Pontificia Universidad Católica del Perú

INF656, Minería Web
Trabajo práctico No. 7
Segundo semestre 2016, Master en Informática

Representación vectorial

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 2.5 horas.

1. Introducción

El uso de la frecuencia de palabras es bastante útil cuando analizamos datos textuales. Representar un documento o un conjunto de ellos, vectorialmente, gracias a una lista de frecuencias (un vector en un *bag-of-words*), nos permite el uso de una amplia gama de herramientas matemáticas desarrolladas para estudiar y manipular vectores. En este trabajo, nos enfocamos justamente en la tarea de construcción de un *bag-of-words* en Python utilizando la extensión *scikit-learn* y en las consultas que podemos hacer sobre ella.

1.1. Acerca de la extensión *Scikit-learn*

En este trabajo práctico, dejaremos de utilizar la extensión *NLTK* de Python. Esta vez, utilizaremos otra extension llamada *scikit-learn* (`http://scikit-learn.org/stable/index.html`), que es un módulo de Python creado para realizar una gran cantidad de tareas de *machine learning*. Esta herramienta ha sido construida en base a la extensión *SciPy* y es distribuida bajo licencia BSD.

El proyecto se inició en 2007 por *David Cournapeau* como un proyecto de *Google Summer School* y desde entonces muchos voluntarios han contribuido. En la actualidad se mantiene por un equipo de voluntarios y ha sido probado en Python 2.6, Python 2.7, y Python 3.4.

Para realizar nuestros experimentos en este trabajo práctico, primeramente, creamos un script que lo llamaremos "bagOfWords.py", que luego podrá ser ejecutado con `python bagOfWords.py` desde la línea de comandos de Linux. Esta vez, no leeremos archivos como en los trabajos prácticos anteriores, sino que los recuperaremos desde el mismo código. El corpus que analizaremos esta compuesto de seis documentos asociados a las novelas escritas por *Jane Austin* y por *Charlotte Brontë*. Podemos empezar nuestro script con las líneas siguientes:

```
from sklearn.feature_extraction.text import CountVectorizer

filenames = ['data/Austen_Emma.txt',
'data/Austen_Pride.txt',
'data/Austen_Sense.txt',
'data/CBronte_Jane.txt',
'data/CBronte_Professor.txt',
'data/CBronte_Villette.txt']
```

```
vectorizer = CountVectorizer(input='filenames')
```

La primera línea permite importar la clase CountVectorizer. Esta clase es personalizable y nos permite crear una matriz de términos-documentos. Por ejemplo, podemos utilizar una lista de palabras "vacías" con el parámetro *stop_words*. Otros posibles parámetros (lista no exhaustiva) son:

- *lowercase* (predeterminado True), convierte todo el texto a minúsculas antes de segmentarlas
- *min_df* (por defecto 1), elimina los términos del vocabulario que se producen en menos de *min_df* documentos
- *vocabulary*, ignora las palabras que no aparecen en esta lista
- *strip_accents*, eliminar acentos
- *token_pattern*, permite utilizar expresiones regulares para ignorar palabras
- *tokenizer* (por defecto desactivada), utiliza una función personalizada para tokenizing

Posteriormente, en la segunda línea, creamos un vector llamado *filenames*, el representa nuestro conjunto de documentos. Estos documentos debe estar almacenados en una carpeta llamada *data*. Finalmente, gracias a la línea `vectorizer = CountVectorizer(input='filenames')` creamos una instancia del objeto *CountVectorizer*, a partir del cual, podremos crear una matriz de términos-documentos. Esta instancia ha sido creada utilizado un solo parámetro, que es la lista de documentos *filenames*. Para ver qué otros parámetros existen y cuál es su valor actual, puede utilizar la línea `print(vectorizer)`.

Como se vio en el curso teórico, algorítmicamente, la forma más intuitiva de construir un *bag-of-words* es mediante dos pasos simples:

1. Asignar un identificador fijo entero a cada palabra w que ocurre en el corpus, por ejemplo, mediante la construcción de un diccionario de las palabras representados por índices enteros.
2. Para cada documento d, contar el número de apariciones de cada palabra w y almacenarla en una lista $dtm[d, i]$ donde i es el índice de la palabra w en el diccionario previamente construido.

También se vio en el curso teórico que, la mayoría de los valores de esta matriz $dtm[d, i]$ serán ceros. Por esta razón se dice que los *bag-of-words* suelen ser un conjunto de datos dispersos de alta dimensión. Podemos ahorrar una gran cantidad de memoria almacenando solo los valores distintos de cero de los vectores de características. Las matrices *scipy.sparse* son estructuras de datos que permiten almacenar justamente este tipo de matrices "ralas" y la extensión *scikit-learn* las utiliza.

La instancia *CountVectorizer* creada anteriormente, nos brinda la matriz de términos-documentos y un vocabulario de términos, ambos almacenados en una sola lista. Convenientemente, separaremos esta lista en sus dos componentes mediante una variable *dtm* (document-term matrix) en la cual vamos a almacenar la matriz *scipy.sparse* construida a partir de nuestro corpus y mediante la variable *vocab* que guardará el vocabulario. Esto se puede realizar mediante las líneas de código siguientes.

```
dtm = vectorizer.fit_transform(filenames)
vocab = vectorizer.get_feature_names()
```

Problema 1

Responda las siguientes preguntas.

- Modifique la línea que instancia la clase *CountVectorizer* de la siguiente manera: `vectorizer = CountVectorizer(input='filenames', min_df = 1, stop_words = 'english')`. Explique, qué cambios produce esta modificación sobre sus resultados. ¿Son éstos benéficos? Sugerencia: utilice `len(vocab)` para ver los cambios.

- Utilice el comando `print dtm` para ver el contenido de la matriz *dtm*. ¿Cómo está organizada la información? Explique cada uno de los campos en esta estructura de datos.

Antes de que podamos consultar la matriz y saber, por ejemplo, cuántas veces aparece la palabra "house" en *Emma* (el primer documento), tenemos que convertir esta matriz actual (obtenida por *CountVectorizer*) en una matriz dispersa que puede ser utilizada por *NumPy*. También vamos a convertir la lista que almacena nuestro vocabulario, en una matriz de *NumPy*, como una matriz compatible con una mayor variedad de operaciones realizables sobre ellas.

Para ello, agregamos las líneas de código siguientes a nuestro script.

```
import numpy as np # creamos un alias np para numpy
dtm = dtm.toarray()
vocab = np.array(vocab)
```

2. Consultas sobre la *dtm*

Con el trabajo de preparación previo, realizar consultas sobre la *dtm* es simple. Por ejemplo, las siguientes mostradas más abajo demuestran dos maneras de encontrar cuántas veces aparece la palabra "casa" en el primer texto, (la novela *Emma*).

```
# primera forma
house_idx = list(vocab).index('house')
print dtm[0, house_idx]
# segunda forma
print dtm[0, vocab == 'house']
```

NOTA: Si deseamos verificar, en qué posición del vector que almacena nuestro corpus está la obra *Emma*, podemos escribir la siguiente línea en la línea de comandos de Python `print filenames[0] == 'data/Austen_Emma.txt'` (la respuesta debe ser *true*).

En las lineas agregadas anteriormente, la variable *dtm* es utilizada (técnicamente) como una matriz *NumPy*. Debemos tener en cuenta que, las matrices *NumPy* soportan algunas operaciones matriciales como el producto escalar.

Problema 2

Construya una tabla con las ocurrencias de las palabras siguientes: "and", "emma", "house", "pride" y "of". Esta lista debe mostrar las ocurrencias por cada documento.

3. Comparación de dos documentos

"Arreglar" nuestros textos en una matriz de términos-documentos nos permite utilizar una serie de operaciones exploratorias sobre los datos contenidos en esta matriz. Por ejemplo, podemos calcular la semejanza entre los documentos (novelas). Dado a que cada fila de la matriz de términos-documentos es una secuencia de frecuencias de palabras que representan una novela, es posible aplicar algunas nociones matemáticas, de manera a calcular la semejanza entre dos novelas. Como vimos en los cursos teóricos, para medir cuán separados están dos vectores, podemos utilizar la distancia Euclidiana (aunque ésta no sea la mejor medida). La distancia Euclidiana entre dos vectores en el plano es la longitud de la hipotenusa que une los dos vectores. El concepto de distancia no se limita a dos dimensiones, i.e., ella se extiende a un número arbitrario de dimensiones.

A partir de dos novelas de nuestro corpus, expresados mediante dos vectores, podemos calcular la distancia Euclidiana entre ellas. Para ello, utilizamos la función *euclidean_distances* de la extensión *scikit-learn* de Python. Entonces, agregamos las siguientes líneas de código a nuestro script.

```
dist = euclidean_distances(dtm)
print np.round(dist, 1)
```

Si queremos conocer la distancia entre dos documentos, por ejemplo las novelas *Pride and Prejudice* y *Jane Eyre*, solo bastará con lanzar el comando `print dist[1, 3]`. En el comando anterior, *dist* permite calcular la distancia Euclidiana entre los documentos con índices 1 (*Pride and Prejudice*) y 3 (*Jane Eyre*).

También, podemos ver si la distancia calculada anteriormente es más grande a la distancia entre las novelas *Jane Eyre* y *Villette*. Esto se puede corroborar gracias a la sentencia `print dist[1, 3] > dist[3, 5]`.

Problema 3

- ¿Cuál es el resultado de la última sentencia agregada a nuestro script? ¿Cuál es la interpretación que puede darle?
- ¿Qué hace la función `np.round(dist, 1)`?
- Acerca de la tabla de distancias entre los 5 documentos de nuestro corpus. ¿Qué puede comentar de ella? ¿Qué representa y cómo se calcula cada valor de las celdas?

4. Calculando el ángulo entre dos vectores

Como habíamos comentado en clase, la distancia Euclidiana no es totalmente adecuada para medir la semejanza entre dos documentos. Para solucionar esto, podemos calcular la semejanza calculando el coseno del ángulo θ formado por los vectores que representan los documentos. En Python, esto se logra importando el módulo *sklearn.metrics.pairwise.cosine_similarity* y utilizándolo en lugar de la extensión *euclidean_distances*.

No debemos olvidar que el coseno es una medida de semejanza (w.r.t., distancia) que oscila entre 0 y 1. Con el fin de obtener una medida de la distancia (o diferencia), tenemos que "voltear" la medida de manera que, a mayor ángulo mostremos un mayor valor. Entonces, proponemos lo siguiente: la distancia será calculada mediante la unidad (1) menos el coseno del ángulo formado por los dos vectores, i.e., $1 - cos(\theta)$.

```
from sklearn.metrics.pairwise import cosine_similarity
dist = 1 - cosine_similarity(dtm)
print np.round(dist, 2)
```

Esta matriz de distancias, funciona igual que la anterior, es decir, si queremos conocer la distancia entre dos documentos, por ejemplo las novelas *Pride and Prejudice* y *Jane Eyre*, solo bastará con lanzar el comando `print dist[1, 3]`.

Problema 4

- Acerca de la tabla de distancias (coseno) entre los 5 documentos de nuestro corpus. ¿Qué puede comentar de ella? ¿Las distancias están en proporción a las calculadas en la tabla anterior (Problema 3)? Comente.

5. Visualización de los vectores

A menudo, es necesario visualizar las distancias por pares de documentos en nuestro corpus. Un enfoque general para visualizar las distancias, es asignar un punto en un plano para cada vector, teniendo en cuenta de que la distancia entre puntos sea proporcional a las distancias calculadas en los ejercicios anteriores. Este tipo de visualización es llamada, *multidimensional scaling* (MDS) y pertenece a la familia de funciones comunes de *scikit-learn*. Para realizar esta tarea, también necesitaremos la biblioteca *matplotlib* de python. Entonces, necesitamos importar las siguientes extensiones.

```
import os
import matplotlib.pyplot as plt
from sklearn.manifold import MDS
```

Una vez importadas las extensiones necesarias, utilizamos la función *multidimensional scaling* para crear nuestra visualización.

```
mds = MDS(n_components=2, dissimilarity="precomputed", random_state=1)
pos = mds.fit_transform(dist)
```

En la primera línea, instanciamos el objeto MSD con 3 parámetros: *n_components* = 2 ya que visualizaremos solamente 2 dimensiones, *dissimilarity* = "*precomputed*", porque nuestra matriz de semejanza ya está calculada; y, *random_state* = 1 para indicar que nuestra visualización es reproductible. En la segunda línea, calculamos las coordenadas de los puntos en función al número de componentes y al número de muestras. Cada uno de los puntos representa un documento.

Finalmente, solo nos queda, mostrar nuestro objeto con la leyenda que queremos y en un formato que sea fácilmente entendible. Para ello, agregamos las siguientes líneas que permiten, entre otros, agregar un nombre a los puntos mostrados y cambiarle los colores según la autora de la novela.

```
xs, ys = pos[:, 0], pos[:, 1]
names = [os.path.basename(fn).replace('.txt', '') for fn in filenames]
for x, y, name in zip(xs, ys, names):
  color = 'orange' if "Austen" in name else 'skyblue'
  plt.scatter(x, y, c=color)
  plt.text(x, y, name)
plt.axhline(0, color='black') #agregar linea eje X
plt.axvline(0, color='black') #agregar linea eje Y
```

Solo nos queda visualizar nuestra imagen con el comando `plt.show()`.

Si deseamos visualizar nuestros documento en 3D, solo debemos hacer algunos cambios en nuestro script. Estos cambios son:

```
mds = MDS(n_components=3, dissimilarity="precomputed", random_state=1)
pos = mds.fit_transform(dist)

from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(pos[:, 0], pos[:, 1], pos[:, 2])
for x, y, z, s in zip(pos[:, 0], pos[:, 1], pos[:, 2], names):
  ax.text(x, y, z, s)
plt.show()
```

Problema 5

Agregue las líneas de los exes X, Y y Z en la figura 3D obtenida anteriormente.

–
Hugo Alatrista-Salas
PUCP