

Resumen del capítulo: Vectorización de textos

Lematización

Estos son los pasos para el preprocesamiento de textos:

1. **Tokenización:** tendrás que dividir el texto en tokens (frases, palabras y símbolos);
2. **Lematización:** tendrás que reducir las palabras a sus formas raíz (lema).

Puedes usar estas librerías tanto para la tokenización como para la lematización:

- Natural Language Toolkit (NLTK)
- spaCy

Importa la función de tokenización y crea un objeto de lematización:

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

lemmatizer = WordNetLemmatizer()
```

Pasa a la función `lemmatize()` el texto "Todos los modelos son incorrectos, pero algunos son útiles" como tokens separados:

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

tokens = word_tokenize(text.lower())

lemmas = [lemmatizer.lemmatize(token) for token in tokens]
```

La función `word_tokenize()` divide un texto en tokens y la función `lemmatize()` devuelve el lema de un token que se le pasó. Debido a que nos interesa lematizar una oración, el resultado generalmente se presenta como una lista de tokens lematizados.

```
['all', 'model', 'are', 'wrong', ',', 'but', 'some', 'are', 'useful', '.']
```

También convertimos el texto a minúsculas porque así lo exige el lematizador NLTK.

Usa la función `join()` para volver a convertir la lista de tokens procesados en una línea de texto, separando los elementos con un espacio opcional:

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

Obtenemos:

```
'all model be wrong , but some be useful .'
```

Un conjunto de textos se denomina colectivamente corpus. Por lo general, es necesario definir un corpus para una tarea de análisis de texto (por ejemplo, para construir un diccionario o un espacio vectorial basado en él). Un algoritmo de machine learning procesa cada registro de texto en función de su "posición" en el corpus. No te preocupes, analizaremos todo esto con más detalle en las próximas lecciones.

Crea un corpus a partir de las reseñas. Convierte la columna `review` en una lista de textos.

```
corpus = data['review']
```

Expresiones regulares

Una expresión regular es un instrumento para encontrar patrones complejos en los textos. Estos patrones pueden luego ser manipulados como a ti te guste (extraerlos, reemplazarlos, etc.). ¡Las expresiones regulares son herramientas poderosas que se usan casi en cualquier lugar donde aparezca un texto!

Python tiene un módulo incorporado para trabajar con expresiones regulares: `re` (que significa "expresiones regulares"):

```
import re
```

Echa un vistazo a la función `re.sub()`. Esta encuentra todas las partes del texto que coinciden con el patrón dado y luego las sustituye con el texto elegido.

```
# patrón
# sustitución: con qué debe sustituirse cada coincidencia de patrón
# texto: el texto que la función escanea en busca de coincidencias con el patrón
re.sub(pattern, substitution, text)
```

Solo necesitamos mantener las letras y los espacios en estos textos lematizados de reseñas, así que vamos a escribir una expresión regular para encontrarlos.

La expresión comienza por `r` y va seguida de corchetes entre comillas:

```
r'[]'
```

Todas las letras que coinciden con el patrón se enlistan entre corchetes, sin espacios, y se pueden colocar en cualquier orden. Vamos a encontrar las letras de la "a" a la "z". Si asumimos que pueden estar tanto en minúsculas como en mayúsculas, entonces el código debería escribirse de la siguiente manera:

```
# un rango de letras se indica con un guión:
# a-z = abcdefghijklmnopqrstuvwxyz
r'[a-zA-Z]'
```

Vamos a tomar una de las reseñas del conjunto de datos. Necesitamos mantener todas las letras latinas, los espacios y los apóstrofes, así que nuestro patrón deberá identificarlos correctamente. Si llamamos a `re.sub()`, serán sustituidos por espacios. Para indicar los caracteres que no coinciden con el patrón, pon un caret (^) delante de la secuencia:

```
# texto de reseña
text = ""
Me gustó este programa desde el primer episodio que vi, que fue el episodio "Rhapsody in B
```

```
lue" (para quienes no saben qué es, el Zan se vuelve loco y se convierte en pau episodio n
ivel 10). Los mejores efectos visuales y especiales que había visto en una serie de televi
sión, no hay nada parecido en ninguna parte.
"""
re.sub(r'^a-zA-Z\'', ' ', text)
```

```
" Me gustó este programa desde el primer episodio que vi que fue el episodio Rhapsody in
Blue para quienes no saben qué es el Zan se vuelve loco y se convierte en pau episodio n
ivel 10 Los mejores efectos visuales y especiales que había visto en una serie de tel
evisión no hay nada parecido en ninguna parte "
```

Ahora no nos quedan más que las letras latinas y los espacios. En el siguiente paso, vamos a deshacernos de los espacios extra, ya que dificultan el análisis. Podemos eliminarlos utilizando la combinación de las funciones `join()` y `split()`.

Veamos el ejemplo de un texto con espacios extra en medio, al principio y al final de la línea. Utiliza el método `split()` para convertirlo en una lista. Si llamamos a `split()` sin argumentos, dividirá el texto en espacios o grupos de espacios:

```
text = "          Me gustó este programa "
text.split()
```

El resultado es una lista sin espacios:

```
['Me', 'gustó', 'este', 'programa']
```

Combina estos elementos en una línea con espacios usando el método `join()`:

```
" ".join(['Me', 'gustó', 'este', 'programa'])
```

Entonces obtenemos una línea sin espacios adicionales:

```
'Me gustó este programa'
```

Bolsa de palabras y n-grama

Ahora vamos a aprender cómo podemos convertir datos de texto en datos numéricos, los cuales son un tipo de datos mucho más adecuado para las computadoras. La conversión generalmente se realiza para que una palabra o una oración se convierta en un vector numérico.

Una técnica común para convertir texto se conoce como modelo "bolsa de palabras". Lo que hace es transformar textos en vectores sin tomar en cuenta el orden de las palabras y, por eso, se llama bolsa.

Tomemos como ejemplo un proverbio famoso:

```
For want of a nail the shoe was lost.  
For want of a shoe the horse was lost.  
For want of a horse the rider was lost.
```

Si nos deshacemos de las letras mayúsculas y lematizamos con spaCy, obtenemos esto:

```
for want of a nail the shoe be lose  
for want of a shoe the horse be lose  
for want of a horse the rider be lose
```

Vamos a contar cuántas veces aparece cada palabra:

- "for," "want," "of," "a," "the," "be," "lose" — 3;
- "shoe," "horse" — 2;
- "nail," "rider" — 1.
- "herraje", "caballo" - 2;
- "clavo", "jinete" - 1.

a		be		for		horse		lose		nail		of		rider		shoe		the		want
3		3		3		2		3		1		3		1		2		3		3

Aquí está el vector de este texto:

```
[2, 2, 2, 1, 1, 1, 1, 1]
```

Si hay varios textos, la bolsa de palabras los transforma en una matriz. Las filas representan los textos y las columnas las palabras únicas de todos los textos del corpus. Los números en sus intersecciones representan ocurrencias de cada palabra única.

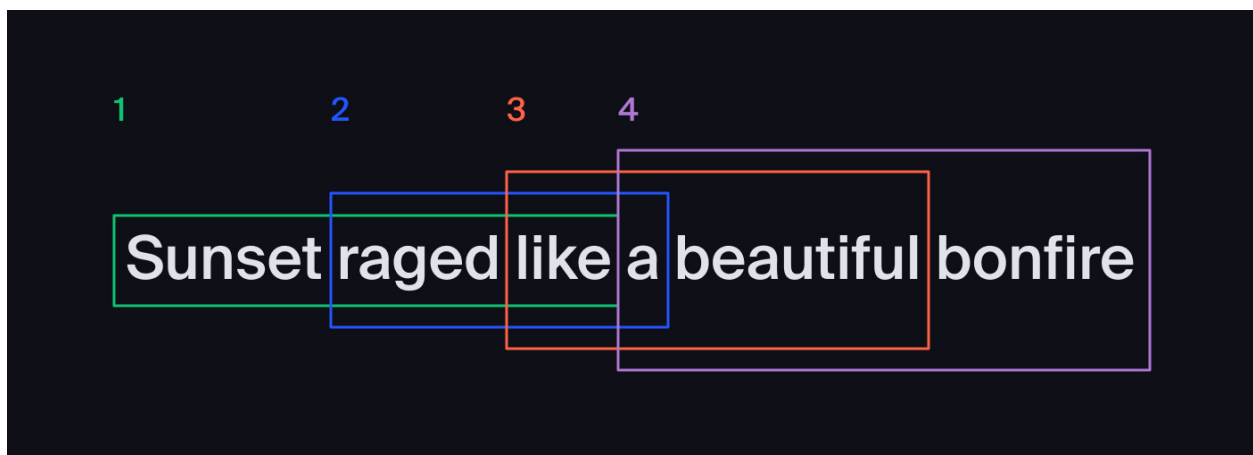
La bolsa de palabras cuenta cada palabra única, pero no considera el orden de las palabras ni las conexiones entre ellas. Por ejemplo, observa este texto lematizado:

```
Peter travel from Tucson to Vegas
```

Esta es la lista de palabras: "Peter", "travel", "from", "Tucson", "to", "Vegas" ("Peter," "viaja," "de," "Tucson," "a", "Vegas"). Entonces ¿a dónde va Peter? Para responder a la pregunta, veamos las frases o **n-gramas**.

Un n-grama es una secuencia de varias palabras. N indica el número de elementos y es arbitrario. Por ejemplo, si $N=1$, tenemos palabras separadas o **unigramas**. Si $N=2$, tenemos frases de dos palabras o **bigramas**. Si $N=3$, tenemos **trigramas**. Está claro, ¿cierto?

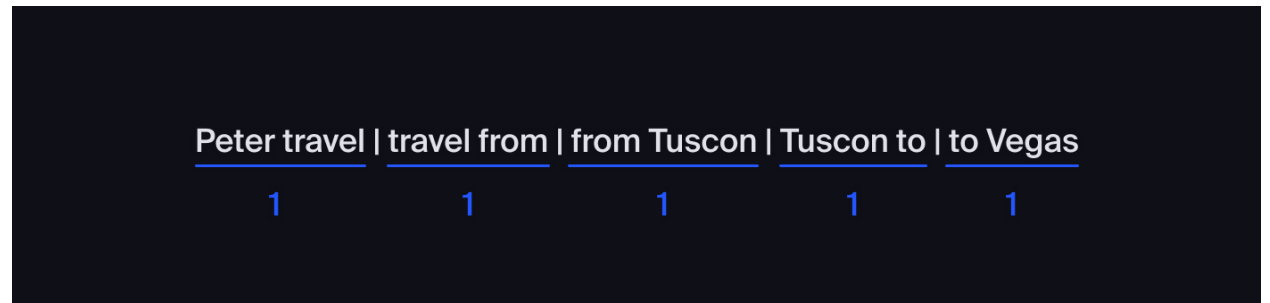
Encontremos todos los trigramas para la oración "Sunset raged like a beautiful bonfire" ("Atardecer ardió como una hermosa fogata.")



Tenemos cuatro trigramas: "Sunset raged like", "raged like a", "like a beautiful", "a beautiful bonfire" ("Atardecer ardió como", "ardió como una", "como una hermosa", "una hermosa fogata"). La palabra "beautiful" no puede iniciar un trigrama porque solo hay una palabra después de esta y necesitamos dos.

Volvamos con Peter. Encuentra los bigramas en el texto. "Peter viaja", "viaja de", "de Tucson", "Tuscon a", "a Vegas." Ahora tenemos las partes A y B del viaje. Peter va de Tucson a Vegas.

Los n-gramas son similares a las bolsas de palabras porque también se pueden convertir en vectores. Este es el vector para el texto de Peter:



The diagram illustrates the process of converting a sentence into a vector using n-grams. The sentence "Peter travel | travel from | from Tuscon | Tuscon to | to Vegas" is shown with vertical bars separating the words. Below each bar, the number "1" is written, indicating the count of each n-gram in the vector.

Trigram	Count
Peter travel	1
travel from	1
from Tuscon	1
Tuscon to	1
to Vegas	1

Crear una bolsa de palabras

Vamos a aprender a crear una bolsa de palabras y a encontrar **palabras vacías**.

Para convertir un corpus de texto en una bolsa de palabras, usa la clase

`CountVectorizer()` del módulo `sklearn.feature_extraction.text`.

Importa la clase:

```
from sklearn.feature_extraction.text import CountVectorizer
```

Crea un contador:

```
count_vect = CountVectorizer()
```

Pasa el corpus de texto al contador. Llama a la función `fit_transform()`. El contador extrae palabras únicas del corpus y cuenta cuántas veces aparecen en cada texto del corpus. El contador no cuenta letras separadas.

```
# bow = bolsa de palabras
bow = count_vect.fit_transform(corpus)
```

Este método devuelve una matriz donde las filas representan textos y las columnas muestran palabras únicas del corpus. Los números en sus intersecciones representan cuántas veces aparece una determinada palabra en el texto.

Usemos el corpus (ya lematizado) de la lección anterior:

```
corpus = [
    'for want of a nail the shoe be lose',
    'for want of a shoe the horse be lose',
    'for want of a horse the rider be lose',
    'for want of a rider the message be lose',
    'for want of a message the battle be lose',
    'for want of a battle the kingdom be lose',
    'and all for the want of a horseshoe nail'
]
```

Vamos a crear una bolsa de palabras para la matriz. Utiliza el atributo `shape` para descubrir el tamaño de la matriz:

```
bow.shape
```

```
(7, 16)
```

El resultado es 7 textos y 16 palabras únicas.

Aquí está nuestra bolsa de palabras como una matriz:

```
print(bow.toarray())
```



```
[[0 0 0 1 0 0 0 1 0 1 1 0 1 1 1 1]
 [0 0 0 1 1 0 0 1 0 0 1 0 1 1 1 1]
 [0 0 0 1 1 0 0 1 0 0 1 1 0 1 1 1]
 [0 0 0 1 0 0 0 1 1 0 1 1 0 1 1 1]
 [0 0 1 1 0 0 0 1 1 0 1 0 0 1 1 1]
 [0 0 1 1 0 0 1 1 0 0 1 0 0 1 1 1]
 [1 1 0 1 0 1 0 0 0 1 1 0 0 1 1 0]]
```

La lista de palabras únicas en la bolsa se llama vocabulario; se almacena en el contador y se puede acceder a ella llamando al método `get_feature_names()`:

```
count_vect.get_feature_names()
```

Aquí está el vocabulario para nuestro ejemplo:

```
['all',
 'and',
 'battle',
 'for',
 'horse',
 'horseshoe',
 'kingdom',
 'lost',
 'message',
 'nail',
 'of',
 'rider',
 'shoe',
 'the',
 'want',
 'was']
```

`CountVectorizer()` también se usa para cálculos de n-gramas. Especifica el tamaño del n-grama con el argumento `ngram_range` para que cuente las frases. Por ejemplo, si necesitamos encontrar frases de dos palabras, debemos especificar el rango de esta manera:

```
count_vect = CountVectorizer(ngram_range=(2, 2))
```

El contador funciona de la misma forma con frases y con palabras.

Dado que un corpus grande conlleva una bolsa de palabras más grande, algunas de las palabras pueden mezclarse y terminar causando más confusión que claridad. Para ayudar con esto, por lo general, puedes eliminar las conjunciones y las preposiciones sin perder el significado de la oración. Si tienes una bolsa de palabras más pequeña y limpia, encontrarás más fácilmente las palabras más importantes para la clasificación del texto.

Para asegurarte de obtener una bolsa de palabras más limpia, encuentra las **palabras vacías** (palabras que no significan nada por sí solas). Hay muchas, y difieren según el idioma. Echemos un vistazo al paquete `stopwords` del módulo `nltk.corpus`:

```
from nltk.corpus import stopwords
```

Deberás descargar el paquete una vez para que funcione:

```
import nltk
nltk.download('stopwords')
```

Llama a la función `stopwords.words()` y utiliza `'spanish'` como un argumento para obtener un conjunto de palabras vacías para español:

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

Pasa la lista de palabras vacías al `CountVectorizer()` al crear el contador:

```
count_vect = CountVectorizer(stop_words=stop_words)
```

Ahora el contador sabe qué palabras se deben excluir de la bolsa de palabras.

TF-IDF

La importancia de una palabra dada se determina por el valor de **TF-IDF (frecuencia de término-frecuencia inversa de documento)**. La TF representa el número de

ocurrencias de una palabra en un texto y la IDF mide la frecuencia con la que aparece en el corpus.

Esta es la fórmula para TF-IDF:

$$TFIDF = TF * IDF$$

Así es como se calcula TF:

$$TF = \frac{t}{n}$$

En la fórmula, t (término) es el número de ocurrencias de la palabra y n es el número total de palabras en el texto.

El papel de la IDF en la fórmula es reducir el peso de las palabras más utilizadas en cualquier otro texto del corpus dado. La IDF depende del número total de textos en un corpus (D) y el número de textos donde aparece la palabra (d).

$$IDF = \log_{10} \left(\frac{D}{d} \right)$$

TF-IDF en sklearn

Puedes calcularla mediante la librería sklearn. La clase `TfidfVectorizer()` se encuentra en el módulo `sklearn.feature_extraction.text`. Impórtala como se muestra a continuación:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

Crea un contador y define palabras vacías, tal como lo hicimos con `CountVectorizer()` :

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

Llama a la función `fit_transform()` para calcular la TF-IDF para el corpus de texto:

```
tf_idf = count_tf_idf.fit_transform(corpus)
```

Podemos calcular los n-gramas al pasar el argumento `ngram_range` a `TfidfVectorizer()` .

Si los datos se dividen en conjuntos de entrenamiento y prueba, llama a la función `fit()` solo para el conjunto de entrenamiento. De lo contrario, la prueba estará sesgada porque el modelo tomará en cuenta las frecuencias de las palabras del conjunto de prueba.

Análisis de sentimiento

Con el fin de determinar el tono del texto, podemos utilizar los valores TF-IDF como características.

El análisis de sentimiento identifica textos cargados de emociones. Esta herramienta puede ser extremadamente útil en los negocios al evaluar las reacciones de los consumidores ante un nuevo producto. Un humano necesitaría varias horas para analizar miles de reseñas, mientras que una computadora lo haría en un par de minutos.

El análisis de sentimiento funciona etiquetando el texto como positivo o negativo. Al texto positivo se le asigna un "1" y al texto negativo se le asigna un "0".