Minería de textos

Máster Online en Ciencia de Datos

UCO ONLINE

Dr. José Raúl Romero

Profesor Titular de la Universidad de Córdoba y Doctor en Ingeniería Informática por la Universidad de Málaga. Sus líneas actuales de trabajo se centran en la democratización de la ciencia de datos (*Automated ML* y *Explainable Artificial Intelligence*), aprendizaje automático evolutivo y analítica de software (aplicación de aprendizaje y optimización a la mejora del proceso de desarrollo de software).

Miembro del Consejo de Administración de la *European Association for Data Science*, e investigador senior del Instituto de Investigación Andaluz de *Data Science and Computational Intelligence*.

Director del **Máster Online en Ciencia de Datos** de la Universidad de Córdoba.

Detección de términos en minería de datos

UCO ONLINE

Introducción

- **nltk** es un paquete para el desarrollo de programas en Python que trabajen con datos en formato texto
- Permite realizar un gran número de tareas como tokenización, lematización, stemming, o etiquetado, entre muchas otras
- Está disponible para Windows, Mac OS X y Linux
- Es un proyecto gratuito, de código abierto e impulsado por la comunidad
- Al igual que Scrapy, nltk requiere de Python y una serie de librerías para poder funcionar

Instalación de nltk

- Dependiendo de si usamos conda como gestor paquetes o no, utilizaremos uno de los siguientes comandos para instalar nltk:

```
conda install -c conda-forge nltk
```

```
pip install nltk
```

- Además, es necesario instalar el corpus y los modelos que serán utilizados por varias funciones de nltk. Existen dos opciones:

```
python -m nltk.downloader popular
```

```
import nltk; nltk.download('popular')
```

⚠ *En una shell de Python*

Tokenización de frases

- La función por defecto y recomendada por nltk para la tokenización de frases es `sent_tokenize`. Internamente utiliza la clase `PunktSentenceTokenizer`, la cual es apta para tokenizar textos en varios idiomas, entre ellos el español

```
1   from nltk.tokenize import sent_tokenize
2
3   text = "Hola a todos. Bienvenidos al Máster en Ciencia de Datos. Estamos estudiando Minería de Textos"
4   sentences = sent_tokenize(text)
5   print(">>>>>>>>", sentences)
>>>>>>>> ['Hola a todos.', 'Bienvenidos al Máster en Ciencia de Datos.', 'Estamos estudiando Minería de Textos']
```

- Aunque no suele ser necesario, se pueden utilizar modelos especializados para otros idiomas como, por ejemplo, el español

```
5   es_tokenizer = nltk.data.load(resource_url='tokenizers/punkt/spanish.pickle')
6   sentences = es_tokenizer.tokenize(text)
```

Idiomas disponibles en: https://www.nltk.org/nltk_data/

Tokenización de palabras

- La función por defecto y recomendada por nltk para la tokenización de palabras es `word_tokenize`. Esta función utiliza la clase `TreebankWordTokenizer`, la cual define varias expresiones regulares para realizar la tokenización

```
1   from nltk.tokenize import word_tokenize
2
3   words = word_tokenize("Bienvenidos al Máster en Ciencia de Datos")
4   print(">>>>>>>>", words)
```

```
>>>>>>>> ['Bienvenidos', 'al', 'Máster', 'en', 'Ciencia', 'de', 'Datos']
```

También podríamos emplear `RegexpTokenizer`, la cual nos permite definir nuestras propias expresiones regulares

- La tokenización de palabras se debe hacer tras realizar la de frases

```
1   from nltk.tokenize import sent_tokenize, word_tokenize
2
3   text = "Hola a todos. Bienvenidos al Máster en Ciencia de Datos. Estamos estudiando Minería de Textos"
4   words = [word_tokenize(sentence) for sentence in sent_tokenize(text)]
5   print(">>>>>>>>", words)
```

```
>>>>>>>> [['Hola', 'a', 'todos', '.'], ['Bienvenidos', 'al', 'Máster', 'en', 'Ciencia', 'de', 'Datos', '.'],
['Estamos', 'estudiando', 'Minería', 'de', 'Textos']]
```

Corpus de texto

- nltk dispone de varias colecciones de textos. Cabe destacar los libros del Proyecto Gutenberg, un repositorio público de libros libres y/o sin derechos de copyright en vigor
- Podemos usar la función `fileids()` para devolver el listado de libros disponibles en nltk

```
1   from nltk.corpus import gutenberg
2
3   print(">>>>>>", gutenberg.fileids(), "\n")
4
5   alice = gutenberg.raw("carroll-alice.txt")
6   print(alice[:200]) #imprimimos los 200 primeros caracteres
```

```
>>>>>> ['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-kjv.txt', 'blake-poems.txt', 'bryant-
stories.txt', 'burgess-busterbrown.txt', 'carroll-alice.txt', 'chesterton-ball.txt', 'chesterton-brown.txt', 'cheste
rton-thursday.txt', 'edgeworth-parents.txt', 'melville-moby_dick.txt', 'milton-paradise.txt', 'shakespeare-caesar.tx
t', 'shakespeare-hamlet.txt', 'shakespeare-macbeth.txt', 'whitman-leaves.txt']

[Alice's Adventures in Wonderland by Lewis Carroll 1865]

CHAPTER I. Down the Rabbit-Hole

Alice was beginning to get very tired of sitting by her sister on the
bank, and of having nothing to do: once
```

Además del método `raw`, podemos utilizar los métodos `words` y `sents`, los cuales devuelven el texto ya tokenizado por palabras o frases, respectivamente

TF-IDF

- scikit-learn, mediante la clase `TfidVectorizer`, permite calcular el valor de TF-IDF
- Es necesario instalar los paquetes **scikit-learn y pandas**:

```
conda install -c conda-forge scikit-learn pandas
```

```
pip install scikit-learn pandas
```

⚠ *Aunque cumple el mismo objetivo, obsérvese que en scikit-learn se calcula de forma diferente a la formulación original del método.*

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer


corpus = [
    'This is a book on data mining',
    'This book describes data mining and text mining using RapidMiner',
]

vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(corpus)
feature_names = vectorizer.get_feature_names_out()
dense = vectors.todense()
denselist = dense.tolist()
df = pd.DataFrame(denselist, columns=feature_names)
print(df)
```

	and	book	data	describes	is	mining	on	rapidminer	text	this	using
0	0.000000	0.354649	0.354649	0.000000	0.498446	0.354649	0.498446	0.000000	0.000000	0.354649	0.000000
1	0.342119	0.243420	0.243420	0.342119	0.000000	0.486841	0.000000	0.342119	0.342119	0.243420	0.342119

Filtrado de palabras vacías

- nltk dispone de una lista de palabras vacías para varios idiomas

```python
import nltk

stop_words = nltk.corpus.stopwords.words('spanish')
print(stop_words[:10]) # mostramos las 10 primeras
```

```
['de', 'la', 'que', 'el', 'en', 'y', 'a', 'los', 'del', 'se']
```

Podemos utilizar la función `nltk.corpus.stopwords.fileids()` *para consultar los idiomas para los que hay disponibles una lista de palabras vacías*

- A partir de dicha lista borraremos las palabras vacías de los tokens que obtuvimos en el paso anterior

¡Cuidado al usar listas predefinidas! La lista anterior incluye "no" que es muy útil en análisis de sentimientos

```python
import nltk

stop_words = nltk.corpus.stopwords.words('spanish')
words = ["Bienvenidos", "al", "Máster", "en", "Ciencia", "de", "Datos"]
filtered_tokens = [word for word in words if word not in stop_words]

print(">>>>>>>", filtered_tokens)
```

```
>>>>>>> ['Bienvenidos', 'Máster', 'Ciencia', 'Datos']
```

Stemming

- nltk implementa varios métodos de stemming y uno de lematización en el modulo `nltk.stem`. El método de stemming más utilizado es `PorterStemmer`, mientras que el método de lematización disponible es `WordNetLemmatizer`

```
1   from nltk.stem import PorterStemmer
2
3   stemmer = PorterStemmer()
4   print(">>>>>>", stemmer.stem("accuracies"))
>>>>>> accuraci
```

```
1   from nltk.stem import WordNetLemmatizer
2
3   lemmatizer = WordNetLemmatizer()
4   print(">>>>>>", lemmatizer.lemmatize("accuracies"))
>>>>>> accuracy
```

- Para textos en español, u otros idiomas, es más apropiado usar `SnowballStemmer` para realizar el proceso de stemming.

```
1   from nltk.stem import SnowballStemmer
2
3   stemmer = SnowballStemmer("spanish")
4   print(">>>>>", stemmer.stem("Estuviste"))
>>>>> estuv
```

⚠ *nltk no dispone de métodos para lematización en español*

Lematización en español con spacy

- Para realizar la lematización en español se puede utilizar **spacy**, el cual puede ser instalado con uno de los siguientes comandos para instalar nltk:

```
conda install -c conda-forge spacy
```

```
pip install spacy
```

- Además, es necesario instalar el componente que utilizaremos para realizar la lematización en español, es este caso `es_core_news_sm`:

```
python -m spacy download es_core_news_sm
```

```
import spacy

lemmatizer = spacy.load('es_core_news_sm')
doc = lemmatizer("televisores")

print(">>>>>>>", doc[0].lemma_)
```

```
>>>>>>> televisor
```

N-gramas

- La función `ngrams` de nltk implementa el método de la **ventana deslizante** para obtener los n-gramas de tamaño arbitrario
  - Nota: El texto debe haber sido tokenizado previamente

```python
from nltk import ngrams
from nltk.tokenize import word_tokenize

sentence = "Bienvenidos al Máster en Ciencia de Datos"
words = word_tokenize(sentence)
bigrams = ngrams(words, 2)

for grams in bigrams:
    print(grams)
```

```
('Bienvenidos', 'al')
('al', 'Máster')
('Máster', 'en')
('en', 'Ciencia')
('Ciencia', 'de')
('de', 'Datos')
```

El segundo parámetro de `ngrams` *es el tamaño de los n-gramas*

TF-IDF con n-gramas

- La clase `TfidfVectorizer` de scikit-learn puede ser inicializada con el parámetro `ngram_range` para que calcule el TF-IDF de los n-gramas
- Por ejemplo, si fijamos el valor de dicho parámetro a `(2,2)` estaremos indicando que nos interesan los n-gramas que tienen como mínimo y máximo dos palabras, es decir, solo se calculará la importancia de los bigramas

```
5  corpus = [
6      'This is a book on data mining',
7      'This book describes data mining and text mining using RapidMiner',
8  ]
```

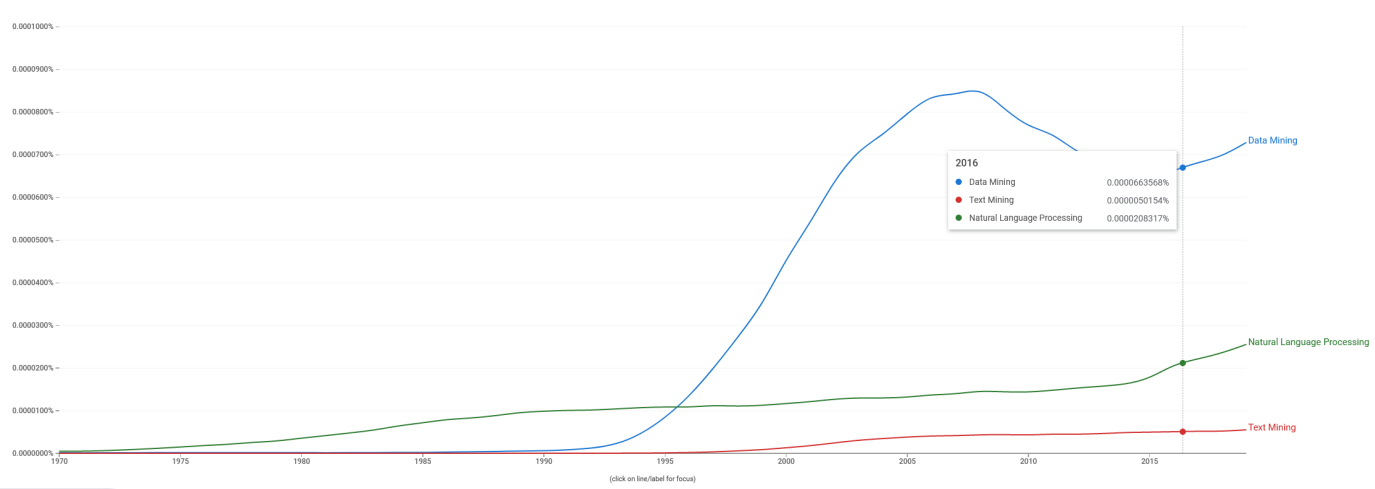
```
vectorizer = TfidfVectorizer(ngram_range=(2,2))
```

	and text	book describes	book on	data mining	describes data	is book	mining and	mining using	on data	text mining	this book	this is	using rapidminer
0	0.000000	0.000000	0.471078	0.335176	0.000000	0.471078	0.000000	0.000000	0.471078	0.000000	0.000000	0.471078	0.000000
1	0.342871	0.342871	0.000000	0.243956	0.342871	0.000000	0.342871	0.342871	0.000000	0.342871	0.342871	0.000000	0.342871

La ventana deslizante crea un bigrama por cada par de palabras consecutivas del corpus de documentos y calcula el tf-idf como si se tratasen de una única palabra

Ngram Viewer de Google

- Google dispone de una herramienta para calcular la frecuencia de aparición de n-gramas en el corpus de libros de Google



https://books.google.com/ngrams/

¡Gracias!

UCO ONLINE