Proyecto IDI 2024

Técnicas de NLP E IA aplicadas a escritos judiciales

Parte 1: Text Mining

introducción

Las empresas modernas disponen de mucha información sobre sus clientes o su sector de actividad. Las nuevas tecnologías como las redes sociales, el comercio electrónico, las aplicaciones móviles, entre otras, dan acceso a un **gran volumen de información**. Al analizar esos datos, es posible **obtener conocimiento** para posteriormente explotarlo de diferentes formas. Sin embargo, algunos **tipos de datos** son más difíciles de explotar que otros.

Como en este proyecto nos vamos a centrar en el análisis de escritos judiciales, vamos a trabajar con textos. La información que se obtiene de estos escritos no tiene formato alguno, es decir, que los datos obtenidos son **no estructurados**. Estos datos no pueden ser tratados correctamente por softwares o herramientas de análisis de datos tradicionales. Por lo que necesitaremos buscar otros métodos para poder tratar los textos y obtener el conocimiento que se necesita. Para esto vamos a recurrir al **text mining**.

¿Qué es el lenguaje natural?

El lenguaje natural es el lenguaje que hablamos todos los días, nuestra forma de comunicarnos por excelencia. Para nosotros es texto sin estructura. Algunos problemas que puede traer son:

1. **Complejidad:** El lenguaje natural es complejo y flexible, permitiendo la expresión de una amplia gama de conceptos y significados.
2. **Ambigüedad:** Las palabras y frases pueden tener múltiples significados dependiendo del contexto, lo que introduce ambigüedad en el lenguaje natural.

¿Qué es el text mining?

El Text Mining, o análisis de textos, consiste en **transformar** un **texto no estructurado** en **datos estructurados** para proceder posteriormente al análisis.

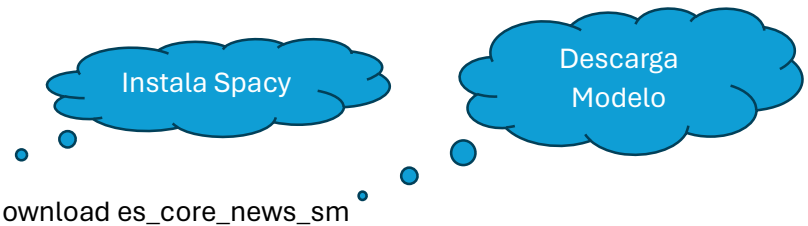
Este trabajo se hace utilizando la tecnología NLP (Natural Language Processing) que consiste en modelos de inteligencia artificial que son capaces de comprender y **tratar textos en lenguaje natural** (no estructurado) de manera automática.

¿Cómo empezamos?

Para empezar a trabajar con text mining y NLP necesitamos: Python y una librería llamada Spacy. Instalar Python: https://www.python.org/. Spacy: https://spacy.io/

Spacy es una biblioteca de procesamiento del lenguaje natural (NLP, por sus siglas en inglés) de código abierto desarrollada en Python. Se utiliza para realizar tareas como tokenización, lematización, reconocimiento de entidades nombradas, análisis de dependencias y más.

Ejemplo 1



Correr en terminal:

- pip install spacy
- python -m spacy download es_core_news_sm
- Repo: https://github.com/Luminicen/NLP-Lanto

Descargar del repositorio de github el código ejemplo1.py en la carpeta **EjemplosBasicosSpacy.**

En este primer ejemplo podemos ver que el modelo, en este caso 'es_core_news_sm', tomó el texto y etiquetó cada palabra según su tipo que puede ser un sustantivo, adjetivo, signo de puntuación, etc. Esto se llama POS tagging (Part-of-Speech tagging).

Entonces, el **POS tagging** asigna a cada palabra en un texto una **etiqueta gramatical** que indica su función gramatical específica en una oración. Estas etiquetas gramaticales **representan las categorías lingüísticas** a las que pertenecen las palabras, como sustantivos, verbos, adjetivos, adverbios, pronombres, preposiciones, conjunciones, entre otro.

Este proceso nos sirve para comprender la **estructura gramatical** de un texto, lo que facilita el análisis y procesamiento automáticos del lenguaje natural.

A partir de ahora vamos a llamar a cada palabra **token.** Y cada texto, ya sea una oración, un texto entero como **documento**.

Lo que vimos al ejecutar el ejemplo1.py son cada token con su pos tagging correspondiente.

Ejemplo 2

Descargar y ejecutar el ejemplo2.py de la carpeta de EjemplosBasicosSpacy del repositorio github.

Lo que vemos en este ejemplo es que a partir del uso de las etiquetas podemos obtener información. En este caso seguimos un patrón especifico donde encuentre un DET + NOUN. Pero se puede extender a otro tipo de patrones y a otro tipo de "etiquetas".

El Matcher de spaCy es una herramienta que permite realizar coincidencias de patrones en documentos procesados por spaCy. Se suele usar para **encontrar** secuencias específicas de **tokens** en un texto basándose en reglas definidas por **patrones**.

Ejemplo 3

Descargar y ejecutar el ejemplo3.py de la carpeta de EjemplosBasicosSpacy del repositorio de github. Una vez ejecutado ir a http://localhost:5000/ y visualizar el resultado.

En este ejemplo podemos ver un gráfico con las dependencias y las etiquetas.

El parser de dependencias de Spacy se encarga de **analizar la estructura gramatical** de una oración y construir un **árbol de dependencias** que representa las **relaciones sintácticas** entre las palabras.

El parser de dependencias de Spacy asigna etiquetas a cada token en función de su función gramatical en la oración y establece las relaciones de dependencia entre ellos. Por ejemplo, algunas de las etiquetas comunes de dependencias incluyen nsubj (sujeto nominal), ROOT (raíz), obj (objeto), amod (modificador adjetival).

Ejemplo 4

Descargar y ejecutar el ejemplo4.py de la carpeta de EjemplosBasicosSpacy del repositorio de github.

En el contexto del Procesamiento del Lenguaje Natural (NLP), una **entidad** se refiere a un **objeto o concepto** que tiene una **existencia real o abstracta** y que puede ser claramente identificado en el texto. Las entidades pueden ser nombres de personas, organizaciones, ubicaciones, fechas, cantidades, productos, términos médicos y cualquier otro tipo de elemento concreto o abstracto que sea relevante para la tarea en cuestión.

En el ejemplo:

- "Estados Unidos": Localización (GPE - Geo-Political Entity)
- "Joe Biden": Persona (PER - Person)
- "Berlín": Localización (LOC - Location)

La mayoría de las etiquetas parecen bastante abstractas, y varían entre los idiomas. spacy.explain te mostrará una breve descripción. Por ejemplo, spacy.explain("VBZ") devuelve "verbo, 3ra persona del singular, presente".

Ayuda útil para entender etiquetas

Ejemplo 5

Descargar y ejecutar el ejemplo5.py de la carpeta de EjemplosBasicosSpacy del repositorio de github.

En este ejemplo podemos ver que podemos **combinar diferentes matchers**. Además, que podemos acceder a **diferentes propiedades** como por ejemplo si está en mayúscula, minúscula.

Ejemplo 6

Descargar y ejecutar el ejemplo6.py de la carpeta de EjemplosBasicosSpacy del repositorio de github.

En spaCy, los operadores en los patrones del Matcher son modificadores que **permiten especificar** la **cantidad** de veces que un token debe aparecer en el texto. Estos operadores añaden flexibilidad a los patrones y permiten hacer coincidencias más generales o específicas. Alguno de ellos:

- *** (Asterisco):** Indica que el token anterior puede aparecer 0 o más veces. Es decir, el token es opcional
- **+ (Más):** Indica que el token anterior debe aparecer 1 o más veces.
- **? (Interrogación):** Indica que el token anterior puede aparecer 0 o 1 vez. Es decir, el token es opcional.

- **! (Exclamación):** Niega el token anterior. Indica que el token no debe aparecer.

Mas Información ver el siguiente **enlace**

Ejercicios

- Imagina que eres un analista de datos y estás encargado de extraer información clave de un texto. Tu tarea es utilizar spaCy para analizar las entidades presentes en los textos y proporcionar un resumen de las entidades detectadas. El texto es el siguiente:
    - El presidente de la compañía XYZ, Juan Pérez, anunció hoy en una conferencia de prensa que la empresa ha alcanzado un acuerdo para adquirir a su competidor principal, Google. La transacción está valuada en 1.5 mil millones de dólares y se espera que se complete a finales de este año.
- Eres un analista de datos que está trabajando con textos no estructurados. Tu jefe te pide que crees un script para poder extraer fechas del formato dd/mm/aaaa. Imprimir en pantalla el resultado. Ayuda: Investigar "SHAPE"
- Estas participando en un proyecto de investigación. El líder del proyecto necesita de tu ayuda para poder extraer de un texto los verbos y sujetos de los siguientes textos. También necesita que obtengas los objetos directos para darles un tratamiento especial por separado. Ayuda: investigar atributo dep:
    - "En el estudio de las ciencias sociales, los investigadores examinan cómo los patrones de migración afectan las dinámicas familiares. El análisis detallado de estos patrones revela tendencias que sugieren cambios en la estructura social."
    - "La investigación médica sobre el impacto de la dieta en la salud cardiovascular demuestra que ciertos alimentos promueven un corazón más saludable. Los datos recopilados indican que las personas que siguen una dieta rica en antioxidantes tienen menos probabilidades de desarrollar enfermedades cardíacas."
    - "En el campo de la inteligencia artificial, los expertos están explorando cómo los algoritmos pueden mejorar la eficiencia en la toma de decisiones. Los resultados preliminares sugieren que la implementación de estas tecnologías puede tener un impacto significativo en diversos sectores."

DESAFIO

Estamos en el proceso de entrenar un modelo de inteligencia artificial para clasificar documentos en tres categorías distintas: Inteligencia Artificial, Matemáticas y Electrónica. Tu tarea consiste en desarrollar un script que permita extraer la información más relevante y detallada de un documento, proporcionando así elementos clave que faciliten su clasificación en la categoría correspondiente.

Probar el script con los siguientes textos:

- En el ámbito de la electrónica, los circuitos integrados desempeñan un papel fundamental. Estos dispositivos compactos contienen componentes electrónicos interconectados, permitiendo la implementación de funciones complejas en dispositivos electrónicos modernos. La miniaturización y la

eficiencia son objetivos clave en el diseño de circuitos integrados para impulsar el avance tecnológico

- La inteligencia artificial (IA) revoluciona la forma en que las máquinas aprenden y realizan tareas. Los algoritmos de aprendizaje profundo, un subcampo de la IA, permiten a las máquinas procesar datos de manera autónoma y mejorar su rendimiento con el tiempo. La IA encuentra aplicaciones en la visión por computadora, el procesamiento del lenguaje natural y la toma de decisiones automatizada.
- Los fractales, una rama fascinante de las matemáticas, exhiben patrones complejos y auto semejantes a diferentes escalas. Un ejemplo icónico es el conjunto de Mandelbrot, cuya estructura revela detalles infinitos. La teoría de fractales se aplica en diversas disciplinas, desde la representación visual hasta la modelización de fenómenos naturales, proporcionando una visión única de la geometría y la complejidad matemática.

Preprocesamiento

Como vimos en la introducción, los **datos recopilados** de diferentes fuentes, como encuestas, aplicaciones, dispositivos, etc., pueden contener errores, inconsistencias y otros **problemas**. Particularmente, cuando tratamos con textos escritos por un humano, podemos encontrarnos errores de ortografía, abreviaciones, emojis, palabras que no agregar información y solo sirven de conectores, entre otras que pueden **generar ruido y sesgos** al momento de utilizar el texto para una aplicación particular. Como vamos a procesar textos y entrenar Inteligencias Artificiales, estos problemas pueden **afectar el rendimiento** de modelos.

¿Qué problemas pueden tener los datos si no preprocesamos?

Alguno de ellos:

- *Ruido*: Datos con errores o fluctuaciones no deseadas que pueden deberse a problemas de medición, entradas incorrectas o interferencia.
- *Datos faltantes*: Falta de información en algunas observaciones o características, lo que puede conducir a una pérdida significativa de información.
- *Inconsistencia*: Inconsistencias entre observaciones o características, lo que puede resultar en contradicciones en los datos.
- *Datos duplicados*: Múltiples copias de la misma información, que pueden introducir sesgos y errores en el análisis.

El **preprocesamiento** de datos ayuda a **eliminar estos errores**, **corregir inconsistencias y mejorar la calidad** de los datos de entrada. Esto a su vez, mejora el rendimiento del sistema de IA al proporcionarle datos limpios y estructurados. Para nuestro caso, no es lo mismo entregarle a un modelo 1700 tokens de un texto a analizar, que 600 tokens que son los más representativos.

A mayor calidad en los datos, tendremos mejores resultados. Específicamente si vamos a entrenar modelos de inteligencia artificial. También surge un riesgo al modificar los datos que es el de introducción de errores, lo que puede afectar negativamente el resultado final.

El preprocesamiento también **depende del uso que se le den los datos.** En nuestro caso, un preprocesamiento puede suponer extracción de palabras de un texto, eliminación de

palabras, reemplazos, entre otras cosas. Esto puede suponer alguna ventaja o una desventaja. Por ejemplo, como vimos al principio de esta parte, si vamos a analizar la semántica de un documento, necesitamos el texto en su totalidad, con sus signos de puntación. Pero en cambio si queremos extraer la información mas importante, no nos sirven los signos de puntuación y ciertas palabras como por ejemplo conectores.

El preprocesamiento se debe hacer en base al trabajo que se quiere hacer con el texto en cuestión. Cualquier modificación que se realice a los datos tiene un gran impacto en el resultado final del procesamiento o tarea a la cual está dirigida.

Normalización

Esta es una de las primeras tareas que se realizan al texto en cuestión. Algunas de las principales tareas que se realizan son las siguientes:

- Se reescribe el documento completo en **minúscula**.
- **Se reemplazan las abreviaturas** por sus palabras completas.
- Se **eliminan caracteres especiales**. Por ejemplo, tags de HTML, caracteres especiales de MS Word, etc.
- Se revisa el documento completo por un **spellchecker**.

Estas tareas se deben realizar de forma cuidadosa, ya que pueden alterar fácilmente el documento. Por ejemplo, con algoritmos de spellchecker al no ser 100% perfectos pueden confundir palabras y reemplazar por otras palabras que no se ajustan a la palabra original. Esto también puede pasar con algoritmos que reemplazan abreviaturas.

Este proceso se puede saltear si por ejemplo si queremos entrenar una inteligencia artificial para detectar si un texto pertenece a una persona X y otro a una persona Y por la forma de escribir. Por ejemplo, si la persona X comete más errores de ortografía que la persona Y. Si la persona Y utiliza abreviaturas mas seguido que X, etc. Nuevamente **depende** de que **aplicación** tenga el texto en cuestión.

STOPWORDS

“Stop words are very common words that carry no meaning or less meaning compared to other keywords”, Kulkarni, A., & Shivananda, A. En general sirven para dar coherencia y naturalidad en el texto. Por ejemplo, algunas stopwords son ‘y’, ‘como’, ‘cuando’. Estas palabras por sí solas no aportan información (no tienen semantica).

Hay otras palabras que tienen significado semántico, pero no aportan información. Si, por ejemplo, estamos en el dominio de ingeniería de software y tenemos la palabra usuario. Nosotros ya sabemos que alguien, el usuario, va a utilizar el software, por lo que no agrega información adicional.

¿Cuál es la importancia de eliminarlas?

Esto se puede ver fácilmente con un ejemplo. Se quiere saber el índice de similitud entre dos documentos:

- Como programar en Python un chatbot
- Como jugar en casa un juego

Para comparar la similitud vamos a usar Jaccard's index. Para ello vamos a tomar las palabras comunes entre ambos documentos y las vamos a dividir por la cantidad total de palabras entre ambos documentos.

Palabras totales: como, programar, en, python, un, chatbot, jugar, casa, juego. Cantidad:9

Palabras en común: como, en, un. Cantidad:3

Los documentos se parecen en un 33% según el índice de Jaccard. Pero podemos ver que no, ya que están hablando de cosas distintas. Programar un chatbot no es lo mismo que Jugar en casa.

¿Qué pasa si eliminamos las StopWords?

- ~~Como~~ programar ~~en~~ Python ~~un~~ chatbot
- ~~Como~~ jugar ~~en~~ casa ~~un~~ juego

Palabras totales: programar, python, chatbot, jugar, casa, juego. Cantidad: 6

Palabras en común: Ø. Cantidad: 0.

Los documentos se parecen en un 0%. Podemos ver que el sesgo de estas stopwords tiene un impacto importante al momento de comparar documentos. Por lo que si necesitamos que un modelo identifique que un documento pertenece a una determinada clasificación, esto podría influir bastante e incluso hasta confundirlo.

Stemming/Lemmatizing

¿Qué pasa con el siguiente ejemplo?

Queremos comparar dos documentos y ver el % de similitud

- El gato está jugando y corriendo un hilo
- El gato juega y corre un hilo

Eliminamos las stopwords

- gato está jugando corriendo hilo
- gato juega corre hilo

Comparamos por el método de Jaccard:

Cantidad de palabras totales: gato, jugando, corriendo, hilo, juega, corre, está (7)

Cantidad de palabras en común: gato, hilo (2)

Obtenemos que se parecen en un 28%. Pero semánticamente podemos ver que los documentos se refieren a lo mismo.

Entonces…

Una palabra puede tener varias formas en un documento. Por ejemplo, jugar, jugando, jugó… Podemos ver que una palabra puede tener múltiples conjugaciones diferentes. Tenemos que buscar alguna forma de representarla de una forma 'estándar'. La idea es poder obtener su forma estándar. Es decir, a partir de palabras conjugadas obtendremos su palabra raíz o estándar.

Para ello tenemos dos técnicas cuyo objetivo es obtener esta palabra raíz pero con diferentes enfoques.

Stemming: se obtiene la raíz a partir de reglas o heurísticas que sacan prefijos y sufijos. Los stemming varían según el lenguaje empleado ya que las reglas son diferentes de un lenguaje a otro. El resultado que devuelve se denomina stem.

Por ejemplo, si tenemos las palabras 'biblioteca' y 'biblotecario', aplicando stemming obtendremos la raíz 'biblotec'. Esta palabra no existe en el diccionario, pero genera una representación univoca.

También puede generar raíces erróneas. Por ejemplo, para la palabra en ingles 'caring' la técnica nos va a devolver 'car'. Podemos ver que es una palabra cuyo significado esta lejos del de 'care'.

Esta técnica es bastante sencilla e implica eliminar caracteres de una palabra siguiendo las reglas o heurísticas que tiene cada lenguaje.

Lemmatizing: para obtener la raíz utiliza un diccionario y devuelve palabras válidas sin conjugar. El resultado que devuelve se denomina lemma.

Por ejemplo, si tenemos las palabras 'best' y 'better' al aplicar esta técnica nos devolvería la palabra 'good' que es una palabra valida y pertenece al diccionario.

Esta técnica es muy compleja, ya que utiliza un montón de tablas y look-ups para poder resolver el lemma.

Volviendo al ejemplo anterior...

- gato está jugando corriendo hilo
- gato juega corre hilo

Aplicamos Lemmatizing

- gato estar jugar correr hilo
- gato jugar correr hilo

Palabras comunes: gato jugar correr hilo (4)

Palabras totales: gato jugar correr hilo estar

Por lo que el índice de Jaccard nos da que son un 80% similares.

Stemming vs Lemmatizing

Stemming	Lemmatization
Stemming is a process that stems or removes last few characters from a word, often leading to incorrect meanings and spelling.	**Lemmatization** considers the context and converts the word to its meaningful base form, which is called Lemma.
For instance, stemming the word '**Caring**' would return '**Car**'.	For instance, lemmatizing the word '**Caring**' would return '**Care**'.
Stemming is used in case of large dataset where performance is an issue.	Lemmatization is computationally expensive since it involves look-up tables and what not.

¿Cuándo conviene usar una u otra? La respuesta es **depende**.

Si tenemos un gran volumen de información a procesar, nos conviene usar la técnica mas liviana. Como Stemming lo que hace es sacar algunos caracteres y por ende el costo de procesamiento es más bajo.

Si necesitamos tener más precisión en los resultados, nos conviene usar más la técnica de Lemmatization.

Ejercicio

- Preprocesar los siguientes textos:
    - En el ámbito de la electrónica, los circuitos integrados desempeñan un papel fundamental. Estos dispositivos compactos contienen componentes electrónicos interconectados, permitiendo la implementación de funciones complejas en dispositivos electrónicos modernos. La miniaturización y la eficiencia son objetivos clave en el diseño de circuitos integrados para impulsar el avance tecnológico
    - La inteligencia artificial (IA) revoluciona la forma en que las máquinas aprenden y realizan tareas. Los algoritmos de aprendizaje profundo, un subcampo de la IA, permiten a las máquinas procesar datos de manera autónoma y mejorar su rendimiento con el tiempo. La IA encuentra aplicaciones en la visión por computadora, el procesamiento del lenguaje natural y la toma de decisiones automatizada.
    - Los fractales, una rama fascinante de las matemáticas, exhiben patrones complejos y auto semejantes a diferentes escalas. Un ejemplo icónico es el conjunto de Mandelbrot, cuya estructura revela detalles infinitos. La teoría de fractales se aplica en diversas disciplinas, desde la representación visual hasta la modelización de fenómenos naturales, proporcionando una visión única de la geometría y la complejidad matemática.

Comparar el resultado obtenido con el ejercicio DESAFIO realizado anteriormente.

En esta primera parte pudimos ver varias formas de procesar un texto, desde la vista semántica y gramatical hasta la sintáctica. Todas las tareas que vimos también se pueden combinar para obtener mejores resultados. Pero como siempre se dijo en esta parte, la aplicación de las diferentes técnicas **depende** de la aplicación a la que este destinado el documento. También vimos ventajas y desventajas de las técnicas en cuestión y las repercusiones que pueden tener en el resultado final. En la siguiente parte vamos a ver como trabajar con Inteligencia Artificial pero orientada al Machine Learning.

Dependency Matcher

Dependency Matcher permite hacer coincidir patrones dentro del análisis de dependencias mediante operadores Semgrex. En lugar de definir una lista de tokens adyacentes como en los Matchers, los patrones de Dependency Matcher coinciden con los tokens del análisis de dependencias y especifican las relaciones entre ellos.

Un patrón agregado al Dependency Matcher consta de una lista de diccionarios, en los que cada diccionario describe un token que debe coincidir y su relación con un token existente en el patrón. A excepción del primero, que define un token de anclaje (un

gancho) usando solo RIGHT_ID y RIGHT_ATTRS, cada patrón debe tener las siguientes claves:

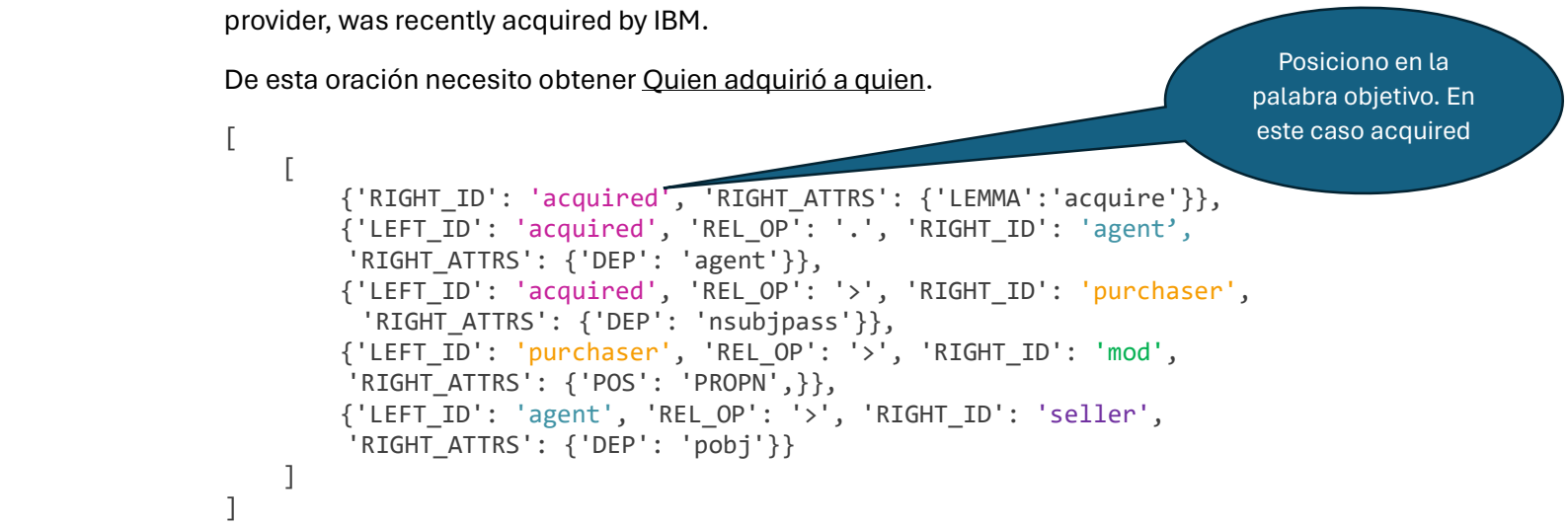
```
{
  "LEFT_ID": "founded",
  "REL_OP": ">",
  "RIGHT_ID": "subject",
  "RIGHT_ATTRS": {"DEP": "nsubj"}
 },
```

Nombre	Descripción
LEFT_ID	El nombre del nodo izquierdo de la relación, que se ha definido en un nodo anterior.
REL_OP	Operador que describe cómo se relacionan los dos nodos.
RIGHT_ID	Un nombre único para el nodo derecho de la relación.
RIGHT_ATTRS	Los atributos de token que deben coincidir para el nodo de la derecha en el mismo formato que los patrones proporcionados al Matcher normal basado en tokens

Cada token adicional agregado al patrón se vincula a un token existente LEFT_ID por la relación REL_OP. Al nuevo token se le asigna el nombre RIGHT_ID y se describe mediante los atributos RIGHT_ATTRS. Ejemplo:

Supongamos la siguiente oración en inglés: Red Hat, a prominent open-source solutions provider, was recently acquired by IBM.

De esta oración necesito obtener Quien adquirió a quien.




```
[
    [
        {'RIGHT_ID': 'acquired', 'RIGHT_ATTRS': {'LEMMA':'acquire'}},
        {'LEFT_ID': 'acquired', 'REL_OP': '.', 'RIGHT_ID': 'agent’,
         'RIGHT_ATTRS': {'DEP': 'agent'}},
        {'LEFT_ID': 'acquired', 'REL_OP': '>', 'RIGHT_ID': 'purchaser',
          'RIGHT_ATTRS': {'DEP': 'nsubjpass'}},
        {'LEFT_ID': 'purchaser', 'REL_OP': '>', 'RIGHT_ID': 'mod',
         'RIGHT_ATTRS': {'POS': 'PROPN',}},
        {'LEFT_ID': 'agent', 'REL_OP': '>', 'RIGHT_ID': 'seller',
         'RIGHT_ATTRS': {'DEP': 'pobj'}}
    ]
]
```

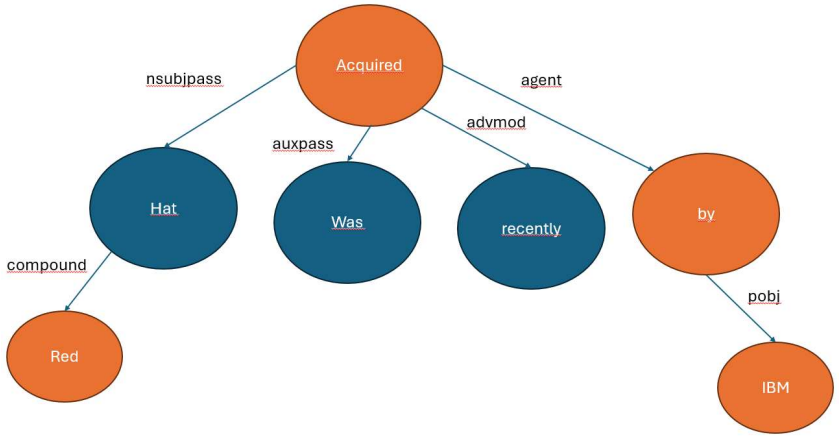
En este ejemplo podemos ver que nos posicionamos en acquired. Pedimos que al lado de acquired este un agente ya que la oración esta en voz pasiva. Podemos ver que solo nos estamos centrando en esta partecita de la oración: “**acquired by**” (by es el agent). Luego busco el sujeto nominal pasivo con el que está relacionado ese acquired, es decir red hat. Hasta acá vamos a obtener “Red Hat acquired by”. Luego vamos a buscar que el agente

tenga una relación con un objeto preposicional porque este inducido por una preposición. Si no dijera “by” seria un objeto directo inducido por el verbo. De ahí pedimos el pobj para quedarnos con IBM. Como resultado tenemos Red Hat acquired by IBM.

Por ahí podemos verlo como un árbol, donde su raíz es la palabra ‘adquired’.

Si miramos el grafico que nos genera displacy* podemos ver a adquired como raíz, en sus nodos hijos tenemos, recently, was,by, Hat. By a su vez tiene un nodo hijo que es IBM. Hat tiene como hijo a Red. Y así sucesivamente…

*Pueden ver el diagrama usando displacy como vimos la vez pasada en el codigo o https://demos.explosion.ai/displacy ingresando la oracion del ejemplo. (Si pego la imagen se ve muy mal)



Otro ejemplo: GitHub, a leading platform for version control and collaborative software development, was acquired by Microsoft.

```
[
    [
        {'RIGHT_ID': 'acquired',
          'RIGHT_ATTRS': {'LEMMA':'acquire'}},
        {'LEFT_ID': 'acquired', 'REL_OP': '.',
         'RIGHT_ID': 'agent', 'RIGHT_ATTRS': {'DEP': 'agent'}},
        {'LEFT_ID': 'acquired', 'REL_OP': '>',
         'RIGHT_ID': 'purchaser',
         'RIGHT_ATTRS': {'DEP': 'nsubjpass'}},
        {'LEFT_ID': 'agent', 'REL_OP': '>',
          'RIGHT_ID': 'seller', 'RIGHT_ATTRS': {'DEP': 'pobj'}}
    ]
]
```

Hacemos la misma trayectoria, nos paramos en la palabra acquired. Nos fijamos que esta palabra este acompañada de un agente y nos lo quedamos (la palabra by). Luego pedimos el sujeto nominal pasivo de acquired y nos lo quedamos (la palabra GitHub). Por último al agente que encontramos le pedimos el pobj y nos lo quedamos (la palabra Microsoft). Como resultado obtenemos GitHub acquired by Microsoft.

Pueden ver el diagrama usando displacy como vimos la vez pasada en el codigo o https://demos.explosion.ai/displacy ingresando la oracion del ejemplo.

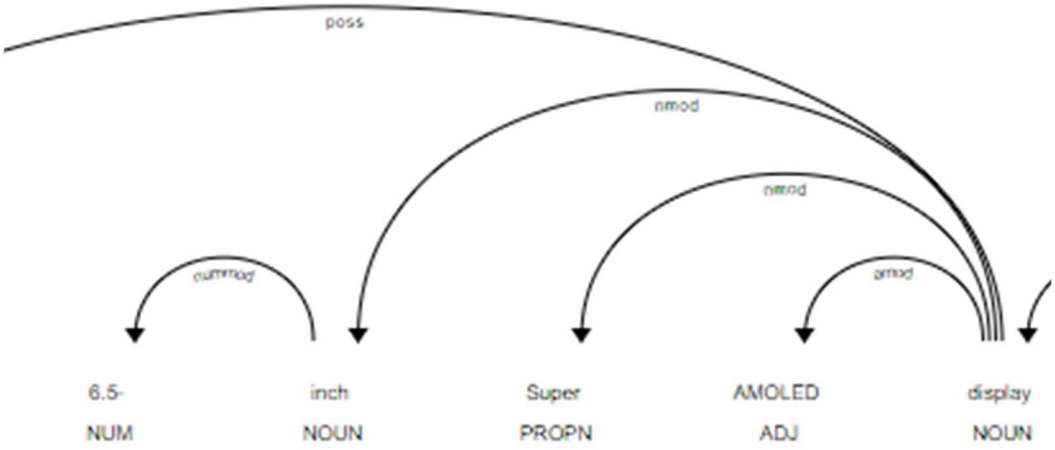
Un ultimo ejemplo: The Galaxy A54's 6.5-inch Super AMOLED display is bright, sharp, and has wide viewing angles.

En este ejemplo nos queremos quedar con algunas de las caracteristicas de la pantalla.

Ancla

Case insensitive

```
[
    {'RIGHT_ID': 'display', 'RIGHT_ATTRS': {"LOWER": "display"}},
    {'LEFT_ID': 'display', 'REL_OP': '>', 'RIGHT_ID': 'mod', 'RIGHT_ATTRS': {'DEP': 'amod'}},
    {'LEFT_ID': 'display', 'REL_OP': '>', 'RIGHT_ID': 'size', 'RIGHT_ATTRS': {'DEP': 'nmod'}},
    {'LEFT_ID': 'size', 'REL_OP': '>', 'RIGHT_ID': 'val', 'RIGHT_ATTRS': {'DEP': 'nummod'}}
]
```



Podemos ver que nos anclamos a display. De ahí vamos a pedir sus hijos, que son amod para quedarnos con AMOLED. Luego del display me dirijo a los nmod y de ahí bajo hasta nummod para quedarme con 6.5.

Resultado: 6.5 inch AMOLED display

¡ATENCION! Dado que los nombres de token únicos en LEFT_ID y RIGHT_ID se usan para identificar tokens, el orden de los diccionarios en los patrones es importante: un nombre de token debe definirse como RIGHT_ID en un diccionario del patrón antes de que se pueda usar como LEFT_ID en otro diccionario.

SYMBOL	DESCRIPTION
`A < B`	`A` is the immediate dependent of `B`.
`A > B`	`A` is the immediate head of `B`.
`A << B`	`A` is the dependent in a chain to `B` following dep → head paths.
`A >> B`	`A` is the head in a chain to `B` following head → dep paths.
`A . B`	`A` immediately precedes `B`, i.e. `A.i == B.i - 1`, and both are within the same dependency tree.
`A .* B`	`A` precedes `B`, i.e. `A.i < B.i`, and both are within the same dependency tree *(Semgrex counterpart: `..`)*.
`A ; B`	`A` immediately follows `B`, i.e. `A.i == B.i + 1`, and both are within the same dependency tree *(Semgrex counterpart: `-`)*.
`A ;* B`	`A` follows `B`, i.e. `A.i > B.i`, and both are within the same dependency tree *(Semgrex counterpart: `--`)*.
`A $+ B`	`B` is a right immediate sibling of `A`, i.e. `A` and `B` have the same parent and `A.i == B.i - 1`.
`A $- B`	`B` is a left immediate sibling of `A`, i.e. `A` and `B` have the same parent and `A.i == B.i + 1`.
`A $++ B`	`B` is a right sibling of `A`, i.e. `A` and `B` have the same parent and `A.i < B.i`.
`A $-- B`	`B` is a left sibling of `A`, i.e. `A` and `B` have the same parent and `A.i > B.i`.
`A >+ B` V3.5.1	`B` is a right immediate child of `A`, i.e. `A` is a parent of `B` and `A.i == B.i - 1` *(not in Semgrex)*.
`A >- B` V3.5.1	`B` is a left immediate child of `A`, i.e. `A` is a parent of `B` and `A.i == B.i + 1` *(not in Semgrex)*.
`A >++ B`	`B` is a right child of `A`, i.e. `A` is a parent of `B` and `A.i < B.i`.
`A >-- B`	`B` is a left child of `A`, i.e. `A` is a parent of `B` and `A.i > B.i`.
`A <+ B` V3.5.1	`B` is a right immediate parent of `A`, i.e. `A` is a child of `B` and `A.i == B.i - 1` *(not in Semgrex)*.
`A <- B` V3.5.1	`B` is a left immediate parent of `A`, i.e. `A` is a child of `B` and `A.i == B.i + 1` *(not in Semgrex)*.
`A <++ B`	`B` is a right parent of `A`, i.e. `A` is a child of `B` and `A.i < B.i`.
`A <-- B`	`B` is a left parent of `A`, i.e. `A` is a child of `B` and `A.i > B.i`.

Fuente: https://spacy.io/usage/rule-based-matching/#dependencymatcher

Ejercitación

- Dado el siguiente texto en inglés:
  - Open IA, the company that created ChatGPT, was recently acquired by Microsoft.
  - Extraer Que empresa compró a la otra. Pista: Conviene usar como palabra de anclaje a “acquired”
  - Extraer qué producto creó Open IA
- Dado el siguiente texto en inglés:
  - In all of its televised BattleBots appearances, Witch Doctor featured a potent vertical spinning disc which has been capable of causing major damage to opponents. It often featured small front wedges which could be replaced with a large steel plow to combat spinners. It was also accompanied by a 30lb minibot named Shaman in early seasons of the show. Shaman was a wedge robot designed to get behind the opponent and distract their drivers so Witch Doctor could go in for the kill. Shaman was not unarmed, as it was also equipped with a flamethrower to toast the underside of opponents it can get underneath.
  - Extraer características del robot “Witch Doctor”.
  - Extraer la función principal de “Shaman”

- Del siguiente texto:
    - Que el juez a cargo del Juzgado Federal de Oberá declaró procedente la extradición de César Elías Fucks a la República Federativa del Brasil para someterlo a proceso por el delito de robo seguido de muerte.
    - Se deberá extraer información para poder llenar e imprimir la siguiente información:

Juez:	
Criminal:	
Destino:	
Delito:	