JUN
15

SPRINT 3 - MEETING 39

Procesamiento Lenguaje Natural (Parte 1)

Grabación

Presentación

El equipo docente subió contenido extra de la meeting

Lista de recursos

- DS_Bitácora_39_y_40_NI
- Challenge
- Profundiza

Contenido extra

- Grabación

Procesamiento de Lenguaje Natural

> *"The orcs' response was a deafening onslaught of claws, claws, and claws; even Elrond was forced to retreat. "You are in good hands, dwarf," said Gimli, who had been among the first to charge at the orcs; it took only two words before their opponents were reduced to a blood-soaked quagmire, and the dwarf took his first kill of the night."* —GPT-2.

¡Gracias Martina Cantaro por la creación de los contenidos de esta bitácora!

Hasta ahora, en el curso pudiste analizar información contenida en tablas, expresada mayormente en números (integers o floats) o strings, que en general se componían de unos pocos caracteres. ¡Pero hay muchísima información en el mundo que no está expresada en formato de tablas y números!

Piensa, por ejemplo, en la cantidad de información que transmitimos las personas cotidianamente a través de las palabras. Canciones, historias clínicas, tweets, boletines oficiales, libros, noticias, papers científicos, audios y mensajes de whatsapp, recetas médicas, guiones de teatro, descripciones de productos publicados en mercadolibre, denuncias policiales, fallos judiciales, reclamos, programas de televisión, podcasts. Seguramente se te ocurren muchos ejemplos más. Si tomáramos un solo día de toda esa información, aún sería ENORME.

Y cada vez más, esta información circula por medios digitales. Con las herramientas de NLP podemos analizar masivamente esta información.

Definición

Lo primero que pensamos cuando decimos la palabra "lenguaje" es en aquel que usamos todos los días para comunicarnos entre personas. A este lenguaje se lo llama "lenguaje natural", en contraste a otros lenguajes que se conocen como "lenguajes artificiales".

Ya conoces al menos un lenguaje artificial: Python. Todos los lenguajes de programación son artificiales, y lo que los distingue es que fueron diseñados premeditadamente, siguiendo un plan. En general, podemos señalar el momento, lugar y personas que crearon cada uno de los lenguajes artificiales, y los crearon de manera "top-down": primero pensaron su propósito, luego su arquitectura y sus reglas, y por último, crearon el lenguaje. En cambio, los lenguajes naturales se desarrollan de manera no premeditada, colectiva, "bottom-up", evolucionando con el uso y la repetición. ¡Así fue como se crearon el español, el coreano, el guaraní, el náhuatl e incontables otros!

Los lenguajes naturales tienen muchas reglas que podemos codificar fácilmente, pero hay otros aspectos mucho más sutiles que son más complejos de interpretar por una computadora (pensemos, por ejemplo, en la ironía). Debido a esta complejidad, el procesamiento de lenguaje natural tiene su propia área dentro del campo de estudio de la inteligencia artificial, con conceptos y herramientas propias: lo llamamos **Natural Language Processing** o **NLP.**

¿Cómo se hace NLP?

Cuando hacemos NLP podemos trabajar sobre texto o audio, pero generalmente el audio se pasa primero a texto para hacer el análisis mediante herramientas **speech-to-text.**

Si usas los asistentes de Android (Google Now) o iOS (Siri), verás que pasan tus palabras a texto. Cuando Android recibe la información del usuario en formato audio, lo convierte primero a texto para analizarlo con *deep learning* y devolver una respuesta.

El problema central que tratamos de resolver es: ¿cómo hacemos que la computadora se acerque a *interpretar el significado* del texto?

¡Empecemos por poner manos a la obra! Vamos a trabajar sobre texto en inglés, ya que la mayoría de la documentación está pensada para este lenguaje (aunque muchas librerías cuentan con compatibilidad para otros idiomas, incluyendo el castellano).

Comenzaremos con una librería llamada **Natural Language Tokenizer** o **NLTK.** Esta librería se especializa en tokenizar textos. ¿Qué significa esto? Significa separar cada *unidad semántica* (una manera elegante de decir "un pedacito de texto con significado propio") para poder analizarla por separado. La mayoría de las veces, la unidad semántica es exactamente lo mismo que la *unidad léxica* (que es una manera elegante de decir "palabra"); entonces tokenizar se parece mucho a "separar un texto en palabras". Hay algunas excepciones, ¡ya lo verás!

~~~~
```
from nltk.tokenize import sent_tokenize, word_tokenize

cypher = "Cypher: You know, I know this steak doesn't exist. I know that when I put it in my mouth, the Matrix is telling my brain that it is juicy and delicious. After nine years, you know what I realize? Ignorance is bliss."

print(sent_tokenize(cypher))
~~~~

~~~~
Aquí NLTK hace un buen trabajo separando nuestro texto en oraciones, porque reconoce las puntuaciones correctamente:

```html
~~~~

["Cypher: You know, I know this steak doesn't exist.", 'I know that when I put it in my mouth, the Matrix is telling my brain that it is juicy and delicious.', 'After nine years, you know what I realize?', 'Ignorance is bliss.']

Incluso tiene incorporados criterios un poco más sutiles, como que el punto después del título Mr. no representa el final de una oración:

~~~~
```
agent_smith = "Tell me, Mr. Anderson... what good is a phone call... if you're unable to speak?"
print(nltk.sent_tokenize(agent_smith))
~~~~

~~~~
Vamos a ver cómo trabaja nltk a nivel palabra, usando el primer texto. Este es nuestro input:

```html
~~~~

print(nltk.word_tokenize(cypher))

Y recibimos este output:

~~~~
```
['Cypher', ':', 'You', 'know', ',', 'I', 'know', 'this', 'steak', 'does', "n't", 'exist', '.', 'I', 'know', 'that', 'when', 'I', 'put', 'it', 'in', 'my', 'mouth', ',', 'the', 'Matrix', 'is', 'telling', 'my', 'brain', 'that', 'it', 'is', 'juicy', 'and', 'delicious', '.', 'After', 'nine', 'years', ',', 'you', 'know', 'what', 'I', 'realize', '?', 'Ignorance', 'is', 'bliss', '.']
~~~~

~~~~
¿Has visto lo que pasó con la palabra "doesn't"? NLTK la divide en dos tokens: "does" y "n't". Es decir, por un lado el verbo "hace", y por otro su negación (¡*doesn't* no es más que la conjunción de *does* y *not* abreviado!). A esto nos referíamos cuando decíamos que la tokenización se parece mucho a separar un texto en sus palabras pero a veces hay algo más -como en este caso- donde separamos una palabra en dos tokens.

Esto es porque, para tokenizar el texto, NLTK está haciendo más que cortar la string usando los espacios (esto lo podríamos hacer fácilmente con Python sin tener que usar una librería específica). La librería incluye cierto "conocimiento previo" del lenguaje, que le permite cortar el texto con un criterio que incorpora cierto procesamiento semántico.

## **Stop words**

Mirando un poco más el texto de arriba, si queremos efectuar un análisis, hay tokens que van a ser más útiles que otros. Por ejemplo, los tokens 'brain' o 'Matrix' nos van a dar más información sobre qué significa el texto que los tokens 'the' o ','.

En lenguaje natural, no todas las partes transmiten la misma cantidad de información, algunas son más informativas que otras, es decir, tienen mayor **carga semántica.**

En NLP, muchas veces vamos a querer deshacernos de los tokens de menor carga semántica para trabajar solamente con aquellos de mayor carga. Nos referimos a los tokens descartados como "stop words". Para no tener que hacer el trabajo de identificar y filtrar a mano cada una de las stop words, nltk nos ofrece un catálogo predefinido de *stop words* que podemos modificar a gusto según lo necesitemos.

```html
~~~~

from nltk.corpus import stopwords

Veamos qué es lo que contiene:

~~~~
```
set(stopwords.words('english'))
{'ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'there', 'about', 'once', 'during', 'out', 'very', 'having', 'with', 'they', 'own', 'an', 'be', 'some', 'for', 'do', 'its', 'yours', 'such', 'into', 'of', 'most', 'itself', 'other', 'off', 'is', 's', 'am', 'or', 'who', 'as', 'from', 'him', 'each', 'the', 'themselves', 'until', 'below', 'are', 'we', 'these', 'your', 'his', 'through', 'don', 'nor', 'me', 'were', 'her', 'more', 'himself', 'this', 'down', 'should', 'our', 'their', 'while', 'above', 'both', 'up', 'to', 'ours', 'had', 'she', 'all', 'no', 'when', 'at', 'any', 'before', 'them', 'same', 'and', 'been', 'have', 'in', 'will', 'on', 'does', 'yourselves', 'then', 'that', 'because', 'what', 'over', 'why', 'so', 'can', 'did', 'not', 'now', 'under', 'he', 'you', 'herself', 'has', 'just', 'where', 'too', 'only', 'myself', 'which', 'those', 'i', 'after', 'few', 'whom', 't', 'being', 'if', 'theirs', 'my', 'against', 'a', 'by', 'doing', 'it', 'how', 'further', 'was', 'here', 'than'}
~~~~

~~~~
¿Qué pasa si filtramos nuestro texto quitando las stop words que contiene este catálogo? Nuestro input:

```html
~~~~

stop_words = set(nltk.corpus.stopwords.words('english'))

word*tokens = word*tokenize(cypher)

filtered*sentence = [w for w in word*tokens if not w in stop_words]

filtered_sentence = []

for w in word_tokens: if w not in stop_words: filtered_sentence.append(w)

print(word_tokens) print(filtered_sentence)

El output va a contener nuestro texto antes y después del filtrado:

~~~~
['Cypher', ':', 'You', 'know', ',', 'I', 'know', 'this', 'steak', 'does', "n't", 'exist', '.', 'I', 'know', 'that', 'when', 'I', 'put', 'it', 'in', 'my', 'mouth', ',', 'the', 'Matrix', 'is', 'telling', 'my', 'brain', 'that', 'it', 'is', 'juicy', 'and', 'delicious', '.', 'After', 'nine', 'years', ',', 'you', 'know', 'what', 'I', 'realize', '?', 'Ignorance', 'is', 'bliss', '.']

['Cypher', ':', 'You', 'know', ',', 'I', 'know', 'steak', "n't", 'exist', '.', 'I', 'know', 'I', 'put', 'mouth', ',', 'Matrix', 'telling', 'brain', 'juicy', 'delicious', '.', 'After', 'nine', 'years', ',', 'know', 'I', 'realize', '?', 'Ignorance', 'bliss', '.']
~~~~

¡Mucho mejor! Aunque todavía tenemos signos de puntuación que no nos van a servir mucho para hacer análisis. Las expresiones regulares servirán para filtrarlos.

Regex

Las expresiones regulares (regular expressions) o **regex** son una herramienta incluida en muchos lenguajes de programación que nos permite buscar texto a través de ciertas reglas que definimos.

En Python, la librería para hacer esto se llama re. Para seleccionar de nuestra lista todas las palabras, descartando los signos de puntuación, usamos '\w+', que selecciona solamente caracteres alfanuméricos.

~~~~
```
import re

text = ['Cypher', ':', 'You', 'know', ',', 'I', 'know', 'steak', "n't", 'exist', '.', 'I', 'know', 'I', 'put', 'mouth', ',', 'Matrix', 'telling', 'brain', 'juicy', 'delicious', '.', 'After', 'nine', 'years', ',', 'know', 'I', 'realize', '?', 'Ignorance', 'bliss', '.']

p = re.compile('\w+')

filtered = [ s for s in texto if p.match(s) ]

print(filtered)
~~~~

~~~~
El resultado de esto es:

```html
['Cypher', 'You', 'know', 'I', 'know', 'steak', "n't", 'exist', 'I', 'know', 'I', 'put', 'mouth', 'Matrix', 'telling', 'brain', 'juicy', 'delicious', 'After', 'nine', 'years', 'know', 'I', 'realize', 'Ignorance', 'bliss']
~~~~

Podemos usar varias reglas como esta, hasta llegar a filtros muy complejos que busquen cosas específicas. No vamos a profundizar en regex pero puedes ver una explicación en este link y jugar haciendo regex en esta página.

Información de contexto

En general, cuando una persona lee un artículo periodístico sobre la selva amazónica, lo comprende no solamente por la información que contiene el artículo, sino porque cuenta con décadas de uso diario del lenguaje, muchos artículos leídos anteriormente y cierto conocimiento del tema (por ejemplo, qué es el Amazonas). ¡La computadora, a priori, no conoce nada de esto! Si le diéramos solamente nuestro artículo como información, no le alcanzaría para interpretarlo.

Por suerte, podemos pasarle información para dar un contexto previo al texto que estamos analizando y ayudar a la computadora a interpretarlo. Algunas de las maneras de hacerlo son los **diccionarios** o **lexicons** y los **corpus:**

Diccionarios o lexicons: índice de palabras con sus definiciones. Puede ser el típico diccionario de un idioma, o bien uno orientado a un tema específico. Por ejemplo, un diccionario de fútbol, con todo su vocabulario técnico (incluyendo términos como "mano", "offside" y "DT", y nombres de equipos y jugadores), o de un área de especialidad como la navegación o la astronomía.

Corpus: compendio de texto agrupado según un criterio (por ejemplo: poemas de Mary Oliver, journals médicos, discursos presidenciales, el idioma español)

¡Y conoces algunas herramientas para preprocesar tu texto y pre-entrenar tu algoritmo! En la próxima, aprenderás cómo hacer el análisis.

Empieza a trabajar en el notebook

Para el encuentro, lleva hechas las secciones 1, 2 y 3. La sección 3 también la trabajaremos en clase. Ten en cuenta que también trabajaremos el notebook en el encuentro siguiente, así que tienes para adelantar si así lo deseas.

DS_Bitácora_39_y_40_NLP

¡Prepárate para el próximo encuentro!

Profundiza
Te invitamos a conocer más sobre el tema de esta bitácora.

Challenge
Te proponemos el siguiente desafío, ¿te animas?

Contenido extra de la meeting

Grabación
Código de acceso para acceder: Acamica321*