Universidad
Internacional
Menéndez Pelayo

Máster Universitario en Investigación en Inteligencia Artificial

Curso 2020-2021

Recuperación y extracción de información, grafos y redes sociales

Práctica Bloque II: Recuperación de información y minería de texto

19 de abril de 2021

Laura Rodríguez Navas
DNI: 43630508Z
e-mail: rodrigueznavas@posgrado.uimp.es

Índice

1. Resumen **3**

2. Rastreador web (crawler) **3**

3. K-Means **5**
3.1. Datos de entrada 5
3.2. Palabras vacías, stemming y tokenización 5
3.3. Extracción de características 6
3.4. Entrenamiento del algoritmo 7
3.5. Evaluación 9

Bibliografía **9**

1. Resumen

En esta práctica se ha implementado un rastreador web (crawler) en Python [1] (ver Sección 2), que se complementa con un proceso de agrupamiento (ver Sección 3), también implementado en Python, de la información extraída por el rastreador web.

2. Rastreador web (crawler)

En esta sección se describe como se ha implementado el rastreador web (crawler) en Python usando la librería Scrapy. Para empezar con la implementación se debe ejecutar el siguiente comando:

```
$ scrapy startproject books
```

Este comando crea un proyecto Scrapy en el directorio books, siguiendo la estructura por defecto, común para todo proyecto Scrapy. También crea el fichero *scrapy.cfg*, que contiene el nombre del módulo en Python que define la configuración del proyecto books (*books.settings*). El proyecto lo he nombrado books, porqué se rastreará y se recuperará información de un catálogo de libros que se encuentra en la página web: http://books.toscrape.com.

Una vez se ha creado el proyecto, se tienen que definir los ítems de cada libro que se quieran extraer del catálogo. En este caso los ítems que se van a extraer son: el título, la categoría, la descripción, el precio y la valoración de cada libro. Para ello, se tiene que modificar el fichero *books/items.py*, para incluir los cinco ítems que se quieren extraer. Vemos el contenido de *items.py* a continuación:

```
import scrapy

class BooksItem(scrapy.Item):
  # define the fields for your item here like:
  # name = scrapy.Field()
  title = scrapy.Field()
  category = scrapy.Field()
  description = scrapy.Field()
  price = scrapy.Field()
  rating = scrapy.Field()
```

El siguiente paso es describir la manera de extraer la información definida en el fichero *items.py*. Para ello, se utilizan reglas de expresión XPath y CSS. Por ejemplo, si nos fijamos en el código HTML de uno de los libros que se van rastrear (ver Figura 1), veremos que el título del libro es fácil de extraer con la siguiente regla de expresión CSS: **"h1 ::text"**. Pero cuando la extracción de información se complica un poco más, se usan reglas de expresión XPath. Por ejemplo, para extraer las descripciones de todos los libros se usará la regla de expresión: **"//div[@id='product_description']/following-sibling::p/text()"**. Una vez, definidas todas las reglas de expresión para cada ítem que se va a rastrear, se crea la araña *books/spiders/books_toscrape.py*.

Definición 1. *Las arañas son clases que definen cómo se rastrea una página web determinada (o un grupo de páginas web), incluido cómo realizar el rastreo y cómo extraer la información deseada. En otras palabras, las arañas son el lugar donde se define el comportamiento personalizado para rastrear y analizar las páginas web.*

En el caso de la práctica, la araña *books.toscrape* representará el lugar donde se definen las reglas de expresión. En las arañas también se tienen que especificar las solicitudes iniciales para rastrear las URLs y una función de devolución de llamada (*parse*) a la que se llamará para generar los ítems de respuesta de esas solicitudes. Por último, los ítems devueltos por las arañas normalmente se conservan en una base de datos o se escriben en un archivo. En el caso de la práctica, los ítems (título, categoría, descripción, precio y valoración de cada libro)

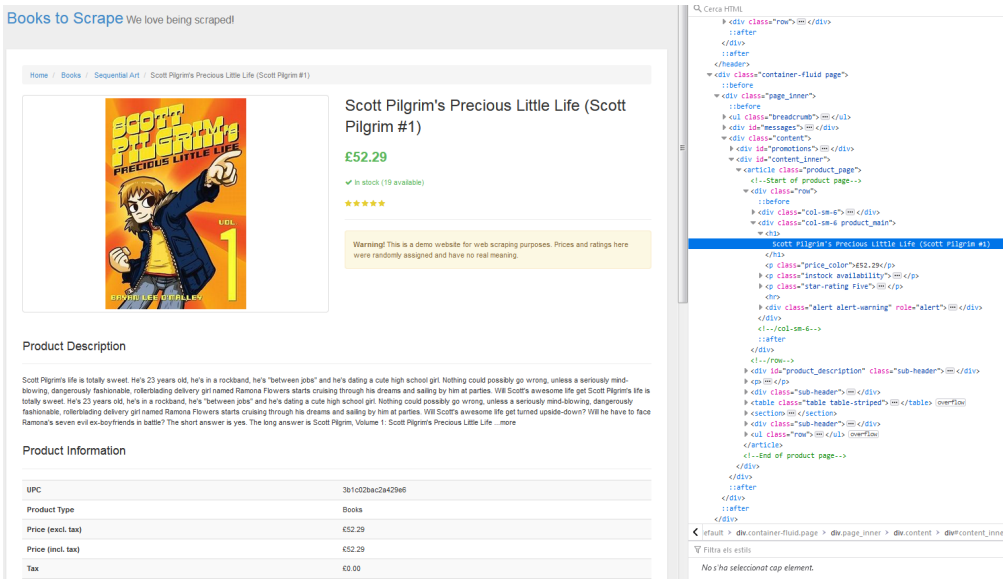


Figura 1: Ejemplo de libro a rastrear.

devueltos por la araña serán guardados en el fichero *books.json*. La araña *books.toscrape* que procesa todas las URLs descubiertas de http://books.toscrape.com, utilizando la función *parse* que a su vez llama a la función *parse_book_page* donde son definidas todas las reglas de expresión de cómo extraer la información deseada, se muestra a continuación:

```
import scrapy

class BooksToscrapeSpider(scrapy.Spider):
  name = 'books.toscrape'
  allowed_domains = ['books.toscrape.com']
  start_urls = ['http://books.toscrape.com/']

  def parse(self, response):
    for book_url in response.css("article.product_pod > h3 > a ::attr(href)").extract():
      yield scrapy.Request(response.urljoin(book_url), callback=self.parse_book_page)
    next_page = response.css("li.next > a ::attr(href)").extract_first()
    if next_page:
      yield scrapy.Request(response.urljoin(next_page), callback=self.parse)

  @staticmethod
  def parse_book_page(response):
    item = {}
    product = response.css("div.product_main")
    item["title"] = product.css("h1 ::text").extract_first()
    item['category'] = response.xpath("//ul[@class='breadcrumb']/li[@class='active']/
    preceding-sibling::li[1]/a/text()").extract_first()
    item['description'] = response.xpath("//div[@id='product_description']/following-sibling
    ::p/text()").extract_first()
    price = response.xpath('//th[text()="Price (incl. tax)"]/following-sibling::td/text()').
    extract_first()
    item['price'] = price.replace('£', '')
    rating = response.xpath('//*[contains(@class, "star-rating")]/@class').extract_first()
    item['rating'] = rating.replace('star-rating ', '')
    yield item
```

En este momento ya podemos iniciar la araña para que recupere la información del catálogo web y la guarde en el fichero *books.json*, aunque primero es recomendable modificar el fichero *books/settings.py* para limitar el acceso de la araña al catálogo web, ya que podemos generar un ataque DDoS. Para ello, debemos descomentar la variable DOWNLOAD_DELAY y darle un valor en segundos (p.ej. DOWNLOAD_DELAY = 3).

Para iniciar la araña se debe ejecutar el siguiente comando:

```
$ cd books
$ scrapy crawl books.toscrape -o books.json
```

3. K-Means

En esta sección se describe como se ha implementado el proceso de agrupamiento en Python, usando la librería scikit-learn [2]. La implementación llevada a cabo se encuentra en el directorio kmeans en [1]. Concretamente, el algoritmo de agrupación elegido para esta práctica ha sido: K-Means. En el caso de la práctica, el algoritmo K-Means agrupará los títulos de los libros del catálogo web, recuperados en la Sección 2, en diferentes clústeres.

3.1. Datos de entrada

Si nos fijamos en la implementación del fichero *kmeans/kmeans.py*, concretamente en el *main*, vemos que empezamos extrayendo la información de los libros del catálogo web, almacenada en el fichero *books/books.json*, y la convertimos en un *DataFrame* (ver Definición 2). A continuación, se eliminan los valores NaN que pudieran existir en *DataFrame* y este es almacenado en un fichero CSV (*kmeans/books.csv*). Para este proceso se usa la librería pandas [3].

Las primeras líneas del *DataFrame* que forman los datos de entrada:

title	category	description	price	rating
Sapiens: A Brief History of Humankind	History	From a renowned historian...	54.23	Five
Sharp Objects	Mystery	WICKED above her hipbone, GIRL...	47.82	Four
Soumission	Fiction	Dans une France assez...	50.10	One
Tipping the Velvet	Historical Fiction	Erotic and absorbing...	53.74	One
A Light in the Attic	Poetry	It's hard to imagine...	51.77	Three

Tabla 1: Conjunto de datos de entrada.

Llegados a este punto, se concreta la estructura de datos que se utilizará para alimentar el algoritmo K-Means. Se crea una lista que contiene los títulos de los libros.

```
titles = df["title"].to_list()
print(titles[:10])  # first 10 titles
>> ['Sapiens: A Brief History of Humankind', 'Sharp Objects', 'Soumission', 'Tipping the
    Velvet', 'A Light in the Attic', "It's Only the Himalayas", 'Libertarianism for
    Beginners', 'Mesaerion: The Best Science Fiction Stories 1800-1849', 'Olio', 'Our Band
    Could Be Your Life: Scenes from the American Indie Underground, 1981-1991']
```

Definición 2. *Un DataFrame es una estructura de datos bidimensional etiquetada que acepta diferentes tipos datos de entrada organizados en columnas. Se puede pensar en un DataFrame como una hoja de cálculo o una tabla SQL.*

3.2. Palabras vacías, stemming y tokenización

Esta sección se centra en definir algunas funciones para transformar el texto de los títulos de los libros para facilitar el entrenamiento del algoritmo K-Means. Para empezar, se obtiene la lista de palabras vacías en inglés

(ver Definición 3), utilizando la librería NLTK [4]. La lista de palabras vacías es en inglés ya que los títulos de los libros están inglés. Después, se obtiene el *Snowball Stemmer*, también utilizando la librería NLTK, para descomponer las palabras que forman cada título en sus raíces correspondientes.

```
# nltk's English stopwords as variable called 'stopwords'
stopwords = nltk.corpus.stopwords.words('english')
# nltk's SnowballStemmer as variabled 'stemmer'
stemmer = SnowballStemmer("english")
print(stopwords[:10])   # first 10 stopwords
>> ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

Definición 3. *Las palabras vacías son palabras sin significado como artículos, pronombres, preposiciones, etc. que son filtradas antes o después del procesamiento de datos en lenguaje natural (texto). Por ejemplo: "a", "the", o "in" que no transmiten un significado significativo.*

A continuación se definen las funciones *tokenize_and_stem* y *tokenize_only*:

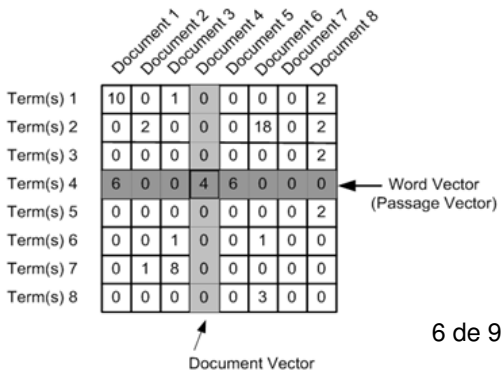
```
def tokenize_and_stem(text):
  # first tokenize by sentence, then by word to ensure that punctuation is caught as it's
    own token
  tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
  filtered_tokens = []
  # filter out any tokens not containing letters (e.g., numeric tokens, raw punctuation)
  for token in tokens:
    if re.search('[a-zA-Z]', token):
      filtered_tokens.append(token)
  stems = [stemmer.stem(t) for t in filtered_tokens]
  return stems

def tokenize_only(text):
  tokens = [word.lower() for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize
    (sent)]
  filtered_tokens = []
  for token in tokens:
    if re.search('[a-zA-Z]', token):
      filtered_tokens.append(token)
  return filtered_tokens
```

Estas funciones son casi iguales, ya que las dos realizan el mismo proceso de tokenización sobre un texto de entrada, pero no son exactamente iguales porqué la función *tokenize_and_stem*, además de realizar la tokenización, extrae las raíces de las palabras que forman parte ese texto de entrada. Las funciones se usarán para calcular la matriz tf-idf en la siguiente sección y para la visualización de los clústeres resultantes del entrenamiento del algoritmo K-Means.

3.3. Extracción de características

En esta sección, se calculará la matriz tf-idf (ver Figura 2). Pero para ello, primero se tienen que calcular las frecuencias de las palabras que contienen los títulos, y el método más popular para hacerlo es el llamado TF-IDF. Este es un acrónimo que significa *Frecuencia de Término – Frecuencia Inversa de Documento* que son los componentes de las puntuaciones resultantes asignadas a cada palabra. Sin entrar en la matemática, TF-IDF son puntuaciones de frecuencia de palabras que tratan de resaltar las palabras que son más interesantes y/o frecuentes. En el caso de la práctica usamos sklearn.feature_extraction.text.TfidfVectorizer.

Figura 2: Ejemplo de matriz tf-idf.



```
tfidf_vectorizer = TfidfVectorizer(stop_words=stopwords, use_idf=True,
  tokenizer=tokenize_and_stem, ngram_range=(1, 3))

# tokenize and build coded vocabulary
tfidf_matrix = tfidf_vectorizer.fit_transform(titles)
print(tfidf_matrix.shape)
>> (998, 7258)
```

La matriz tf-idf está formada por 998 términos y 7258 documentos. También se puede observar el vocabulario (nombrado *terms*) que ha usado *TfidfVectorizer* en la construcción de la matriz:

```
terms = tfidf_vectorizer.get_feature_names()
print(terms[:20])    # first 20 terms
>> ["'d", "'d go", "'d go bernadett", "'m", "'m gone", "'m home", "'m lie", "'m lie tell", "
    's", "'s alic", "'s alic wonderland", "'s astound", "'s astound stori", "'s
    autobiographi", "'s babi", "'s babi ice", "'s berlin", "'s call", "'s call cormoran", "'
    s childhood"]
```

Definición 4. *Un n-grama es una secuencia contigua de n elementos de una muestra determinada de texto o de un discurso. Los elementos pueden ser fonemas, sílabas, letras o palabras según la aplicación. Los n-gramas normalmente se recopilan de un texto.*

Un par de cosas a tener en cuenta sobre los parámetros definidos en la función *TfidfVectorizer*:

- use_idf: habilita la reponderación de frecuencia de documentos inversa.
- ngram_range: define el límite inferior y superior del rango de n-valores para diferentes n-gramas que se extraerán. Ver Definición 4.

3.4. Entrenamiento del algoritmo

Ahora pasamos a la parte divertida. Usando la matriz tf-idf calculada en la sección anterior, se puede ejecutar el algoritmo K-Means utilizando sklearn.cluster.KMeans, para comprender mejor la estructura oculta dentro de los títulos de los libros. El algoritmo K-Means se inicializa con un número predeterminado de clústeres. Elegí el número 40 como número predeterminado de clústeres, ya que el conjunto de datos de entrada contiene libros que pertenecen al menos a más de dos de las 50 categorías existentes en el conjunto de datos, descartando 10 categorías que solo hacen referencia a un único libro.

```
num_clusters = 40
km = KMeans(n_clusters=num_clusters)
km.fit(tfidf_matrix)
clusters = km.labels_.tolist()
```

Para analizar los resultados y visualizar los clústeres (*clusters*), ha sido necesario crear dos nuevos dataframes. Uno que contenga los títulos y sus clústeres asignados (*frame*), y otro que contenga las raíces de los títulos con sus palabras asignadas (*vocab_frame*). Para crear el segundo, además, ha sido necesario crear dos nuevos vocabularios: uno que contenga los títulos tokanizados y otro que contenga las raíces de los títulos tokanizados. Para ello se han usado las funciones: *tokenize_and_stem* y *tokenize_only*, ya nombradas anteriormente (ver Sección 3.2).

```
# new df with titles and clusters
frame = pd.DataFrame({'title': titles, 'cluster': clusters}, index=[clusters],
  columns=['title', 'cluster'])
```

```
# new two vocabularies: stemmed and tokenized
totalvocab_stemmed = []
totalvocab_tokenized = []
for i in titles:
  allwords_stemmed = tokenize_and_stem(i)  # for each item in 'titles', tokenize/stem
  totalvocab_stemmed.extend(allwords_stemmed)  # extend the 'totalvocab_stemmed' list

  allwords_tokenized = tokenize_only(i)
  totalvocab_tokenized.extend(allwords_tokenized)

vocab_frame = pd.DataFrame({'words': totalvocab_tokenized}, index=totalvocab_stemmed)
print('There are ' + str(vocab_frame.shape[0]) + ' items in vocab_frame')
>> There are 6371 items in vocab_frame
```

El beneficio de este procedimiento es que proporciona una forma eficiente de buscar una raíz y devolver la palabra que la contiene muy rápidamente. La desventaja es que hay demasiadas raíces, concretamente 6371. Por ejemplo, la raíz 'run' podría estar asociada con 'ran', 'runs', 'running', etc. Aunque para mi propósito de visualización de los clústeres está bien. A continuación podemos observar las primeras líneas del contenido de los dos nuevos dataframes.

Tabla 2: *frame* y *vocab_frame* dataframes.

title	cluster
Sapiens: A Brief History of Humankind	26
Sharp Objects	0
Soumission	12
Tipping the Velvet	12
A Light in the Attic	9

	words
sapien	sapiens
a	a
brief	brief
histori	history
of	of

Finalmente, vemos la función implementada que muestra los resultados por clúster, identificando las n primeras palabras (elegí n=10) que están más cerca de los centroides de cada uno de ellos, y también vemos el resultado de su ejecución (ver Figura 3).

```
# sort cluster centers by proximity to centroid
order_centroids = km.cluster_centers_.argsort()[:, ::-1]

for i in range(num_clusters):
  print("Cluster {} words:".format(i), end='')
  for ind in order_centroids[i, :10]:  # replace 10 with n words per cluster
    print(' {}'.format(vocab_frame.loc[terms[ind].split(' ')].values.tolist()[0][0]),
      end=',')

  print()
  print()

  print("Cluster {} titles:".format(i), end='')
  for title in frame.loc[i]['title'].values.tolist():
    print(' {},'.format(title), end='')

  print()
  print()
```

Figura 3: Resultados del proceso de *clustering*.

3.5. Evaluación

Bibliografía

[1] Laura Rodríguez-Navas. Recuperación de información y minería de texto. https://github.com/lrodrin/masterAI/tree/master/A14, 2021.

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

[3] Jeff Reback, Wes McKinney, jbrockmendel, Joris Van den Bossche, Tom Augspurger, Phillip Cloud, gfyoung, Sinhrks, Simon Hawkins, Matthew Roeschke, Adam Klein, Terji Petersen, Jeff Tratner, Chang She, William Ayd, Shahar Naveh, Marc Garcia, Jeremy Schendel, Andy Hayden, Daniel Saxton, Vytautas Jancauskas, Ali McMaster, Pietro Battiston, Skipper Seabold, patrick, Kaiqi Dong, chris b1, h vetinari, Stephan Hoyer, and Marco Gorelli. pandas-dev/pandas: Pandas 1.1.5, December 2020.

[4] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. "O'Reilly Media, Inc.", 2009.