Pontificia Universidad Católica del Perú

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

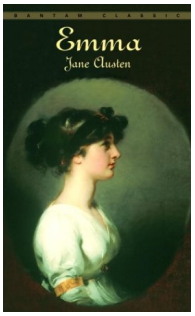
Term Frequency

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. 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 2.5 horas.

1. Calcular la frecuencia de términos en un documento

En este trabajo práctico, construiremos un contador de palabras "artesanal" que nos permita contar el número de ocurrencias de un término dentro de un documento. Para ello, necesitaremos haber terminado los anteriores trabajos prácticos, ya que, re-utilizaremos parte del código implementado anteriormente.

1.1. Acerca del documento a analizar



Para realizar nuestros experimentos en este trabajo práctico, utilizaremos una novela de la famosa escritora inglesa *Jane Austin*. *Emma*, es una novela sobre la arrogancia juvenil y los peligros del mal interpretado romance. La novela fue publicada por primera vez en diciembre de 1815.

Como hasta ahora, en este TP, utilizaremos, Python y su extensión NLTK para el análisis de datos textuales. Primeramente, como ya lo hicimos anteriormente, utilizaremos como delimitador de palabras (tokens) los espacios en blanco que existen en el documento. Para ello creamos un script que lo llamaremos "frecuency.py", que luego podrá ser ejecutado con `python frecuency.py Austen_Emma.txt` utilizando la línea de comandos de Linux. Como podemos notar, el script debe leer un archivo *Austen_Emma.txt* que es, justamente, el documento que describimos líneas arriba.

```
from nltk.tokenize import RegexpTokenizer
import re
import sys
f = open(sys.argv[1])
text = f.read()
tokenizer = RegexpTokenizer(r'\w+')
tokens = tokenizer.tokenize(text)
print tokens
```

Como podemos ver en el resultado de nuestro primer script, existe una gran cantidad de términos. Para trabajar con una cantidad menor, seleccionaremos solamente los que comienzan con la letra *h* (ignorando mayúsculas/minúsculas). Para ello, necesitamos reemplazar las tres líneas anteriores por las siguientes:

```
tokenizer = RegexpTokenizer('^[hH]\w+')
tokens = tokenizer.tokenize(text)
print tokens
```

- ¿Cuántos términos habían en el documento antes de filtrar los términos? ¿Cuántos términos existen luego de filtrarlos utilizando la letra h?
- ¿Cree que todos los términos son interesantes a analizar? ¿Cuáles cree que no son interesantes?

2. Creación de un conjunto de *stop words*

Pese a que ya existen herramientas que permiten eliminar algunos caracteres especiales en un documento (o en un corpus completo), existen algunas palabras que deberían formar parte de los *stop words*. Si deseamos crear un conjunto de palabras que sean ignoradas antes de realizar algún tratamiento sobre el documento, antes debemos pensar en una representación de las palabras de manera a minimizar el número de comparaciones entre las palabras *no deseadas* y las palabras del documento.

Una de las tantas posibles soluciones es, normalizar el documento homogeneizando las palabras en función al uso de mayúsculas y minúsculas. Para ello, creamos un vector que lo llamaremos *tokens_low* donde guardaremos las palabras que empiezan con h y con H, pero convertidas a minúsculas. Posteriormente, llenaremos con el vector con las palabras almacenadas en nuestro vector *tokens* pero en minúsculas.

```
tokens_low = []
for word in tokens:
  tokens_low.append(word.lower())
```

- Este proceso NO es equivalente a utilizar la función $RegexpTokenizer('[hH]\w+')$. ¿Por qué? Explique con un ejemplo.

Volviendo a los *stop words*, creamos un *vector* de palabras, las cuales nos servirá para filtrar nuestro grupo de *tokens* mostrados en la etapa anterior

```
stop_words = ['he', 'his']
```

Ahora, mostraremos los términos de nuestro documento que comienzan con la letra H y que no están incluidas en la lista de *stop words*. Para ello, utilizaremos un bucle *for*.

```
for word in tokens_low:
  if word not in stop_words:
    print word
```

Problema 1

Complete la lista de *stop words* de manera a tener un conjunto más completo de palabras que usted crea conveniente eliminarlas.

3. Cálculo de la medida *term frequency*

Para calcular la medida *term frequency* para cada uno de los términos de nuestro documento, utilizaremos la noción de *diccionarios* en Python. Un diccionario es una colección de pares *clave-valor*. A diferencia de una lista, que es una colección de objetos indexada por números enteros consecutivos, un *diccionario* permite como clave cualquier tipo de datos inmutable, y los valores pueden ser totalmente arbitrarios. Los diccionarios tienen una sintaxis especial en Python usando las llaves {}.

Entonces, primero creamos un diccionario vacío utilizando la línea `count = {}`. Posteriormente, recorremos nuestro vector conteniendo las palabras en minúsculas (*tokens_low*) y por cada token, verificar si él está incluido en nuestro diccionario. Para ello, utilizamos el operador *in* que comprueba si un término es una clave del diccionario. Si es así, le sumamos una unidad (ocurrencia), sino, lo agregamos y le asignamos una ocurrencia de 1. El siguiente código permite realizar estas operaciones.

```
count = {}
for word in tokens_low:
  if word in count:
    count[word] += 1
  else:
    count[word] = 1
```

Podemos fácilmente ver ambos componentes del diccionario utilizando las líneas `print count.keys()` y `print count.values()`.

Problema 2

Como podemos ver en los resultados finales de este TP, aún tenemos los problemas de palabras repetidas. El motivo por el cual es que, seguimos utilizando el vector *tokens_low* en el proceso de cálculo de las frecuencias. Se pide al alumno, crear un nuevo script en Python y que integre todas las cosas aprendidas hasta este momento de manera a calcular la medida del *term frequency* del documento *Austen_Emma.txt*, utilizado todas las palabras del documento (utilice una la lista de *stop words* con palabras que cree conveniente omitirlas).

4. Midiendo la polaridad del texto

En esta última parte del trabajo práctico, aprenderemos a medir qué tan positivo o negativo puede ser un texto en función de su contenido. Para ello, trabajaremos con dos diccionarios provistos por *Robert A. Stine* de la Universidad de Pensilvania. Estos dos diccionarios contienen palabras *positivas* y *negativas*. La idea, es construir dos diccionarios a partir de estos archivos y comparar si una palabra del documento pertenece a uno de estos dos diccionarios para "aumentar" el peso "positivo" o "negativo" del documento analizado (la novela "Emma").

Problema 3

En esta parte, se pide al alumno, mejorar el script desarrollado anteriormente de manera a calcular la polaridad del texto estudiado. Para ello, se sugiere utilizar una medida diferente a la frecuencia, como por ejemplo, el logaritmo de la frecuencia, para evitar valores desproporcionados. ¿Cómo determinar la neutralidad del texto? Provea una solución algorítmica (empíricamente) para verificar si un término es positivo, negativo o neutro, basado en el contenido del documento. Utilice todos los métodos aprendidos hasta el momento (*steaming*, lematización, eliminación de *stop-words*, etc.)

–
Hugo Alatrista-Salas
PUCP