Descubrimiento de información en textos. Tema 5. Representación de textos

Eva Sánchez Salido

7 de marzo de 2021

En este trabajo generaremos representaciones de un conjunto de textos mediante dos tipos de funciones de pesado, una local y otra global. Para ello utilizaremos distintas librerías de Python, como **Beautiful Soup** para la extracción de datos en documentos HTML y **nltk** para tokenizar los textos y realizar un análisis léxico.

El código creado para realizar esta tarea se comenta a lo largo de la entrega y está disponible para su uso aquí.

1. Procesamiento de documentos HTML

Para esta tarea se han elegido diez artículos de Wikipedia que tratan temas variados: la epidemia del baile de 1518, los estándares de jazz, el flamenco, el teorema del mono infinito, el número de Erdős, el efecto tetris, un dicho popular usado en lingüística para ejemplificar la ambigüedas sintáctica, el ironing extremo, el cocido madrileño y la lasaña, todos ellos en inglés.

Cuando trabajamos con un conjunto pequeño de documentos (en este caso páginas web) es posible abordar la obtención del código fuente de cada página simplemente descargándola manualmente. El método adoptado en esta entrega busca aligerar este proceso cuando se trata con cantidades grandes de documentos, de manera que el programa lo único que requiere es una lista que contenga las direcciones de las páginas de las que queremos extraer información. Así, mediante el módulo `urllib.request` de la librería **request** es posible abrir los url y almacenar el código HTML en archivos de texto.

En la lista `docs` almacenamos las rutas de los archivos que contienen el código HTML de cada página, ya que serán útiles más adelante.

```
import urllib.request

urls = ['https://en.wikipedia.org/wiki/Dancing_plague_of_1518', 'https://en.wikipedia.org/wiki/Infinite_monkey_theorem', 'https://en.wikipedia.org/wiki/Tetris_effect', 'https://en.wikipedia.org/wiki/Time_flies_like_an_arrow;_fruit_flies_like_a_banana', 'https://en.wikipedia.org/wiki/Jazz_standard', 'https://en.wikipedia.org/wiki/Extreme_ironing', 'https://en.wikipedia.org/wiki/Flamenco', 'https://en.wikipedia.org/wiki/Erd%C5%91s_number', 'https://en.wikipedia.org/wiki/Cocido_madrile%C3%B1o', 'https://en.wikipedia.org/wiki/Lasagne']
docs = []
for i in range(len(urls)):
  # stores HTML code of urls[i] in text file 'text[i].txt'
  url = urls[i]
  text_file = 'html' + str(i) + '.txt'
  urllib.request.urlretrieve(url, text_file) # this creates a .txt document with the HTML source, named html[i].txt
  # create a list of directions of the html txt files
  docs.append('/content/' + urllib.request.urlretrieve(url, text_file)[0])
```

Procedemos a limpiar los documentos HTML de manera que para cada archivo obtengamos uno nuevo que contenga tan sólo el título y los párrafos que componen el documento. Para ello usamos la librería **Beautiful Soup** [3] de Python, que sirve para extraer datos de documentos HTML y XML.

```
from bs4 import BeautifulSoup
texts_dirs = []
for i in range(len(docs)): # cleaning each HTML document
  current = docs[i]
  with open(current) as doc:
    soup = BeautifulSoup(doc, "html.parser")

  title = soup.head.title.text # returns title inside <head>
  paragraphs = soup.find_all("p") # returns a list of elements inside tags containing 'p'

  file_name = 'text' + str(i) + '.txt' # creating new text file with cleaned text
  f = open(file_name, "w+")
  f.write(title + ' ')
  for p in range(len(paragraphs)): # for every paragraph gets the plain text without tags
    f.write(paragraphs[p].get_text())
  # create a list of directions of the text files
  texts_dirs.append('/content/' + file_name)
  f.close()
```

2. Análisis léxico

Ahora los archivos contienen tan sólo la información que buscamos analizar y podemos comenzar a trabajar con ellos. Lo primero será realizar un análisis léxico con el que seleccionaremos las características con las que generaremos las representaciones. Usaremos la librería **nltk** [1] para tokenizar los textos y realizar el análisis léxico. En este paso realizaremos la selección de características eliminando las etiquetas menos relevantes: determinantes (DT), preposiciones y conjunciones subordinantes (IN), conjunciones coordinantes (CC) y cardinales y numerales (CD). Aprovechamos también para eliminar signos de puntuación (puntos, comas, guiones, comillas, corchetes, etc).
Lo que obtenemos después de este paso es la lista `relevant_words` que a su vez contiene una lista por documento con el conjunto de palabras relevantes.

```
import nltk
nltk.download() # select d (download) + all
from nltk.tokenize import sent_tokenize, word_tokenize
import string

relevant_words = []
for i in range(len(texts_dirs)):
  current = texts_dirs[i]
  with open(current) as doc:
    tokens = word_tokenize(doc.read())
    pos_tags = nltk.pos_tag(tokens)
    less_pos_tags = pos_tags.copy() # copy tags list so that we can edit it
    k = 0 # counter to remove items appropiately
    delete_str = 'DT IN CC TO CD' + string.punctuation + '`` '' "'
    for i in range(len(pos_tags)-1):
      if pos_tags[i][1] in delete_str:
        less_pos_tags.pop(i-k)
        k = k + 1
    relevant_words_here = [i[0] for i in less_pos_tags]
    # relevant_words is a list of list
    # each list contains the relevant words of a document, given by
        relevant_words_here
    relevant_words.append(relevant_words_here)
    # relevant_words[0] gives the relevant words of the first document, etc
```

3. Eliminación de stop-words

Pasamos a eliminar las palabras vacías o *stop-words*, palabras que se consideran ruido en los textos. Algunos ejemplos en inglés son *is*, *am*, *this*, *a*, etc. La librería **nltk** contiene una lista de stop-words en diversos idiomas. La descargamos y filtramos el conjunto de palabras que hemos obtenido en el paso anterior.

```
from nltk.corpus import stopwords

stop_words = stopwords.words('english')

filtered_relevant_words = []
for set in relevant_words:
  for word in set:
    if word not in stop_words:
      filtered_relevant_words.append(word)

# create vocabulary with the filtered relevant words
f = open('vocabulary.txt', "w+")
for w in filtered_relevant_words:
  f.write(w + ' ')
f.close()
# we've created a txt document containing the vocabulary of all documents, with
    repeated and not truncated words
```

Obtenemos un documento de texto que llamamos `vocabulary`, que contiene el conjunto de todas las palabras "relevantes" de todos los documentos, sin truncamiento y sin la eliminación de palabras repetidas.

4. Stemming y eliminación de palabras repetidas

El siguiente paso es eliminar los sufijos de las palabras que conforman el vocabulario mediante el algoritmo de Porter [2]. Extraemos el código directamente desde tartarus.org/martin/PorterStemmer/python.txt y truncamos las palabras mediante una lista por comprensión.

```
p = PorterStemmer()
stemmed_words = [p.stem(k,0,len(k)-1) for k in filtered_relevant_words]
```

El último paso para obtener el vocabulario final es eliminar las palabras que se repiten. Lo primero que haremos es convertir las letras mayúsculas en minúsculas. Después usaremos la función **fromkeys()**, que genera un diccionario a partir de las entradas especificadas, con lo cual no incluye duplicados. Después bastará convertir de nuevo el diccionario en una lista.

```
lower_cased = [i.lower() for i in stemmed_words]
final_vocabulary = list(dict.fromkeys(lower_cased))
```

Y finalmente generamos un documento de texto con las palabras del vocabulario separadas por espacios.

```
# create a .txt for the final vocabulary
f = open('final_vocabulary', "w+")
for w in final_vocabulary:
  f.write(w + ' ')
f.close()
# vocabulary containing all words appearing in the documents, stemmed and with no
    repetitions
```

5. Funciones de pesado

Vamos a crear las representaciones vectoriales de los textos. Generaremos una local, donde cada fila del documento representará la frecuencia de aparición de cada término del vocabulario en dicho documento, y una global, donde cada fila representará el valor TF-IDF de cada palabra del vocabulario en cada documento.

5.1. Local

Calculamos la **frecuencia** de cada palabra del vocabulario en cada documento por separado, sin más que contar las veces que aparece cada palabra en el documento y dividiendo por el número total de palabras en el documento.

```
f1 = open('local_term_frequencies.txt', 'w+')
term_frequencies = [] # this will be a list of lists, containing tf for each document,
    useful for creating the inverse document
for i in range(len(texts_dirs)):
  current = texts_dirs[i]
  with open(current) as doc:
    splitted = list(doc.read().split())
    lowered = [i.lower() for i in splitted]
    p = PorterStemmer()
    text = [p.stem(k,0,len(k)-1) for k in lowered] # list of words of the current text
        , lowercased and stemmed
    count_dic_text = {i:text.count(i) for i in final_vocabulary} # dictionary
        containing all words of the vocabulary and number of occurrences in current
        document
    term_frequencies_doc = [] # list containing tf of current document
    n = len(text) # number of words in the given text
    for word in final_vocabulary:
      term_frequencies_doc.append(count_dic_text[word]/float(n))
    term_frequencies.append(term_frequencies_doc)
```

```
16      for tf in term_frequencies_doc:
17        f1.write(str(tf) + ' ')
18      f1.write('\n')
19  f1.close()
```

Obtenemos un fichero de texto donde cada fila representa las frecuencias de aparición de cada palabra del vocabulario en cierto documento. Es decir, dado el vocabulario

```
word_1 word_2 word_3 ... word_n
```

el fichero `term_frequencies` representa

```
tf_word_1_in_doc_1 tf_word_2_in_doc_1 ... tf_word_n_in_doc_1

tf_word_1_in_doc_2 tf_word_2_in_doc_2 ... tf_word_n\in_doc_2

...

tf_word_1_in_doc_m tf_word_2_in_doc_m ... tf_word_n\in_doc_m
```

donde n es el número de palabras que componen el vocabulario y m el número de documentos.
Con estos datos construimos el fichero invertido, con la siguiente estructura:

```
word_1: tf_in_doc_1 tf_in_doc_2 ... tf_in_doc_m

word_2: tf_in_doc_1 tf_in_doc_2 ... tf_in_doc_m

...

word_n: tf_in_doc_1 tf_in_doc_2 ... tf_in_doc_m
```

```
1  f2 = open('fichero_invertido.txt', 'w+')
2  inversed = [[str(i) + ':'] for i in final_vocabulary]
3  for i in range(len(final_vocabulary)):
4    for j in range(len(term_frequencies)):
5      inversed[i].append('doc' + str(j) + '=' + str(term_frequencies[j][i]))
6    f2.write(str(inversed[i]) + '\n')
7  f2.close()
```

5.2. Global

Por último calculamos los valores TF-IDF (Frecuencia del Término × Frecuencia inversa de Documento). En primer lugar se obtienen las frecuencias de los términos en la colección (DF), contando el número de documentos en los que aparece cada término del vocabulario. Después las frecuencias inversas (IDF) se calculan como $log(\frac{N}{1+df(t_i)})$, donde N es el número de documentos. Finalmente multiplicamos estos valores por las TF que hemos calculado anteriormente para obtener un fichero con los valores TF-IDF.

```
# First: For each word of the vocabulary, number of documents that contain it (DF)
N = len(urls)
df = []
for j in range(len(final_vocabulary)):
  count = 0
  for i in range(len(term_frequencies)):
    if term_frequencies[i][j] != 0:
      count = count + 1
  df.append(count)

# Inverse document frequencies (IDF)
import math

idf = [math.log(N/(1 + df[i])) for i in range(len(final_vocabulary))]

#TFIDF

f3 = open('tfidfs.txt', 'w+')
for i in range(len(term_frequencies)):
  for j in range(len(final_vocabulary)):
    f3.write(str(term_frequencies[i][j]*idf[j]) + ' ')
  f3.write('\n')
```

Obtenemos un documento de texto que la fila i contiene los valores TF-IDF de cada término del vocabulario respecto en el documento i.

Referencias

[1] Steven Bird, Ewan Klein y Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. .°'Reilly Media, Inc.", 2009 (vid. pág. 3).

[2] Martin F Porter. «An algorithm for suffix stripping». En: *Program* (1980) (vid. pág. 4).

[3] Leonard Richardson. «Beautiful soup documentation». En: *April* (2007) (vid. pág. 2).