Pontificia Universidad Católica del Perú

INF656, Minería Web
Trabajo práctico No. 6
Segundo semestre 2016, Master en Informática

Topic Models

Importante: Los scripts y capturas de pantalla mostrando los resultados de los dos problemas propuestos en este TP, deben ser colocados en el repositorio creado en Paideia al terminar la clase o en su defecto, enviarlo al correo `halatrista@pucp.pe`. Si su respuesta involucra 2 o más archivos, comprímalos en un solo archivo ZIP y envíelo con su nombre. El tiempo estimado para solucionar este trabajo práctico es de 1 hora.

1. Topic Models

El modelado por tópicos (o Topic Models) es una técnica para tratar documentos que no tienen ninguna categorización, y asume que cada documento es una mezcla aleatoria de categorías o tópicos. Un tópico, en el contexto de modelado de tópicos, es una distribución de probabilidades de palabras para un conjunto, e indica la probabilidad que una palabra aparezca en un documento sobre un tópico en particular. En este sentido, el modelado por tópicos tiene como objetivo identificar las relaciones latentes entre documentos pertenecientes a una colección, con el fin de dar una descripción sucinta de ésta sin perder información desde el punto de vista estadístico. En este trabajo práctico, utilizaremos el módulo de Python *scikit-learn* para realizar la tarea extracción de tópicos sobre un conjunto de documentos textuales (corpus).

1.1. Acerca del corpus a analizar

Con respecto al corpus que analizaremos, éste está compuesto de seis documentos asociados a las novelas escritas por *Jane Austin* y por *Charlotte Brontë*. En este trabajo práctico, no utilizaremos las novelas separadas en archivos (seis novelas), sino que han sido divididas en conjuntos (según el número de palabras) para que el análisis de tópicos sea más rápido. Este proceso se realizó gracias a que, como vimos en clases, el análisis se hace por documento y es mejor tener muchos documentos con pocas palabras a analizar que pocos documentos con muchas palabras. Para ello, recupere el corpus que se encuentra en Paideia (*austen-bronte-split*) y cópielo dentro del repertorio *data* donde se encuentran las novelas.

```
import os
CORPUS_PATH = os.path.join('data', 'austen-bronte-split')
filenames = sorted([os.path.join(CORPUS_PATH, fn) for fn in os.listdir(CORPUS_PATH)])
```

En las tres líneas anteriores, solamente definimos una variable con la ruta de nuestros archivos y leemos los documentos que se encuentran en él. Podemos ver que tenemos 813 documentos gracias al comando `print len(filenames)`. Debemos recordar que, la unión de estos 813, son las seis novelas con las cuales hemos trabajado en los trabajos prácticos anteriores.

Posteriormente, construimos una matriz de términos-documentos, como en los trabajos anteriores.

```
import sklearn.feature_extraction.text as text
vectorizer = text.CountVectorizer(input='filename', stop_words='english', min_df=1)
dtm = vectorizer.fit_transform(filenames).toarray()
vocab = np.array(vectorizer.get_feature_names())
```

1.2. Utilización de la técnica NMF

A diferencia del método estudiado en la clase teórica, en este trabajo utilizaremos la técnica llamada *non-negative matrix factorization NMF* la cual se parece mucho al método LDA visto en clase. Entones, utilizaremos NMF para obtener una matriz de documentos relacionados con el tópico (llamados también "componentes") y una lista de las mejores palabras para cada tópico o tema. Para ello, importamos la librería *descomposition* de *scikit-learn*, el cual nos permite extraer modelos.

```
import numpy as np
from sklearn import decomposition
num_topics = 20
num_top_words = 20
clf = decomposition.NMF(n_components=num_topics, random_state=1)
doctopic = clf.fit_transform(dtm)
topic_words = []
for topic in clf.components_:
  word_idx = np.argsort(topic)[::-1][0:num_top_words]
  topic_words.append([vocab[i] for i in word_idx])
doctopic = doctopic / np.sum(doctopic, axis=1, keepdims=True)
```

En las líneas anteriormente descritas, definimos en 20 el número de tópicos a extraer y en 20 el número de palabras por tópico. Luego, creamos una instancia del objeto *decomposition.NMF* con dos parámetros. Luego, utilizamos esta instancia sobre la matriz de términos-documentos *dtm*. Posteriormente, creamos una matriz (*doctopic*) que almacenará los tópicos y ésta será llenada gracias al bucle que se encuentra luego de la creación de este vector. Por otro lado, creamos un vector llamado *topic_words*, donde se almacenarán las palabras representativas por cada tópico y que también será llenada gracias al bucle *for*. Estas palabras se pueden mostrar gracias a la sentencia `print topic_words`.

Problema 1

Describa en detalle, qué hace el bucle mostrado en el código anterior. Muestre algunos resultados intermediarios obtenidos en cada iteración del bucle.

2. Relación entre los tópicos y el corpus (novelas)

Ahora, queremos mostrar las relaciones entre los tópicos (y sus palabras) con los documentos. Entonces, la idea es crear una matriz que represente la pertinencia de cada documento y por cada tópico. Para ello, recuperamos los nombres de cada unos de los documentos (novelas) y los almacenamos en el vector *novel_names*. Luego, calculamos la matriz *doctopic_grouped* que resume (agrupa) las relaciones existentes entre los documentos del corpus y los tópicos. Esto se realia mediante el código siguiente:

```
novel_names = []
for fn in filenames:
  basename = os.path.basename(fn)
  name, ext = os.path.splitext(basename)
  name = name.rstrip('0123456789')
  novel_names.append(name)
novel_names = np.asarray(novel_names)
doctopic_orig = doctopic.copy()
num_groups = len(set(novel_names))
doctopic_grouped = np.zeros((num_groups, num_topics))
for i, name in enumerate(sorted(set(novel_names))):
```

```
  doctopic_grouped[i, :] = np.mean(doctopic[novel_names == name, :], axis=0)
doctopic = doctopic_grouped
```

Ahora, solo nos queda mostrar las palabras para cada tópico, para ello agregamos las siguientes líneas de código:

```
for t in range(len(topic_words)):
  print("Topic {}: {}".format(t, ' '.join(topic_words[t][:15])))
```

En el código anterior, lo único que hacemos es "barrer" el vector *topic_words*, darle un formato adecuado y mostrar las 15 primeras palabras.

También, podemos mostrar los documentos y cuáles son los tópicos que los representan de mejor manera. Este procedimiento no difiere esencialmente del procedimiento de identificación de las palabras más frecuentes en cada documento. Para ello, agregamos las líneas de código siguientes:

```
novels = sorted(set(novel_names))
for i in range(len(doctopic)):
  top_topics = np.argsort(doctopic[i,:])[::-1][0:3]
  top_topics_str = ' '.join(str(t) for t in top_topics)
  print("{}: {}".format(novels[i], top_topics_str))
```

–
Hugo Alatrista-Salas
PUCP
Basado en `https://de.dariah.eu/tatom/index.html`