Introducción al Procesamiento de Lenguaje Natural (PLN)
© ADR Infor SL

eclap.adrformacion.com © ADR Infor SL
JUAN ALBERTO AZEVEDO IBAÑEZ
eclap.adrformacion.com © ADR Infor SL
JUAN ALBERTO AZEVEDO IBAÑEZ

Indice

Competencias y Resultados de Aprendizaje desarrollados en esta unidad

Competencia:
Diseñar, implementar y gestionar chatbots inteligentes e interactivos en aplicaciones web utilizando Django y Python, integrando habilidades de procesamiento de lenguaje natural para mejorar la experiencia del usuario y optimizar la interacción entre usuarios y chatbots.

Resultados de Aprendizaje:

- Implementar funciones básicas de procesamiento de texto usando las bibliotecas NLTK y spaCy.
- Realizar tokenización, análisis de partes del discurso y análisis de entidades nombradas para mejorar la comprensión del lenguaje por parte del chatbot.

Introducción al Procesamiento de Lenguaje Natural (PLN)

Esto empieza a ponerse interesante, vamos a hablar sobre el Procesamiento del Lenguaje Natural (PLN). Es un campo de la inteligencia artificial y la lingüística computacional que se centra en la interacción entre los ordenadores y el lenguaje humano.

Su objetivo principal es permitir que las máquinas comprendan, interpreten y generen texto y habla de manera similar a los humanos.

El lenguaje humano es extremadamente complejo y variado, con una gran cantidad de ambigüedades, reglas gramaticales y contextos que pueden cambiar significados. El PLN aborda estos desafíos mediante el desarrollo de algoritmos y técnicas que permiten a las computadoras procesar, analizar y generar lenguaje humano de manera efectiva.

Tareas comunes en PLN

Estas son las acciones más comunes realizadas con las librerías de procesamiento de lenguaje natural:

Tokenización
Dividir el texto en unidades más pequeñas, como palabras o frases.
Análisis morfológico
Identificar la estructura y la forma de las palabras, incluyendo su raíz y su forma gramatical.
Análisis sintáctico
Analizar la estructura gramatical de las oraciones para comprender cómo las palabras se combinan para formar significados.
Análisis semántico
Comprender el significado de las palabras y las oraciones en función del contexto y el uso.

Análisis pragmático
Interpretar el significado del lenguaje en función del contexto más amplio, como la intención del hablante o las implicaciones sociales.
Generación de lenguaje natural
Crear texto o habla de manera coherente y comprensible para los humanos.

El PLN tiene numerosas aplicaciones prácticas en diversas áreas, incluyendo la traducción automática, la búsqueda de información, la clasificación de documentos, el análisis de sentimientos en redes sociales, la asistencia virtual, entre muchas otras. A medida que avanza la tecnología y se desarrollan nuevos enfoques y algoritmos, el PLN continúa desempeñando un papel fundamental en la creación de sistemas inteligentes que pueden interactuar de manera más natural y efectiva con los humanos.

Bibliotecas de procesamiento de lenguaje natural

A continuación vamos a ver dos bibliotecas básicas en el procesamiento de lenguaje natural, estas son, NLTK y SpaCy, ahora empezamos a entrar de lleno en la materia del PLN y empezaremos a ver numerosos términos nuevos, intentaré que no se nos escape ninguno y que todos sean comprensibles para su estudio.

Tenemos que tener en cuenta que aunque ambas bibliotecas están pensadas para realizar la misma tarea no enfocan el problema de las misma forma por lo que tendrán conceptos comunes pero a la vez también planteamientos diferentes.

NLTK (Natural Language Toolkit)

NLTK (Natural Language Toolkit) es una biblioteca de Python ampliamente utilizada para el procesamiento del lenguaje natural (PLN). Ofrece una amplia gama de herramientas y recursos para trabajar con texto y lenguaje humano.

Con NLTK, los desarrolladores podemos realizar una variedad de tareas de PLN, como tokenización, análisis gramatical, etiquetado de partes del discurso, análisis de sentimientos, extracción de información y mucho más.

La biblioteca NLTK también proporciona acceso a numerosos corpus de texto y modelos de lenguaje preentrenados, lo que facilita el desarrollo y la experimentación con algoritmos de PLN.

NLTK es una herramienta poderosa y versátil para cualquier persona interesada en explorar y desarrollar aplicaciones de procesamiento del lenguaje natural en Python.

Instalación de NLTK

Para empezar a trabajar tenemos que instalar la herramienta, lo haremos con pip, simplemente debes de escribir lo siguiente en consola:


```
pip install nltk
```

Con este comando ya estaremos en condiciones de trabajar con nltk con nuestro ide de programación python.

Si tienes más de una versión de python instalado, asegúrate de para cual estás instalando pues si instalas para 'python3.10' no funcionará en 'python3.11'

Corpus

Un corpus de texto es un conjunto de documentos o textos escritos que se utilizan para análisis lingüísticos y entrenamiento de modelos de procesamiento del lenguaje natural (PLN). Los corpus de texto son fundamentales para el desarrollo y la evaluación de algoritmos de PLN, ya que proporcionan datos de entrada para tareas como la tokenización, el etiquetado de partes del discurso, el análisis de sentimientos, la traducción automática, entre otras.

NLTK proporciona acceso a varios corpus de texto en diferentes idiomas y campos temáticos.

Corpus de texto en español que se pueden utilizar con NLTK y otras herramientas de procesamiento del lenguaje natural.

Corpus CESS-ESP
Este es uno de los corpus más conocidos para el español. Contiene textos de diferentes géneros, como noticias, literatura, ensayos, etc. Está etiquetado por partes del discurso y es ampliamente utilizado para la investigación en PLN.
Corpus de Cine en Español (CineES)
Contiene transcripciones de diálogos de películas en español, etiquetados con información sobre personajes, escenas, etc. Es útil para el análisis de sentimientos y otras aplicaciones relacionadas con el cine.
Corpus del Español del Siglo XXI (CessEs)
Este corpus recopila textos de diferentes géneros (periódicos, blogs, libros, etc.) en español del siglo XXI. Proporciona una amplia muestra del uso del español contemporáneo.

Corpus PAISA

Es un corpus anotado sintácticamente para el español. Contiene textos de diferentes variedades de español y es utilizado para la investigación en análisis sintáctico y otras áreas de la lingüística computacional.

Corpus de Referencia del Español Actual (CREA)

Proporcionado por la Real Academia Española, este corpus contiene una amplia variedad de textos en español de diferentes épocas y géneros. Es una referencia importante para el estudio del español contemporáneo.

Tokenización

Antes de continuar debemos incluir en nuestros prototipos la descarga de algunas herramientas propias de NLTK que no servirán para poder realizar determinadas tareas.

Existen un gran número de ellas pero de momento con las siguientes tenemos para ir empezando.

Puedes descargarlas utilizando la función nltk.download().

Por ejemplo:

```
import nltk
nltk.download('punkt')  # Descargar el tokenizador de NLTK
nltk.download('cess_esp')  # Corpus etiquetado para español
nltk.download('averaged_perceptron_tagger')  # Descargar el etiquetador de partes del discurso
nltk.download('wordnet')  # Descargar WordNet, un diccionario léxico para NLTK
```

Esto lo deberíamos incluir en todos los ejercicios que vayamos haciendo durante esta parte de la unidad, a continuación empezamos con el proceso de tokenización.

La tokenización es el proceso de dividir un texto en unidades más pequeñas, como palabras o frases. NLTK proporciona un tokenizador incorporado que podemos usar de la siguiente manera:

Tokenización por palabras

```
from nltk.tokenize import word_tokenize

# Descargar recursos necesarios para tokenización y etiquetado POS en español

nltk.download('punkt') # Descargar el tokenizador de NLTK

nltk.download('cess_esp')  # Corpus etiquetado para español
```


texto = "NLTK es una biblioteca de Python para procesamiento del lenguaje natural."

tokens = word_tokenize(texto) print(tokens)

y obtendremos la siguiente salida:

['NLTK', 'es', 'una', 'biblioteca', 'de', 'Python', 'para', 'procesamiento', 'del', 'lenguaje', 'natural', '.']

Como puedes ver, el texto se ha dividido en tokens individuales, separando las palabras y los signos de puntuación en elementos distintos. Esto facilita el procesamiento y análisis posteriores del texto.

Tokenización por frases

from nltk.tokenize import sent_tokenize

Descargar recursos necesarios para tokenización y etiquetado POS en español

nltk.download('cess_esp') # Corpus etiquetado para español

texto = "NLTK es una biblioteca de Python para procesamiento del lenguaje natural. Es ampliamente utilizada en la comunidad de PLN."

frases = sent_tokenize(texto) print(frases)

Y la salida será:

['NLTK es una biblioteca de Python para procesamiento del lenguaje natural.', 'Es ampliamente utilizada en la comunidad de PLN.']

Como puedes ver, el texto se ha dividido en dos frases distintas, lo que facilita el procesamiento y análisis individual de cada una de ellas.

Tokenización de Regexp (Expresiones Regulares)

Permite tokenizar un texto utilizando patrones definidos por expresiones regulares. Esto es útil cuando necesitamos tokenizar basándonos en reglas específicas que no están cubiertas por los tokenizadores estándar.

```
from nltk.tokenize import regexp_tokenize

texto = "La fecha actual es 01/04/2024."

tokens = regexp_tokenize(texto, pattern='\w+|$[\d\.]+|\S+')

print(tokens)
```

En este ejemplo, la expresión regular \w+|$[\d\.]+|\S+ divide el texto en palabras (\w+), números con signo de dólar ($[\d\.]+), y otros caracteres no espaciados (\S+).

Tokenización de Espacios en Blanco

Divide el texto en tokens basándose en los espacios en blanco.

```
from nltk.tokenize import WhitespaceTokenizer

texto = "NLTK es una biblioteca de Python."

tokens = WhitespaceTokenizer().tokenize(texto) print(tokens)
```

Este método es útil cuando el texto está bien formateado y los espacios en blanco separan claramente los tokens.

Tokenización Tweet

Específicamente diseñada para tokenizar texto de Twitter, teniendo en cuenta las peculiaridades de los tweets, como hashtags, menciones de usuario, etc.

```
from nltk.tokenize import TweetTokenizer

texto = "Este es un tweet de ejemplo :) #NLTK #procesamientodelenguaje"

tokens = TweetTokenizer().tokenize(texto) print(tokens)
```

Este tokenizador es útil para mantener intactas las características especiales de los tweets.

Estas son solo algunas de las opciones disponibles en NLTK para la tokenización de texto.

Etiquetado de partes del discurso (POS tagging)

El etiquetado de partes del discurso asigna una etiqueta gramatical a cada palabra en un texto (por ejemplo, sustantivo, verbo, adjetivo, etc.). Puedes realizar esta tarea con NLTK de la siguiente manera:

```
import nltk
from nltk.tokenize import word_tokenize

# Descargar un corpus para español
nltk.download('cess_esp')

texto = "NLTK es una biblioteca de Python para procesamiento del lenguaje natural en español."
tokens = word_tokenize(texto)

# Etiquetar las palabras con partes del discurso
etiquetas_pos = nltk.pos_tag(tokens, lang='esp')
print(etiquetas_pos)
```

Estamos etiquetando las partes del discurso de un texto en español.

Primero, descargamos el etiquetador de partes del discurso para español utilizando nltk.download('cess_esp'). Luego, tokenizamos el texto en palabras utilizando word_tokenize, y finalmente etiquetamos las palabras con sus partes del discurso utilizando nltk.pos_tag con el parámetro lang='esp'.

El resultado será una lista de tuplas donde cada tupla contiene una palabra y su etiqueta de parte del discurso en español.

[('NLTK', 'NP'), ('es', 'VMI'), ('una', 'DI'), ('biblioteca', 'NC'), ('de', 'SP'), ('Python', 'NP'), ('para', 'SPS00'), ('procesamiento', 'NC'), ('del', 'SP'), ('lenguaje', 'NC'), ('natural', 'AQ'), ('en', 'SPS00'), ('español', 'AQ'), ('.', 'Fp')]

Aquí, 'NLTK' está etiquetado como 'NP' (nombre propio), 'es' como 'VMI' (verbo principal, indicativo, presente), 'una' como 'DI' (determinante, indefinido), y así sucesivamente.

Análisis de sentimientos

NLTK también proporciona funcionalidad para el análisis de sentimientos, que puede utilizarse para determinar la polaridad de un texto (positivo, negativo o neutral).

El SentimentIntensityAnalyzer de NLTK no funciona bien para español, por lo que te recomiendo usar inglés para los ejemplos

Aquí hay un ejemplo básico:

```
from nltk.sentiment import SentimentIntensityAnalyzer

analizador_sentimientos = SentimentIntensityAnalyzer()
texto = "NLTK is incredibly useful for processing text."
polaridad = analizador_sentimientos.polarity_scores(texto)
print(polaridad)
```

Qué estamos haciendo:

1. Importamos la clase SentimentIntensityAnalyzer del módulo nltk.sentiment.

2. Creamos una instancia de SentimentIntensityAnalyzer, que es un analizador de sentimientos proporcionado por NLTK.
3. Definimos un texto sobre el cual queremos realizar el análisis de sentimientos.
4. Llamamos al método polarity_scores() del analizador de sentimientos y le pasamos el texto como argumento. Este método devuelve un diccionario con cuatro puntuaciones: negativa, neutral, positiva y compuesta.
5. Imprimimos el diccionario de puntuaciones de polaridad resultante.

Salida:

{'neg': 0.0, 'neu': 0.435, 'pos': 0.565, 'compound': 0.5719}

El diccionario de salida indica que el texto tiene una polaridad positiva ('pos': 0.565), con una puntuación compuesta ('compound') de 0.5719. Las puntuaciones neg y neu indican la proporción de negatividad y neutralidad en el texto, respectivamente. La puntuación compuesta combina todas estas puntuaciones en una única medida de polaridad.

Otras funcionalidades

A parte de las que hemos descrito podemos realizar un sin fin de procesamientos diferentes a nuestross textos con la biblioteca NLTK, estas son algunas:

Stemming y lematización
NLTK proporciona herramientas para reducir las palabras a su forma base, como el stemming (cortar las palabras a su raíz) y la lematización (reducción de palabras a su forma base o lema).
Extracción de entidades nombradas (NER)
Permite identificar y clasificar entidades nombradas, como personas, lugares, organizaciones, fechas, etc., en un texto.
Chunking
Es el proceso de agrupar palabras en "chunks" (bloques) significativos, como frases nominales o verbales. Ayuda a identificar partes importantes de un texto.
Análisis de dependencias
NLTK ofrece herramientas para analizar las relaciones gramaticales entre las palabras en una oración, identificando dependencias sintácticas como sujeto, objeto, etc.

Generación de texto
Permite generar texto automáticamente utilizando modelos de lenguaje o reglas gramaticales.
Desambiguación del sentido de las palabras
Ayuda a determinar el significado correcto de una palabra en función del contexto en el que se utiliza.
Traducción de texto
NLTK proporciona herramientas para traducir texto entre diferentes idiomas utilizando modelos de traducción automática.
Resumen de texto
Permite resumir un texto extrayendo las frases más importantes o relevantes.
Detección de idioma
Puede identificar el idioma en el que está escrito un texto dado.
Similitud de texto
Ofrece herramientas para calcular la similitud entre dos textos, útil para tareas como la recuperación de información o la agrupación de documentos.

Ejercicio guiado NLTK: Contar palabras en un texto

Vamos a desarrollar un ejercicio canónico para contar palabras en un texto.

Descripción del ejercicio

Escribe un programa en Python que tome un texto como entrada y cuente el número de palabras distintas que contiene. Utiliza NLTK para realizar la tokenización del texto y luego contar las palabras únicas.

Pasos a seguir

1. Importa la biblioteca NLTK y descarga los recursos necesarios si aún no lo has hecho.
2. Solicita al usuario que ingrese el texto.
3. Utiliza la tokenización de NLTK para dividir el texto en palabras.
4. Crea un conjunto para almacenar las palabras únicas.
5. Itera sobre las palabras tokenizadas y agrega cada una al conjunto.
6. Calcula la longitud del conjunto para obtener el número de palabras distintas.
7. Muestra el resultado al usuario.

Código de ejemplo

```
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')

nltk.download('cess_esp')  # Corpus etiquetado para español

def contar_palabras(texto):
tokens = word_tokenize(texto.lower()) # Convertir texto a minúsculas y tokenizar
palabras_distintas = set(tokens) # Crear un conjunto de palabras únicas
return len(palabras_distintas)

def main():
texto = input("Ingresa un texto: ")
num_palabras_distintas = contar_palabras(texto)
print("El texto contiene", num_palabras_distintas, "palabras distintas.")

if __name__ == "__main__":
main()
```

Ejemplo de uso

```
Ingresa un texto: NLTK es una biblioteca de procesamiento de lenguaje natural en Python.
El texto contiene 9 palabras distintas.
```

Análisis de Sentimientos de un Texto

Escribe un programa en Python que utilice NLTK para realizar un análisis de sentimientos básico en un texto ingresado por el usuario. El programa determinará si el texto tiene una polaridad positiva, negativa o neutra.

Pasos a seguir:

1. Importa la biblioteca NLTK y descarga los recursos necesarios si aún no lo has hecho (vader_lexicon).
2. Importa la clase SentimentIntensityAnalyzer de nltk.sentiment.
3. Crea una instancia del analizador de sentimientos.
4. Solicita al usuario que ingrese el texto (mejor si es en inglés).
5. Utiliza el analizador de sentimientos para obtener las puntuaciones de polaridad del texto.
6. Determina la polaridad del texto basándote en las puntuaciones obtenidas.
7. Muestra el resultado al usuario.

1. Adjunta el archivo.py que realice la tarea solicitada

Procesando respuesta, no cierres el navegador, este proceso podría tardar unos segundos

spaCy "space" (espacio) + "syntax" (sintaxis)

Spacy es otra biblioteca de procesamiento del lenguaje natural (NLP) en Python que ofrece herramientas para realizar tareas como tokenización, lematización, análisis sintáctico, reconocimiento de entidades nombradas (NER) y muchas otras más. Es ampliamente utilizada en aplicaciones de procesamiento del lenguaje natural debido a su eficiencia, facilidad de uso y capacidades de alto rendimiento.

Algunas de las características clave de Spacy son:

Eficiencia
Spacy está diseñado para ser rápido y eficiente en el procesamiento de grandes volúmenes de texto.
Facilidad de uso
Proporciona una API intuitiva y fácil de usar que permite a los desarrolladores implementar rápidamente soluciones de NLP.

Modelos pre-entrenados
Spacy incluye modelos pre-entrenados para varios idiomas, lo que facilita la implementación de aplicaciones de NLP sin necesidad de entrenar modelos desde cero.
Funcionalidades avanzadas
Además de las tareas básicas de NLP, Spacy también ofrece funcionalidades avanzadas como el análisis de dependencias, el reconocimiento de frases nominales, la extracción de relaciones y mucho más.
Integración con otras herramientas
Spacy se integra fácilmente con otras bibliotecas y herramientas de Python, lo que facilita su uso en proyectos que requieren una combinación de diferentes tecnologías.

Instalación de SpaCy

Como siempre que queremos trabajar con una librería especifica hay que instalarla, para ello lo haremos con pip

```
pip install spacy
```

Si tienes más de una versión de python instalado, asegúrate de para cual estás instalando pues si instalas para 'python3.10' no funcionará en 'python3.11'

Modelos

Después, a diferencia NLTK aunque puede trabajar con corpus, vamos a descargar directamente los modelos del idioma con el que vayamos a trabajar, en este caso español 'es'.

Para eso haremos lo siguiente:

```
python -m spacy download es
```

python -m spacy download es_core_news_sm

Todo esto se realiza en consola antes de ponernos a programar nuestros archivos de código.

Tokenización

Como ya sabes la tokenización es el proceso de dividir un texto en unidades más pequeñas llamadas tokens, que pueden ser palabras, números, puntuaciones, etc. Spacy ofrece una herramienta potente y fácil de usar para la tokenización de texto en Python.

Aquí te muestro cómo puedes realizar la tokenización con Spacy:

Tokenización por palabras

```
import spacy

# Cargar el modelo de idioma español

nlp = spacy.load("es_core_news_sm")

# Texto de ejemplo

texto = "Spacy es una excelente herramienta de procesamiento de lenguaje natural."

# Aplicar la tokenización

doc = nlp(texto) # Iterar sobre los tokens

for token in doc: print(token.text)
```

Tokenización por frases

```
import spacy

# Cargar el modelo de idioma

nlp = spacy.load("es_core_news_sm")

# Texto de ejemplo

texto = "Spacy es una excelente herramienta de procesamiento de lenguaje natural. Python es un lenguaje de programación muy popular."

# Procesar el texto

doc = nlp(texto)

# Iterar sobre las frases

for sent in doc.sents:
```

```
print(sent.text)
```

Este código dividirá el texto en frases y luego imprimirá cada frase por separado.

La tokenización de frases puede ser más compleja que la tokenización de palabras, ya que el punto final no siempre indica el final de una oración (por ejemplo, en abreviaturas o en el caso de puntos en nombres de archivo o direcciones de correo electrónico). Spacy trata de abordar estas ambigüedades utilizando un modelo estadístico entrenado para el idioma correspondiente.

Spacy realiza una tokenización inteligente que tiene en cuenta las peculiaridades del idioma, como la presencia de contracciones, palabras compuestas, signos de puntuación, entre otros, lo que lo hace una opción robusta para la tokenización en comparación con enfoques basados únicamente en expresiones regulares.

Etiquetado de partes del discurso (POS tagging)

Para realizar el etiquetado de partes del discurso (POS tagging) con Spacy, puedes utilizar el modelo de idioma correspondiente cargado previamente.

Aquí tienes un ejemplo de cómo hacerlo:

```
import spacy
# Cargar el modelo de idioma
nlp = spacy.load("es_core_news_sm")
# Texto de ejemplo
texto = "Spacy es una excelente herramienta de procesamiento de lenguaje natural. Python es un lenguaje de programación muy popular."
# Procesar el texto
doc = nlp(texto)
# Etiquetado de partes del discurso (POS tagging)
for token in doc:
print(token.text, token.pos_)
```

En este código, después de cargar el modelo de idioma español "es_core_web_sm", procesamos el texto para obtener un objeto Doc. Luego, iteramos sobre cada token en el documento y utilizamos el atributo .pos_ para obtener la etiqueta de la parte del discurso de cada token.

Este código imprimirá cada token junto con su etiqueta de parte del discurso correspondiente.

Spacy PROPN
es AUX
una DET
excelente ADJ
herramienta NOUN
de ADP
procesamiento NOUN
de ADP
lenguaje NOUN
natural ADJ
. PUNCT
Python PROPN
es AUX
un DET
lenguaje NOUN
de ADP
programación NOUN
muy ADV
popular ADJ
. PUNCT

Ejercicios guiados spaCy: Análisis de Entidades Nombradas con spaCy

undefined

Descripción del ejercicio:

Escribe un programa en Python que utilice spaCy para identificar y clasificar las entidades nombradas en un texto ingresado por el usuario. El programa mostrará las entidades nombradas encontradas junto con su tipo (por ejemplo, persona, organización, ubicación, etc.).

Pasos a seguir:

1
Instala spaCy (pip install spacy) si no lo has hecho antes, y descarga el modelo en español (python -m spacy download es_core_news_sm). Esto lo hacemos en consola antes de programar el archivo .py

2

Una vez creado el archivo py, Importa la biblioteca spaCy

3

Definir la función para analizar entidades

1. Carga el modelo en español.
    - nlp = spacy.load("es_core_news_sm")
2. Procesar el texto para obtener el documento analizado
    - doc = nlp(texto)
3. Extraer las entidades nombradas y sus tipos del documento analizado
    - for ent in doc.ents:
entidad = (ent.text, ent.label_)
entidades.append(entidad)
    - return entidades

4

Definir la función principal

1. Solicita al usuario que ingrese el texto en español.
2. Analizar las entidades nombradas en el texto ingresado
    1. entidades = analizar_entidades(texto)
3. Mostrar las entidades nombradas y sus tipos
    1. Si hay las presento si no hay, digo que no hay

Ejemplo: Código de ejemplo:

```
import spacy

def analizar_entidades(texto):
nlp = spacy.load("es_core_news_sm")
doc = nlp(texto)
entidades = [(ent.text, ent.label_) for ent in doc.ents]
return entidades

def main():
texto = input("Ingresa un texto para realizar el análisis de entidades nombradas en español:
")
entidades = analizar_entidades(texto)
if entidades:
print("Entidades nombradas encontradas:")
for entidad, tipo in entidades:
print(f"- {entidad}: {tipo}")
else:
print("No se encontraron entidades nombradas en el texto.")

if __name__ == "__main__": main()
```

Ejemplo de uso:

Ingresa un texto para realizar el análisis de entidades nombradas en español: Madrid es la capital de España.

Entidades nombradas encontradas:
- Madrid: LOC
- España: LOC

Este programa utiliza spaCy para identificar y clasificar las entidades nombradas en un texto en español. Las entidades nombradas encontradas se muestran junto con su tipo, que puede incluir categorías como LOC (ubicación), PER (persona), ORG (organización), etc.

Técnica	Qué hace	Cómo se accede
Tokenización	Divide texto en tokens	for token in doc:
Análisis sintáctico	Detecta funciones gramaticales y jerarquías	token.dep_, token.head
Extracción de entidades	Detecta entidades nombradas como fechas, lugares…	for ent in doc.ents:

Resumen

En las tareas comunes del procesamiento de lenguaje natural, se incluye la tokenización del texto, el etiquetado de partes del discurso y el análisis de sentimientos, que son fundamentales para entender y manipular datos de texto.

Existen diversas bibliotecas que facilitan el procesamiento de lenguaje natural, siendo NLTK y spaCy dos de las más destacadas en Python.

NLTK proporciona un conjunto de herramientas para realizar tareas PLN. Para usarlo, es necesario instalarlo y familiarizarse con su amplia colección de corpus que facilita el desarrollo de análisis textuales complejos.

La tokenización en NLTK permite dividir un texto en palabras o frases. Por ejemplo, con la función `word_tokenize` se pueden separar las palabras de una oración dada.

El etiquetado de partes del discurso (POS tagging) en NLTK ayuda a identificar categorías gramaticales de las palabras, como sustantivos o verbos, en un texto.

NLTK también ofrece capacidades para el análisis de sentimientos, permitiendo determinar el tono emocional de un texto. Esto es útil, por ejemplo, para analizar reseñas de productos.

Entre otras funcionales de NLTK, se incluye el análisis sintáctico, traducción automática y reconocimiento de expresiones regulares.

En un ejercicio guiado con NLTK, se aprende a contar palabras en un texto, lo que es fundamental para obtener estadísticas básicas y filtrar los datos.

SpaCy es otra biblioteca potente para el PLN en Python y requiere instalación antes de su uso. Es conocida por su eficiencia y simplicidad.

Al igual que NLTK, spaCy realiza tokenización y etiquetado de partes del discurso de manera precisa y rápida, facilitando la comprensión estructural de los textos.

spaCy se destaca en el análisis de entidades nombradas, permitiendo extraer nombres de personas, lugares, organizaciones, entre otros, de un texto, lo que es esencial para tareas como el análisis automatizado de documentos o artículos de noticias.

En un ejercicio guiado con spaCy, se puede realizar un análisis de entidades nombradas, proporcionando experiencia práctica en esta técnica avanzada.

Actividades prácticas

Análisis de Entidades Nombradas con spaCy

En esta actividad, escribirás un programa en Python que utilice spaCy para identificar y clasificar las entidades nombradas en un texto ingresado por el usuario. Practicarás tus habilidades en tokenización, etiquetado de partes del discurso y análisis de entidades nombradas mediante el uso de spaCy.

Pasos a seguir:
1. Si no lo has hecho anteriormente, instala spaCy usando spaCy.

Pasos a seguir:
1. Si no lo has hecho anteriormente, instala spaCy usando `pip install spacy` desde la consola.
2. Descarga `python -m spacy download es_core_news_sm`.
3. Crea un archivo Python (.py) y sigue las instrucciones para el desarrollo del programa.
4. Implementa la función `analizar_entidades` modelo de idioma español usando spaCy y extraerá las entidades nombradas junto con sus tipos.
5. Implementa una función principal que recibirá un texto, procesará dicho texto con spaCy y extraerá las entidades nombradas junto con sus tipos.
5. Implementa una función principal que: solicite al usuario un texto en español, llame la función `analizar_entidades` y muestre los resultados.
6. Ejecuta el programa e ingresa un texto, observa cómo el programa identifica y clasifica las entidades nombradas.
7. Finalmente, proporciona un texto de ejemplo similar al que demuestra la correcta identificación de entidades.

1. Desarrolla el código Python que carga spaCy, procesa el texto ingresado y extrae entidades nombradas.

Procesando respuesta, no cierres el navegador, este proceso podría tardar unos segundos