmatización para obtener la etiquetación morfosintáctica de los tokens es una pieza clave. Con él se generan los lemas de palabras claves que serán candidatas para discriminar nuestras noticias facilitando también la eliminación de aquellas palabras comunes a todas las noticias.
3. La información semántica que obtengamos, específicamente las relaciones semánticas, contribuyen a mejorar los resultados de nuestro algoritmo de clasificación.

4. El experimento de Entidades Nombradas ha sido aquel con el que se han obtenido el menor número de términos únicos y el mayor rand_score (cercano a uno). Dicho experimento se ha realizado sin ser complementado con ninguno de los otros experimentos (a excepción de la traducción y ordenación de textos).

Por ello se concluye que el mejor algoritmo de tratamiento del texto previo a la ejecución del algoritmo de clustering ha de ser: ordenación de textos de acuerdo con el Gold Standard + traducción de textos en español a inglés + selección de entidades nombradas únicas como términos únicos.