Hoja informativa: Vectorización de textos

Práctica

```
# NLTK: obtener la lista de palabras lematizadas de un texto

# es posible que al principio tengas que descargar los archivos de wordnet
# importar nltk
# nltk.download('wordnet') # https://wordnet.princeton.edu/

from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

lemmatizer  = WordNetLemmatizer()

text = "All models are wrong, but some are useful.".

tokens = word_tokenize(text.lower())
lemmas = [lemmatizer.lemmatize(token) for token in tokens]

# mostrar como una sola línea
print(" ".join(lemmas))
```

```
# spaCy: obtener la lista de palabras lematizadas de un texto

# es posible que al principio tengas que descargar un modelo spaCy con
# python -m spacy descargar en

import spacy

nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])

doc = nlp(text.lower())

lemmas = [token.lemma_ for token in doc]

print(" ".join(lemmas))
```

```
# patrón: comprueba la sintaxis en https://docs.python.org/3.7/library/re.html
# sustitución: con qué debe sustituirse cada coincidencia de patrón
# texto: el texto que la función escanea en busca de coincidencias con el patrón

import re
re.sub(pattern, replacement, text)
```

```
# obtener palabras vacías para el español

from nltk.corpus import stopwords

stop_words = set(stopwords.words('spanish'))
```

```
# construir una bolsa de palabras, un modelo simple para vectorizar textos
# stopwords: lista de palabras vacías

from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer

stop_words = set(stopwords.words('spanish'))
count_vect = CountVectorizer(stop_words=stop_words)

bow = count_vect.fit_transform(corpus)

# diccionario de palabras únicas
words = count_vect.get_feature_names()

# mostrar una matriz de bolsa de palabras
print(bow.toarray())
```

```
# construir una bolsa de palabras de n-gramas (sin palabras vacías)
# min_n: valor mínimo de n
# max_n - valor máximo de n

from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer(ngram_range=(min_n, max_n))

bow = count_vect.fit_transform(corpus)

# diccionario de n-gramas
words = count_vect.get_feature_names()

# mostrar una matriz de bolsa de palabras
print(bow.toarray())
```

```
# construir la TF-IDF para un corpus

from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

stop_words = set(stopwords.words('spanish'))

tfidf_vectorizer = TfidfVectorizer(stop_words=stop_words)
tfidf = tfidf_vectorizer.fit_transform(corpus)
```

```
# construir la TF-IDF para n-gramas de un corpus
# min_n: valor mínimo de n
# max_n - valor máximo de n

from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

stop_words = set(stopwords.words('spanish'))

tfidf_vectorizer = TfidfVectorizer(stop_words=stop_words, ngram_range=(min_n, max_n))
tfidf = tfidf_vectorizer.fit_transform(corpus)
```

Teoría

Tokenización: dividir el texto en **tokens**, o sea, frases, palabras y símbolos separados.

Lematización: reducir una palabra a su forma raíz (**lema**).

Corpus: conjunto de textos, generalmente asociados temáticamente entre sí.

Expresión regular: una secuencia de caracteres que define un patrón de búsqueda. Un patrón suele utilizarse para encontrar diferentes ocurrencias que se ajusten al mismo, así por ejemplo, un patrón puede describir diferentes números de teléfono, direcciones de correo electrónico, etc.

Bolsa de palabras: un modelo sencillo para convertir textos en vectores sin tener en cuenta el orden de las palabras en los textos ni su importancia.

TF-IDF: un modelo sencillo para convertir textos en vectores sin tener en cuenta el orden de las palabras, pero sí su importancia.

***N*-grama**: una secuencia de N tokens.

Análisis de sentimientos: tarea de PNL utilizada para identificar el tono de un texto (documento).