

**Facultad de Tecnologías Interactivas**

**Desambiguación de Requisitos de Software en español mediante técnicas de Machine Learning**

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

**Autores:**

Raiko Muñoz Hernández

Daniel Ahmed Maya Valdés

**Tutores:**

Daniellys Cervante Lao

Yunieski Coca Bergolla

**Co-tutor:**

Samira de las Mercedes Enríquez González

**La Habana, mayo de 2024**

**“Año 65 de la Revolución”**

**DATOS DE CONTACTO**

**Nombre y apellidos del tutor**: Daniellys Cervante Lao

**Institución**: Universidad de las Ciencias Informáticas

**Título**: Master en Ciencias Informáticas

**Correo electrónico**: [dlao@uci.cu](mailto:dlao@uci.cu)

**Nombre y apellidos del tutor**: Yunieski Coca Bergolla

**Institución**: Universidad de las Ciencias Informáticas

**Título**: Doctor en Ciencias Informáticas

**Correo electrónico**: [ycoca@uci.cu](mailto:ycoca@uci.cu)

**Nombre y apellidos del tutor**: Samira de las Mercedes Enríquez González

**Institución**: Universidad de las Ciencias Informáticas

**Título**: Ingeniera en Ciencias Informáticas

**Correo electrónico**: [samiradlmeg@uci.cu](mailto:samiradlmeg@uci.cu)

**RESUMEN**

<Breve explicación de la investigación como un todo en un único párrafo entre 150 y 250 palabras, que refleje el por qué de la investigación, su objetivo y objeto de estudio, principales métodos o estrategia metodológica en su ejecución, principales hallazgos y resultados obtenidos y principales conclusiones. Debe lograrse al leer el resumen, obtener una panorámica general de la investigación tanto en su carácter teórico como metodológico y práctico>

PALABRAS CLAVE

Ambigüedad, Ingeniería de Requisitos, Procesamiento del Lenguaje Natural, Machine Learning, Deep Learning.

***ABSTRACT***

<*Se escribe en idioma inglés la traducción del texto en el resumen. Se escribe en letra con formato “Italic” para diferenciarlo visualmente de su antecesor en idioma español*>

*KEYWORDS*

*<Se escriben en idioma inglés la traducción de las palabras clave en español. Igual se hace en letra con formato “Italic” para diferenciarlas visualmente de sus antecesoras en idioma español>*

**TABLA DE CONTENIDOS**

[Introducción 9](#_Toc164667266)

[1 Capítulo I.Procesamiento del Lenguaje Natural para la Ingenieria de Requisitos 13](#_Toc164667267)

[1.1 Ingeniería de Requisitos 13](#_Toc164667268)

[1.2 Ambigüedad 13](#_Toc164667269)

[1.2.1 Ambigüedad Semántica 15](#_Toc164667270)

[1.2.1.1 Ambigüedad Anafórica 16](#_Toc164667271)

[1.2.1.2 Ambigüedad de Coordinación 16](#_Toc164667272)

[1.2.1.3 Ambigüedad de Alcance 17](#_Toc164667273)

[1.3 Tipos de aprendizaje en Inteligencia Artificial 18](#_Toc164667274)

[1.3.1 Machine Learning 18](#_Toc164667275)

[1.3.2 Deep Learning como sub-rama del Machine Learning 19](#_Toc164667276)

[1.3.3 ML/Shallow Models vs DL 19](#_Toc164667277)

[1.4 Procesamiento del Lenguaje Natural 21](#_Toc164667278)

[1.4.1 NLP para la IR 21](#_Toc164667279)

[1.4.1.1 Tareas del NLP para la IR 22](#_Toc164667280)

[1.4.1.2 Técnicas del NLP para la IR 24](#_Toc164667281)

[1.4.1.3 Herramientas del NLP para la IR 27](#_Toc164667282)

[1.5 Herramientas SIDARES 29](#_Toc164667283)

[1.6 Deep Learning en la tarea de NLP 30](#_Toc164667284)

[1.7 Desambiguación del Sentido de la Palabra 32](#_Toc164667285)

[1.7.1 Tareas generales de WSD 33](#_Toc164667286)

[1.7.2 Enfoques 34](#_Toc164667287)

[1.7.3 Métodos de los enfoques seleccionados 44](#_Toc164667288)

[1.8 Metodologías para la obtencion de conocimiento 48](#_Toc164667289)

[1.9 Herramientas y Tecnologías 48](#_Toc164667290)

[2 Referencias Bibliográficas 55](#_Toc164667291)

**ÍNDICE DE TABLAS**

<Listado de las tablas en orden de aparición en la memoria>

**ÍNDICE DE FIGURAS**

<Listado de las figuras en orden de aparición en la memoria>

**OPINIÓN DEL(OS) TUTOR(ES)**

<Contenido de la opinión de los tutores>

<Contiene la firma>

# Introducción

La rápida evolución de las Tecnologías de la Información y las Comunicaciones (TIC) en beneficio de la actividad humana, adquiere en gran medida, el desarrollo de una cultura informática en la sociedad para la construcción de un nuevo entorno donde se facilite el quehacer diario del hombre. Cuba no está exenta de este acelerado ritmo si de tecnologías e informática se trata, por lo que también se encuentra en vías de desarrollo. La principal muestra de lo anterior se aprecia en la Universidad de las Ciencias Informáticas (UCI) la cual ha contribuido con la implementación de aplicaciones informáticas para lograr una mayor dinámica en el desarrollo de sus procesos sociales.

Durante el proceso de desarrollo de software, con motivo de entender y registrar las necesidades del cliente para traducirlo al lenguaje informático se hace necesario aplicar las buenas prácticas de la Ingeniería de Requisitos (IR) para obtener las funcionalidades necesarias de la solución a desarrollar. Durante las entrevistas entre el equipo de desarrollo y el cliente, se registran las reglas del negocio y los requerimientos necesarios para dar solución al problema en cuestión. La entrevista se realiza en lenguaje natural, lo que provoca que los miembros del equipo de desarrollo en ocasiones no comprendan los requisitos que el cliente le comunica, esto está dado por las diferentes interpretaciones que se puede realizar sobre una misma expresión. Por esta razón pudieran existir confusiones y/o malentendidos entre ambas partes lo que trae como consecuencia que el producto final no salga con la calidad requerida. Estas inconvenientes suelen aparecer a través del fenómeno de la ambigüedad (Ferrari et al., 2016a).

La ambigüedad durante el proceso de elicitación de requisitos ocurre cuando el cliente comunica una parte de la información y el significado atribuido por el equipo de desarrollo difiere de las intenciones del cliente de una o varias maneras (Ferrari et al., 2016b). El Lenguaje Natural (NL por sus siglas en ingles Natural Language) es la razón base de que la elicitación de requisitos contenga ambigüedad, debido a que en las entrevistas las partes interesadas deben comunicarse mediante el habla, provocando que se vea afectada la calidad final del software a desarrollar llevando a la no conformidad del usuario final (Ashfaq & Bajwa, 2021).

Virtualmente, cualquier lengua humana puede ser tratada por los ordenadores, lógicamente, limitaciones de interés económico o práctico hace que solo las lenguas más habladas o utilizadas en el mundo digital tengan aplicaciones en uso. Para lograr un tratamiento efectivo de estas lenguas, se hace necesario desarrollar el Procesamiento del Lenguaje Natural (NLP por sus siglas en inglés) (Moreno, 2017) que se refiere a la rama de la informática, y más específicamente a la rama de la [inteligencia artificial (IA](https://www.ibm.com/topics/artificial-intelligence)), que se ocupa de dar a las computadoras la capacidad de comprender textos y palabras habladas de la misma manera que los seres humanos (IBM, 2023). Es muy común encontrar en cualquier idioma palabras con múltiples significados; por ejemplo, “lengua” puede significar la capacidad del habla de los humanos u órgano muscular muy movible que se encuentra fijado en la boca de los vertebrados. (ASALE & RAE,2023)

El significado particular de una palabra viene determinado por el contexto que la rodea y en muchas ocasiones por la situación en que se emplea. Si, por ejemplo, se tiene la oración: “Venía corriendo con la lengua afuera”, en este caso, se sabe exactamente el significado asociado a esta palabra. El procedimiento para decidir los significados de las palabras a partir del contexto que las rodea se conoce como “desambiguación” o “*Word Sense Disambiguation*” (WSD por sus siglas en inglés). En NLP las investigaciones en WSD han existido desde la aparición de esta área de investigación. En este campo el problema de la ambigüedad puede tratarse desde distintas perspectivas siendo las Redes Neuronales Transformers (RNT) las más utilizadas en el campo del NLP como solución a la ambigüedad del habla. (Vázquez Pérez, 2009)

A partir de lo expuesto anteriormente y a través de las investigaciones realizadas fue posible determinar que Cuba hoy cuenta con una única herramienta que permite el entrenamiento, la prueba y validación de técnicas para eliminar la ambigüedad de requisitos de software, por lo que se evidencia una disminución de este problema. SIDARES inicialmente fue desarrollada para detectar ambigüedades léxica y sintáctica por lo que aún no tiene implementado entre sus funcionalidades la desambiguación de requisitos con ambigüedad semántica (Ramírez Reyes & Enríquez González, 2022), también se tomó en cuenta que la mayoría de los datasets, corpus, librerías y modelos se encuentran desarrollados en el lenguaje inglés, dificultando el proceso para la evolución de la herramienta. Según el estudio realizado a SIDARES, se pudo notar que esta solo permite importar documentos en formatos .excel y .csv lo que dificulta el trabajo con la herramienta pues las especificaciones de requisitos en su mayoría son realizados en formato .word.

Como consecuencia de la situación planteada anteriormente se define como problema científico: ¿Cómo contribuir a la desambiguación requisitos de software en español utilizando técnicas de Machine Learning?, se define como **objeto de estudio**: las técnicas de Machine Learning para la desambiguación de Requisitos de Software en español, enmarcado en el **campo de acción**: Procesamiento del Lenguaje Natural para la desambiguación de Requisitos de Software en español utilizando técnicas de técnicas de Machine Learning.

Para resolver el problema identificado se propone el siguiente **objetivo general**: Desarrollar una herramienta para la desambiguación de Requisitos de Software en español utilizando técnicas de Machine Learning.

Con motivo de dar cumplimiento al objetivo trazado se han definido las siguientes **tareas de la investigación**:

1. Elaboración del marco teórico de la investigación para establecer las principales tendencias de las herramientas para desambiguación de Requisitos de Software utilizando técnicas de Machine Learning.

2. Realización del análisis y la modelación de los requerimientos de la solución a desarrollar.

3. Asimilación de la metodología, plataforma, tecnologías, librerías, herramientas y pautas definidas por el equipo de investigación para el desarrollo de la solución.

4. Definición de la arquitectura a utilizar para dar solución a la problemática planteada.

5. Implementación de una herramienta para desambiguación de Requisitos de Software utilizando técnicas de Machine Learning.

6. Validación de la solución propuesta mediante el diseño y elaboración de las pruebas para validar los requisitos de la solución propuesta.

Para guiar la investigación fue necesario la utilización de diferentes **métodos científicos** que se relacionan a continuación:

* **Teóricos**

Analítico – Sintético: posibilitará analizar por partes el objeto para facilitar su estudio mediante la determinación de sus componentes principales (PLN para desambiguación de Requisitos Funcionales de Software).

Histórico – Lógico: permitirá estudiar la evolución de los conceptos significativos relacionados con la investigación, así como trabajos similares con puntos coincidentes.

* **Empírico**

Análisis documental: se utilizará en la revisión de la bibliografía consultada para extraer la información necesaria y obtener los referentes teóricos y conceptuales de la investigación siendo ACM, Springer, Google Schoolar e IEEE las principales bases de datos científicas utilizadas como fuente de investigación.

Encuesta: se empleará para conocer el nivel de satisfacción sobre la solución brindada.

Preguntas científicas: Al cabo del proceso de investigación a realizar se persigue dar respuesta a las siguientes preguntas de investigación:

¿En qué situación se encuentran las herramientas desarrolladas para el Procesamiento del Lenguaje Natural en Ingeniería de Requisitos?

¿Cuáles son los tipos de ambigüedad semántica que existen en los requisitos de software?

¿Qué técnicas han sido desarrolladas para detectar y corregir la ambigüedad semántica?

**Capítulo 1 Procesamiento del Lenguaje Natural Para la Ingeniería de Requisitos**: Para lograr un mejor entendimiento del documento el mismo se encuentra estructurado de la siguiente manera Capitulo 1, se realiza un estudio del arte de las literatura sobre la Ingeniería de Requisitos (epígrafe 1.1), la ambigüedad y sus clasificaciones (epígrafe 1.2), los tipos de aprendizaje de la Inteligencia Artificial (epígrafe 1.3), NLP (epígrafe 1.4), SIDARES (epígrafe 1.5), Deep Learning en la tarea de NLP (epígrafe 1.6), Desambiguacion del Sentido de la Palabra (eqpigrafe 1.7), Herramientas y Tecnologias (epígrafe 1.8)

# Capítulo I. Procesamiento del Lenguaje Natural Para la Ingeniería de Requisitos

**Introducción al capítulo**

En este capítulo se realiza un estudio del estado del arte y se exponen conceptos y problemas relacionados con la ambigüedad de requisitos de software. Se hace referencia a los conceptos, técnicas, tareas, actividades y algoritmos del Procesamiento de Lenguaje Natural (NLP). Se describen además la metodología, herramientas y tecnologías utilizadas en el desarrollo de la solución. Todo lo abordado permitió conocer los puntos que pueden ser incluidos en la propuesta de solución que será abordada en el Capítulo 2. Finalmente se describen las características principales de la herramienta SIDARES y como contribuye a la solución de la problemática planteada.

## Ingeniería de Requisitos

La Ingeniería de Requisitos es tratada como una disciplina que tiene por propósito "desarrollar una especificación de requerimientos completa, consistente y no ambigua, la cual servirá como base para acuerdos comunes entre todas las partes involucradas y en dónde se describen las funciones que realizará el sistema" (Toro & Peláez, 2016). El proceso de Ingeniería de Requisitos es una parte importante de la ingeniería de software y es crucial para el desarrollo de sistemas de software en el mundo real (Rafiq et al., 2017). La práctica de levantamiento de requisitos depende en gran medida de algunos factores, incluyendo el tipo de Sistema, el tipo de software desarrollado y el grado de adaptabilidad (Rafiq et al., 2017). En el proceso de desarrollo de software de la IR consta de cuatro escenarios siendo estos el levantamiento o documentación, análisis, validación y especificación (Ashfaq et al., 2021). Durante la etapa de documentación todos los requisitos son redactados en lenguaje natural y por su naturaleza estos contienen ambigüedad debido a las múltiples interpretaciones que el analista de software puede tener sobre las peticiones del cliente, detectar dichas ambigüedades puede resultar difícil por las partes involucradas en el levantamiento de requisitos la mayor parte del tiempo (Gervasi et al., 2019)

## Ambigüedad

Se entiende por ambigüedad a la posibilidad de que una palabra o frase tenga múltiples significados o interpretaciones. Una expresión en lenguaje natural es ambigua cuando puede ser interpretada en diferentes formas. En las entrevistas entre los clientes y los analistas de software, las ambigüedades son asociadas con situaciones de malentendido, cuando la expresión de un cliente no es entendida o incorrectamente interpretada por el analista de requisitos (Ferrari et al., 2016a). Se dividen en cuatro clases como se evidencia en la Tabla 1 y en dependencia del autor la ambigüedad en la IR se agrupa en diferentes clasificaciones como se muestra en la Tabla 2.

Tabla 1: Muestra los diferentes tipos de ambigüedad. Fuente (Ferrari et al., 2016a)

|  |  |
| --- | --- |
| **Ambigüedad** | **Descripción** |
| **No claridad** | Incluye situaciones en las cuales los analistas de requisitos no pueden dar una interpretación o comprensión aceptable a una pieza de información expresada. |
| **Múltiples entendimientos** | Incluye situaciones en las cuales los analistas de requisitos son capaces de dar múltiples interpretaciones aceptables a la expresión del cliente, una correcta y otras incorrectas. |
| **Desambiguación incorrecta** | Incluye situaciones en las cuales los analistas de requisitos asignan una sola interpretación a la expresión del cliente, pero ésta es diferente del sentido que el cliente pretendía. |
| **Desambiguación correcta** | Incluye situaciones en las cuales los analistas de requisitos pueden asignar más de una interpretación al fragmento del discurso del cliente, pero la única interpretación que resulta aceptable para el analista es la que coincide con la pretendida por el cliente. |

Tabla 2: Muestra la clasificación de la ambigüedad en la IR en Lenguaje Natural según varios autores

|  |  |  |  |
| --- | --- | --- | --- |
| **Autores** | | | |
| (Ferrari et al., 2017)  (Porto Dapena, 2018)  (Gervasi et al., 2019)  (Ashfaq & Bajwa, 2021) | (Ferrari et al., 2016a) | (Gupta et al., 2019) | (Ashfaq et al., 2021) |
| Léxica  Sintáctica  Semántica  Pragmática | Léxica  Sintáctica  Semántica  Pragmática  Vaguedad  Generalidad | Léxica  Sintáctica  Semántica  Pragmática  Vaguedad  Generalidad  Genuina  Polisemia  Nociva  Anafórica  Incompletitud  Referencial  De alcance | Léxica  Polisemia  Discurso Referencial  Dominio  Sintáctica  Semántica  Pragmática  Vaguedad  Generalidad |

Tras haberse estudiado la herramienta propuesta por (Ramírez Reyes & Enríquez González, 2022) donde se abordan las ambigüedades Léxica y Sintáctica, este trabajo se centrará en el tratamiento de la ambigüedad Semántica debido a que SIDARES no cuenta con un algoritmo que permita la detección de este tipo de ambigüedad y sus subtipos, dado que mientras más niveles lingüísticos se traten más potente se considera una herramienta y más efectiva será. A continuación, se detallará sobre la ambigüedad semántica y sus subtipos.

### Ambigüedad Semántica

La ambigüedad semántica es un tipo de ambigüedad que trata con el sentido, las posibles interpretaciones de texto. Esta Ocurre cuando una frase leva/posee múltiples formas de interpretarse dentro del contexto proporcionado, aun cuando no exista ambigüedad léxica, sintáctica o estructural (Gupta et al., 2019), (Ferrari et al., 2017), (Bhatia et al., 2016), (Bano, 2015). Es simple demostrar que una palabra tiene múltiples significados viendo el diccionario. No obstante, puede ser difícil probar que no tiene más de un significado, puesto que el lenguaje está en constante evolución y el sentido de las palabras también dependen del contexto en que se usen. (Hiltunen, 2020). Para una mejor comprensión tomemos el ejemplo de, “Todas las luces tienen un interruptor”; ¿un interruptor para muchas luces, o un interruptor para una luz? (Ferrari et al., 2016b). También puede observarse en palabras como banco (de peces, para sentarse, de datos), donde esta puede interpretarse como:

* Grupo masivo de peces
* Entidad donde se almacena dinero, se realizan peticiones de créditos.
* Almacenes digitales utilizados para guardar información.

La ambigüedad semántica como se menciona en la Figura 1 se subdivide en 3 tipos que a continuación se describen.

#### Ambigüedad Anafórica

Una anáfora es un recurso de referencia textual en la que esta retoma la mención de una entidad que ha aparecido con antelación en el discurso (Bruscato & Baptista, 2021), ej. “Existe un prototipo y este será examinado para volver a usarse”, (“este” es una anáfora la cual se refiere a un prototipo). De forma similar una catáfora se refiere a una expresión que ocurrirá más adelante en el texto, ej. Si son válidos, los parámetros serán almacenados en el subsistema de procesamiento de datos, (“son” es una catáfora la cual se refiere a los parámetros). (H. Yang et al., 2011).

La ambigüedad anafórica ocurre cuando algún pronombre, ej. “él”, “este”, “ese”, “esto”, etc., se refiere a una parte previa del texto, siendo llamada antecedente la parte del texto referida (Ferrari et al., 2018). En la IR la anáfora normalmente se limita a la anáfora pronominal, ej. Cuando la anáfora es un pronombre. En la IR la detección de la ambigüedad anafórica es por tanto la tarea de identificar la aparición ambigua de pronombres. (Ezzini et al., 2022).

#### Ambigüedad de Coordinación

Esta ocurre cuando se usan conjunciones de coordinación (ej. y/o), estas llevan a múltiples interpretaciones de una oración, para esta ambigüedad se consideran dos tipos, el primero incluye oraciones donde más de una conjunción de coordinación son usadas en la misma oración. ej. “Hay un cambio de tarifa entre 1 y 100 kws y a partir de 300 kws debe haber un cambio de tarifa”. El segundo tipo incluye oraciones in las que una conjunción coordinadora es utilizada con un modificador, ej. “Enfoques y plataformas estructurados”, (“Estructurado” puede referirse solo al enfoque o a la plataforma también) (Ferrari et al., 2018)

#### Ambigüedad de Alcance

Una ambigüedad de alcance es una ambigüedad estructural que se deriva de interpretaciones alternativas del alcance de un operador. Ejemplos: «Javier hará dieta y ejercicio sólo si su médico lo aprueba» –alcance del operador sentencial: la aprobación del médico es condición necesaria para que haga dieta y también para que haga ejercicio (alcance largo de «sólo si») versus la aprobación es necesaria para el ejercicio pero no para la dieta (alcance largo de «y»); «Alberto tiene una teoría para todo» –alcance del cuantificador: una gran teoría omniexplicativa (alcance largo de «una teoría», que abarca «todo») versus cada cosa es explicada por alguna teoría («todo» tiene alcance largo). El alcance de un operador es la subfórmula entera más corta en la que aparece el operador. Así, en «(A B) ⊃ C», el alcance de «» es «(A B)». En los lenguajes naturales, el alcance de un operador es lo que le está C-subordinado (Y está C-subordinado a X en un diagrama de árbol cuando el nodo domina inmediatamente a X domina también a Y). Una ocurrencia de un operador tiene alcance largo con respecto al de otro operador siempre que el alcance del primero incluye propiamente al alcance del segundo. Ejemplos: en «¬ (A B)», «¬» tiene alcance largo con respecto a «»; en «(∃x) (∀y)Fxy», el cuantificador existencial tiene alcance largo con respecto al cuantificador universal (Audi, s. f.)

Una oración tiene ambigüedad de alcance si esta incluye alguna palabra como mucho, alguna, cada, todo, etc., las cuales cambian el alcance de la oración (Riaz et al., 2019).

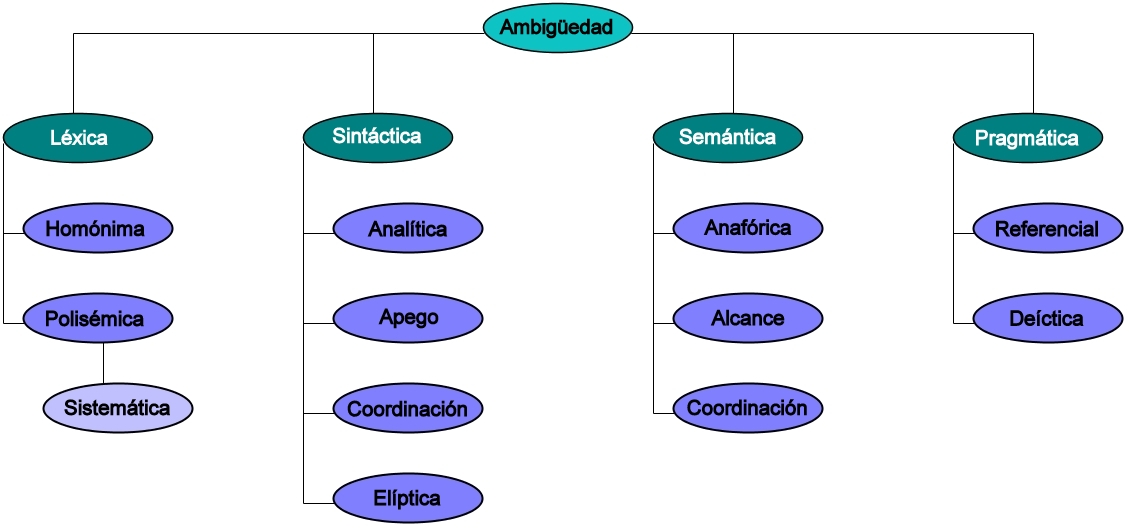


Figura 1: Tipos de ambigüedad adaptado de (Ashfaq & Bajwa, 2021)

## Técnicas de aprendizaje en Inteligencia Artificial

### Machine Learning

El Aprendizaje Automático (ML por sus siglas en inglés *Machine Learning*) o modelos superficiales (shallow models) es la técnica que los sistemas utilizan para mejorar aprendiendo de la experiencia adquirida de los métodos computacionales. (Zhou, 2021) En los sistemas computacionales, dicha experiencia se encuentra en forma de datos, siendo la principal tarea del ML el desarrollar algoritmos de aprendizaje para la construcción de modelos a partir de los datos. (Zhou, 2021) Esto significa que los programas de computadoras mejoran con experiencia con respecto a algunos tipos de tareas y medidas de actuación. Esto es logrado al aplicar algoritmos de que iterativamente aprenden de problemas específicos sobre unos datos de entrenamiento, permitiéndoles a la computadora encontrar señales ocultas y patrones complejos sin haber sido programados para esta función. (Janiesch et al., 2021).

Para algunos autores existen cuatro tipos de aprendizaje para el ML, estos son:

**Supervisado**: Se utilizan datos etiquetados de entrada y de salida y comúnmente se aplican en tareas de clasificación de datos, predecir sentimientos en textos como los tweets y clasificar textos.

**No supervisado**: Analiza los conjuntos de datos no etiquetados sin la necesidad de la inferencia de los humanos, por ejemplo, el proceso de manejo de los datos, se usa en la extracción generativa de características, la identificación de tendencias significativas y estructuras, agrupando en resultados y propósitos de exploración. (Sarker, 2021) Comúnmente se aplican en tareas de clusterización, estimación de destinos, aprendizaje de características, reducción de la dimensionalidad, detección de anomalías definiendo cuan cercano son una pareja de elementos relacionados. (Kamath et al., 2019)

**Semi-supervisado**: es el proceso de aprender conjuntamente de los datos etiquetados y no etiquetados cuando no es posible el etiquetado o la anotación de conjuntos de datos debido al costo o la falta de conocimiento u otras restricciones. (Kamath et al., 2019) El objetivo final de este método es el de proveer un mejor resultado para la predicción que el producido usando los datos etiquetados solamente para el modelo. (Sarker, 2021)

**Reforzado**: es un tipo de algoritmo de ML que permite a los softwares agentes y maquinas el evaluar automáticamente el mejor comportamiento en un contexto en particular o ambiente para mejorar su eficacia. (Sarker, 2021) Se centra en maximizar una recompensa dada una acción o conjunto de acciones realizadas El algoritmo es entrenado para fomentar cierto comportamiento y disuadir otros. Este tiende a funcionar bien en juegos como el ajedrez o el go, en el que la recompensa podría ser ganar el juego. (Kamath et al., 2019)

Al aplicar WSD sobre el modelo preentrenado de FastText, se estaría realizando una tarea de aprendizaje supervisado. En este enfoque, se dispone de un conjunto de datos etiquetados, donde cada palabra ambigua está asociada con su sentido correcto. El modelo aprende a partir de estos datos de entrenamiento para poder predecir el sentido adecuado de nuevas palabras ambiguas.

### Deep Learning como sub-rama del Machine Learning

El Aprendizaje Profundo (DL por sus siglas en ingles Deep Learning) es una de las clases de ML que usa algoritmos para explorar, modelar datos relacionados (Alhogail & Alsabih, 2021) y representar aprendizaje (Sarker, 2021), es utilizado para referirse a redes neuronales con muchas capas consecutivas. En los enfoques tradicionales del ML, las características se definen y extraen cada una de forma manual o haciendo uso de métodos de selección de características. Sin embargo, con los modelos de DL las características son aprendidas y extraídas de forma automática, obteniendo una mayor precisión y desempeño, utilizando Redes Neuronales Artificiales (RNA) y datasets que incluyen textos, sonidos o imágenes, las que actualmente proporcionan las mejores soluciones a muchos problemas en estos campos; así como el Procesamiento del Lenguaje Natural (PLN) (Alhogail & Alsabih, 2021), (Dang et al., 2020).

### ML/Shallow Models vs DL

Las diferencias entre ambos modelos se aprecian a continuación:

* **Tiempo de ejecución**: Incluye el tiempo de entrenamiento y prueba. Debido a la alta complejidad de los modelos de DL, el tiempo de entrenamiento y prueba es mucho más extenso que el de los modelos de ML.
* **Numero de parámetros**: Existen dos tipos de parámetros, los *aprendibles*, se calculan durante la fase de entrenamiento y los *hiperparámetros*, son configurados manualmente antes de comenzar el entrenamiento. Ambos tipos de parámetros superan en gran medida a los shallow models; consecuentemente el entrenamiento y optimización toman más tiempo.
* **Representación de rasgos**: La entrada de datos a los modelos tradicionales de ML es un vector de características, y la ingeniería de características es un paso esencial. Por el contrario, los modelos de DL pueden aprender la representación de características a partir de datos sin procesar y no dependen de la ingeniería de características. Los métodos de DL se pueden ejecutar de un extremo a otro, lo que les brinda una ventaja sobresaliente sobre los métodos tradicionales de ML.
* **Capacidad de aprendizaje**: Las estructuras de los modelos de DL son compleja y contienen un gran número de parámetros (millones o más por lo general). Además, los modelos de DL tienen una fuerte habilidad de ajuste que los shallow models. Sin embargo, los modelos de DL también enfrentan un alto riesgo de sobre ajuste, requiriendo un mayor volumen de datos para su entrenamiento.
* **Interpretabilidad**: Los modelos de DL son como cajas negras; los resultados son casi imposible de interpretar, siendo un factor crítico en el DL. (H. Liu & Lang, 2019)

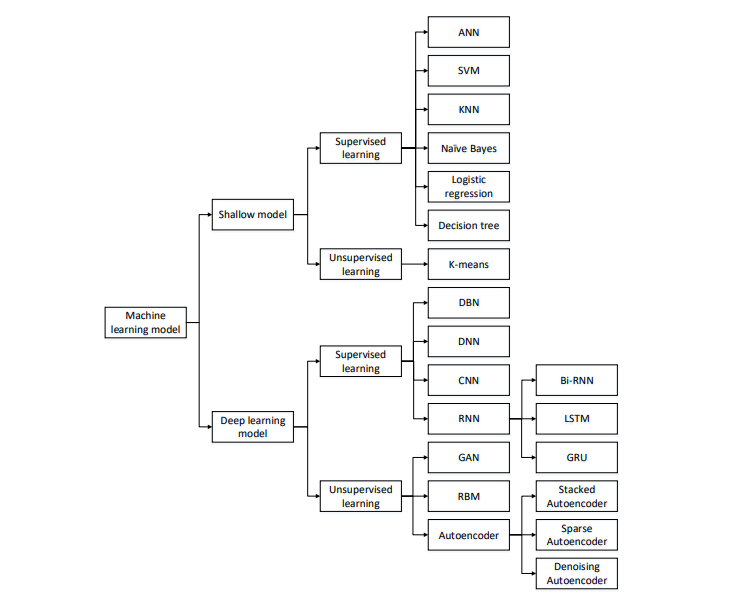


Figura 3: Muestra las categorías de Machine Learning tomado de (H. Liu & Lang, 2019)

La principal ventaja del DL sobre los tradicionales métodos de ML es su mejor rendimiento en algunos casos, en particular el aprendizaje sobre los grandes datasets. El DL permite a las computadoras construir complejos conceptos a partir de simples conceptos, siendo especialmente prometedor para resolver problemas complejos. Partiendo de lo visto anteriormente se ha optado por utilizar el Deep Learning supervisado como posible solución a la problemática que se aborda en nuestra investigación donde se trabajará con algoritmos que basan su arquitectura en los modelos Transformes, se tratará sobre ellos en epígrafes posteriores.

Al aplicar Word Sense Disambiguation sobre el modelo preentrenado de FastText se decidió usar Machine Learning debido a los requisitos computacionales en Deep Learning requiere de mayor potencia para efectuar dicha tarea, requieren mayores volúmenes de datos etiquetados para poder realizar el entrenamiento eficazmente.

## Procesamiento del Lenguaje Natural

El NLP se encuentra entre las técnicas más complicadas en el campo de la Inteligencia Artificial (Hussen Maulud et al., s. f.), es un campo interdisciplinario que combina la lingüística computacional, la ciencia computacional, la ciencia cognitiva y la inteligencia artificial la cual lidia con la con comunicación humana (Kamath et al., 2019), (Deng & Liu, 2018) encargado del estudio y comprensión de la conexión entre el lenguaje humano y las computadoras. Abarca el diseño e implementación de modelos, sistemas y algoritmos para resolver problemas prácticos en el entendimiento del lenguaje humano. (Lauriola et al., 2022). Entre sus aplicaciones se encuentran el reconocimiento del habla, comprensión del lenguaje, sistemas de dialogo, análisis léxico, traducción automática, grafo de conocimiento, la extracción de conocimiento de textos, inferir respuestas mediante algoritmos de inferencia (Hussen Maulud et al., s. f.), (Deng & Liu, 2018). Se puede afirmar que el objetivo principal de la PLN es lograr un procesamiento del lenguaje similar al humano que permita a las computadoras comprender y generar el lenguaje utilizado por los humanos. (Al-Harbi et al., 2017)

### NLP para la IR

Algunos trabajos se han propuesto afrontar el problema de la ambigüedad, con un foco central en la redacción de requisitos. Parte de los trabajos enfocados en la identificación de términos y expresiones que pueden causar ambigüedad (Ferrari & Esuli, 2019) Las herramientas diseñadas para el propósito de levantamiento de requisitos se identifican como herramientas basadas en el lenguaje natural y están dirigidas al procesamiento de especificaciones textuales de requisitos y documentos (Simón et al., 2018).

Procesamiento del Lenguaje Natural para la Ingeniería de Requisito (NLP4RE por sus siglas en ingles Natural Language Processing for Requeriments Engineering) es un área de investigación y desarrollo que persigue emplear técnicas, herramientas de NLP y recursos en el proceso de la [IR](#_Ingeniería_de_Requisitos), para apoyar a los analistas a lidiar con varias tareas de análisis lingüístico en documentos textuales de requisitos, así como detectar problemas en el lenguaje, identificar los conceptos claves y establecer una trazabilidad de los requisitos. (Zhao et al., 2021)

En el desarrollo de software, el ciclo de vida se puede aplicar a todas las fases. Hay artefactos textuales en las fases de análisis y diseño, como el documento de requisitos y la especificación de diseño de software. Los atributos de calidad como el rendimiento, las características, la confiabilidad, la estética y la percepción deben evaluarse en el software de NLP. Estas declaraciones de requisitos deben analizarse y luego debe utilizarse el NLP para ese propósito pues se proporcionan muchas herramientas que ayudan en el análisis lingüístico y ayudan en la asistencia automatizada (Dawood, 2017).

#### Tareas del NLP para la IR

El tipo de tarea en el NLP4RE indica en el ámbito del NLP la tarea que debe realizar una herramienta, define cómo logra este objetivo. Se debe tener en cuenta que una herramienta de NLP4RE puede emplear diferentes tipos de tareas para lograr un objetivo similar (Peer et al., 2024).

* Extracción de texto plano, segmentación, etiquetado POS, análisis sintáctico superficial y de dependencias, reconocimiento de entidades, desambiguación y análisis semántico;
* Analizadores sintácticos o parser, algoritmos de desambiguación, reconocedores de entidades;
* Uso de patrones o reglas, de conocimiento externo y de algoritmos de aprendizaje automático;

A continuación, se detallan otras tareas más centradas a los requisitos de software de las cuales en base al problema que se aborda para la propuesta de solución se decidirá cual tarea utilizar, como plantean algunos autores como se aprecia en la tabla 3.

Tabla 3: Tareas del NLP4RE, adaptado de los autores (Zhao et al., 2021), (Ramírez Reyes & Enríquez González, 2022), (Peer et al., 2024), (Frattini et al., 2024)

|  |  |  |
| --- | --- | --- |
| Tareas | Propósito | Descripción |
| Detección | Problemas lingüísticos son detectados en los documentos de elicitación de requisitos de software | Ayuda a encontrar ambigüedades en los requisitos que fueron pasadas por alto durante la revisión manual |
| Extracción | Reconoce términos claves de dominio y abstracciones | Persigue la extracción de términos de una o varias palabras en los documentos de requisitos para establecer términos de dominio |
| Clasificación | Los requisitos son clasificados en categorías distintas | Pretende realizar la clasificación de los requisitos dentro de sus diferentes tipos, basado en el propósito para la tarea que será aplicada. |
| Modelado | Identifica conceptos de modelado y construye modelos conceptuales | Normalmente esta tarea hace uso de extracción de tareas, tomando diferentes sentidos como la generación de UMLs para apoyar el análisis y diseño |
| Rastreo y relación | Constituye las relaciones que existen entre los requisitos | Asiste a la realización del proceso de seguimiento manual dirigidas a hacer cumplir y demostrar la coherencia del proceso. |
| Búsqueda y recuperación | Buscar y extraer requisitos en los repositorios | Tiene como objetivo la reutilización de requisitos que coincidan con las peticiones de los clientes, apoyando el alcance de dominio en proyectos futuros a desarrollar |
| Generación | Formación de texto nuevo y relevante para los requisitos. | Produce informes, documentos y otros artefactos a partir de requisitos extraídos y analizados. |
| Transformación | Traducción de texto a una especificación semánticamente similar pero sintácticamente diferente. | Convierte los requisitos expresados en lenguaje natural a formatos más formales y estructurados, como modelos. |

#### Técnicas del NLP para la IR

Luego de conocer las principales tareas del NLP4RE se procederá a realizar una revisión de las técnicas que se emplean en este proceso

Tabla 3: Técnicas de NLP más utilizadas en la IR

|  |  |
| --- | --- |
| **Técnica** | **Descripción** |
| ***Word Embedding*** | El término palabras embebidas engloba una serie de técnicas para representar el significado de una palabra en un vector numérico denso dentro del espacio vectorial. Dada esta representación, la similitud semántica entre palabras puede ser computada por medidas de similitud entre vectores. Las palabras embebidas se basan en la hipótesis distribucional de Harris (Harris, 1954), la cual plantea que palabras con similares significados aparecen el contextos lingüísticos similares (Ferrari & Esuli, 2019)  Entre las técnicas de word embedding, una de las más conocidas fue la propuesta por Mikolov[[1]](#footnote-0). Dicha técnica se conoce como salto-grama con muestreo negativo (SGNS por sus siglas en ingles *skip-gram with negative sampling*), la cual fue implementada en el paquete de software de Word2vec. La variante SGNS de Word2vec aprende palabras embebidas como resultado del entrenamiento de una red neuronal de dos capas en la tarea de predecir a partir de una sola palabra *w* de entrada un conjunto de palabras de contexto *cw.* A partir de un documento de corpus, grandes cantidades de ejemplos de entrenamiento pueden ser fácilmente generados (Ferrari & Esuli, 2019) |
| ***Text chunking***/**Cambio de texto** | Es el proceso de descomponer una oración en segmentos no se superponen. Los segmentos incluyen frases sustantivas, frases preposicionales y frases verbales (Arora et al., 2017) |
| ***Clustering refers***/**Clusterización de referencias** | Es la tarea de agrupar objetos relacionados de forma que dichos objetos sean más similares que los objetos de otros clusters (Arora et al., 2017) |
| **Tokenización** | Es el proceso se encarga de descomponer el documento de requisitos no procesado en pequeños fragmentos, siendo conocido este proceso como preparación de los datos (Shreda & Hanani, 2021), resultando en un texto plano separado en párrafos, luego en oraciones y finalmente en palabras (Shreda & Hanani, 2021), (Carreras et al., 2023) |
| **Etiquetado de palabras** | Para este proceso, se adjunta una etiqueta a cada token. Ejemplos del etiquetado:   * **Reconocimiento de entidades nombradas (NER)**, los nombres y ubicaciones de las entidades relevantes se identifican a partir de la secuencia de entrada * **Respuesta a preguntas clásicas**, se utiliza una distribución de probabilidad emitida por un párrafo de entrada para seleccionar un intervalo que contiene la respuesta * **Etiquetado de Parte del Discurso** (**PoS**, ***Part of speech***), es el proceso de marcar/clasificar las partes de las oraciones en un texto correspondiente a una parte particular del discurso (verbo, sustantivo, adjetivo, preposición, ...) (Carreras et al., 2023), (Lauriola et. Al 2022) |
| **Lematización** | Proceso mediante el cual las palabras de un texto que pertenecen a un mismo paradigma flexivo o derivativo, son llevadas a una forma normal que representa a toda la clase (Carreras et al., 2023) |
| **Stemming** | Es una técnica de normalización de texto utilizada para reducir las palabras a su raíz. Esta técnica elimina los afijos de las palabras, lo que puede generar palabras no válidas (*Stemming*, 1994) |
| **Normalización** | En este proceso, se convierten todas las palabras a una secuencia más uniforme transformándolas en formas bases común. (Shreda & Hanani, 2021) |
| **Vectorización** | transforma la oración de requisitos a una característica de representación numérica de forma vectorial de alta dimensión que es utilizada como entrada para entrenar clasificadores de aprendizaje automático. En este proceso, las propiedades de las oraciones de requisitos se extraen en un formato compatible con algoritmos de aprendizaje automático y marcan diferencias para distinguirlas de otras categorías de requisitos (Shreda & Hanani, 2021) |
| **Sequence2sequence**. | Para esta técnica, la secuencia de entrada se utiliza para generar una secuencia de salida. Esta clasificación no es exhaustiva, pero cubre las tareas más populares y relevantes. Entre las tareas de NLP aplicadas a la oración de entrada (Lauriola et. Al 2022) |
| **Análisis basado en reglas** | Emplea reglas gramaticales, reglas semánticas o patrones para el análisis la sintaxis de un texto (Ramírez Reyes & Enríquez González, 2022) |
| **Frecuencia de las palabras** | La frecuencia con la que aparece una palabra en un documento, dividida por el número de palabras que tiene (Ramírez Reyes & Enríquez González, 2022) |
| **Eliminación de palabras vacías** | Es uno de los pasos más comunes para el pre-procesamiento dentro de las tareas de NLP, se basa en eliminar todas las palabras (artículos y pronombres) que aparecen en los corpus (*Stop Word Removal - Natural Language Processing*, s. f.) |
| **Extracción de términos** | Convierte un fragmento de texto en una lista de términos relevantes, útiles para el etiquetado («Term Extraction», s. f.) |
| **WSD** | Determina el sentido correcto de una palabra dado un contexto. (Navigli, 2012) |

Entre todas las técnicas mencionadas anteriormente la más empleada para el NLP4RE es la de Etiquetado POS, siguiéndole la tokenización, el parsing, la eliminación de palabras vacías, extracción de términos y el stemming (Zhao et al., 2021), para lograr la resolución del problema de investigación planteado al inicio del trabajo se ha optado por utilizar dichas técnica incluyendo el Word Embedding y WSD del cual se dedicara un epígrafe más adelante ver epígrafe [1.6](#_Desambiguación_del_Sentido), la Figura 2 permite una mejor comprensión del proceso de aplicación de técnicas sobre el documento de requisitos redactado en NL donde una vez ingresado el documento de texto de los requisitos se procede a la tokenización (para fragmentar el texto), luego se separan las oraciones, sequido se aplica el etiquetado POS (para etiquetar las palabras en verbos, adverbios, sustantivo,etc.), como paso previo al último se identifica la etiqueta según su categoría y finalmente se agrupa la etiqueta con respecto a su módulo . La Figura 3 agrupa las principales técnicas de la IA sobre la IR.

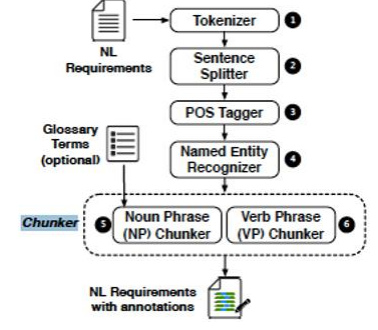


Figura 2: Flujo de aplicación de las técnicas del NLP4RE sobre un documento de requisitos, tomado de (Martínez et al., 2021)

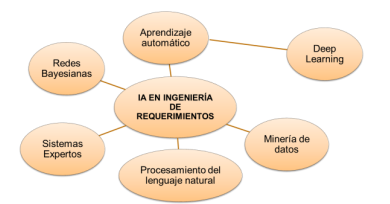
****

Figura 3: Principales técnicas de Inteligencia Artificial aplicadas a la Ingeniería de Requisitos, tomada de (Seleey et al., 2022)

#### Herramientas del NLP para la IR

Tabla 4: Resumen de herramientas de NLP para la IR tomado de (Zhao et al., 2021), (Espinosa Mateo, 2023)

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre** | **Año** | **Disponibilidad** | **Sitio Web** |
| **Tiger Pro** | 2004 | Libre para estudiantes |  |
| **ReqSimile** | 2005 | Software gratuito de código abierto, versión beta | <http://reqsimile.sourceforge.net> |
| **RQA** | 2011 | Licenciada |  |
| **Visure quality analyzer 2011** | 2011 | De pago |  |
| **NASA reconstructed ARM** | 2011 | No disponible |  |
| **Pragmatic Ambiguity Detector** | 2012 | Software gratuito de código abierto | <https://github.com/isti-fmt-nlp/Pragmatic-Ambiguity-Detector> |
| **NFR Locator** | 2013 | Software gratuito de código abierto | <https://github.com/RealsearchGroup/NFRLocator> |
| **ReqWiki** | 2013 | Aplicación Web de código abierto | <http://www.semanticsoftware.info/reqwiki> |
| **SpecQua** | 2014 | Prueba gratuita | <http://specqua.apphb.com> |
| **IntelliReq** | 2014 | Web bloqueada | <http://www.intellireq.org> |
| **QUALICEN** | 2014 | Licenciada |  |
| **Innoslate** | 2014 | Libre |  |
| **QUOD** | 2014 | Libre |  |
| **aToucan** | 2015 | Necesita permiso para acceder | <https://sites.google.com/site/taoyue/atoucan-models> |
| **CMT & FDE** | 2015 | Software gratuito de código abierto | <https://github.com/isti-fmt-nlp/tool-NLPtoFP> |
| **NARCIA** | 2015 | Instalable en el ordenador | <https://sites.google.com/site/svvnarcia/> |
| **Visual Narrator** | 2016 | Software gratuito de código abierto en GitHub | <https://github.com/MarcelRobeer/VisualNarrator> |
| **AnModeler** | 2016 | Software descargable | <https://sites.google.com/site/anmodeler/> |
| **AQUSA** | 2016 | Código abierto gratuito con interfaz de comandos | <https://github.com/RELabUU/aqusa-core> |
| **Qvscribe** | 2016 | Licenciada |  |
| **TACTILE check** | 2017 | Libre |  |
| **FeatureX** | 2018 | Código abierto gratuito con interfaz de comandos | <https://github.com/5Quintessential/FeatureX> |
| **PUMConf** | 2018 | Instalable en el ordenador | <https://sites.google.com/site/pumconf/> |
| **Requirement Assesment Tool (RAT)** | 2018 | Libre |  |
| **Text2UseCase** | 2019 | Aplicación basada en la web, de prueba gratuita | <https://sites.google.com/view/text2usecase/home> |
| **GuideGen** | 2019 | Software gratuito de código abierto | <https://github.com/hotomski/guidegen> |
| **RE-SWOT** | 2019 | Software gratuito de código abierto en GitHub | <https://github.com/RELabUU/re-swot> |
| **TT-RecS** | 2020 | Software gratuito de código abierto | <https://zenodo.org/records/3827169> |
| **SIDARES** | 2021 | Libre para la UCI |  |
| **CiRA** | 2021 | Software gratuito de código abierto en GitHub | <http://www.cira.bth.se/demo> |

## Herramientas SIDARES

SIDARES (Sistema de Detección de Ambigüedades en Requisitos de Software) constituye una herramienta de PLN desarrollada en la Universidad de las Ciencias Informáticas (UCI) con el objetivo de detectar la ambigüedad léxica y sintáctica presente en los requisitos de software, la cual utiliza el lenguaje de programación Python y la herramienta de procesamiento del lenguaje natural NLTK debido a que es de código libre. La herramienta se encarga de procesar y analizar exhaustivamente los requisitos de software y a su vez detectar los defectos en ellos (Espinosa Mateo, 2023). Por la relevancia de esta herramienta para la validación de requisitos en el desarrollo de software en la UCI, es necesario el seguir mejorando dicha herramienta y para lograr este propósito se ha decidido desarrollar un módulo que se encargue de la detección de ambigüedades semánticas encargado de sugerir que términos son más correctos para solucionar dicha ambigüedad en los documentos de requisitos de software en español.

## Deep Learning en la tarea de NLP

Las redes neuronales profundas (DNN por sus siglas en inglés Deep Neural Networks) han ganado una popularidad considerable al abordar una amplia variedad de problemas de aprendizaje automático, como la clasificación de imágenes, la detección de objetos y la detección de malware. Además, se ha empleado el DL para aprender representaciones efectivas de los registros de pacientes con el fin de realizar modelos predictivos y respaldar el feno-tipado de enfermedades. Los DNN también son la opción más popular para aplicaciones desafiantes en reconocimiento de voz, síntesis de voz, traducción de idiomas y NLP (Alshemali & Kalita, 2020). Las DNN suelen constar de más de dos capas ocultas, organizadas en arquitecturas de red profundamente anidadas (Janiesch et al., 2021), algunas de las cuales son capas ocultas, ver Figura 4. Estas utilizan modelos matemáticos sofisticados para procesar datos de muchas maneras diferentes y operaciones avanzadas (por ejemplo, convoluciones) o activaciones múltiples en una neurona en lugar de utilizar una función de activación simple. Estas características permiten que las redes neuronales profundas se alimenten con datos de entrada sin procesar y descubran automáticamente una representación necesaria para la tarea de aprendizaje correspondiente. Esta es la capacidad central de las redes, que comúnmente se conoce como DL (Janiesch et al., 2021).

Una red neuronal (NN por sus siglas en ingles Neural Networks) es un modelo ajustable de salidas como funciones de entradas, que consta de varias capas: una capa de entrada, que incluye datos de entrada; capas ocultas, incluidos nodos de procesamiento llamados neuronas; y una capa de salida, que incluye una o varias neuronas, cuyas salidas son las salidas de la red (Dang et al., 2020). Los NN se modelan como colecciones de capas de neuronas que están conectadas en un gráfico acíclico (Alshemali & Kalita, 2020)

En un flujo de trabajo de DL para el NLP, ver Figura 4 donde se compara el proceso tradicional del NLP con respecto al D, primero se pre-procesa el documento. Luego, el artefacto resultante se procesa mediante un algoritmo de incrustación de palabras (por ejemplo, Word2Vec) y se pasa a uno o más NN. Un elemento central de la NLP en DL son las incrustaciones de palabras. Esta forma de representar el contexto como incrustaciones permite que los modelos de lenguaje generalicen muy bien datos invisibles. Al incorporar palabras, éstas se transforman de símbolos distintos y aislados en objetos matemáticos sobre los que se puede operar. Las incrustaciones de palabras se generan mediante el uso de redes/algoritmos neuronales ya entrenados previamente, donde su entrada es un corpus de texto. Uno de los primeros y populares algoritmos de incrustación se conoce como Word2Vec. Hoy en día existen muchas variantes del algoritmo Word2Vec, como fastText[[2]](#footnote-1) o GloVe (Landolt et al., 2021)

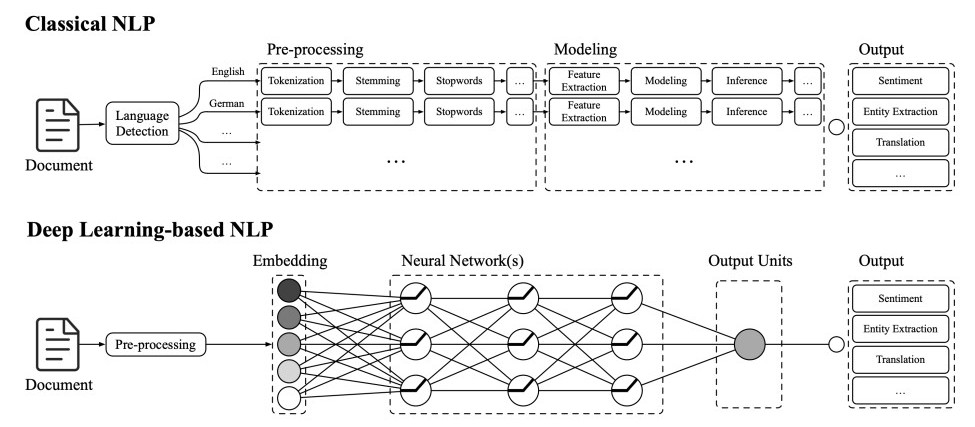


Figura 4: Diferencias del proceso de NLP sobre el DL, tomado de (Landolt et al., 2021)

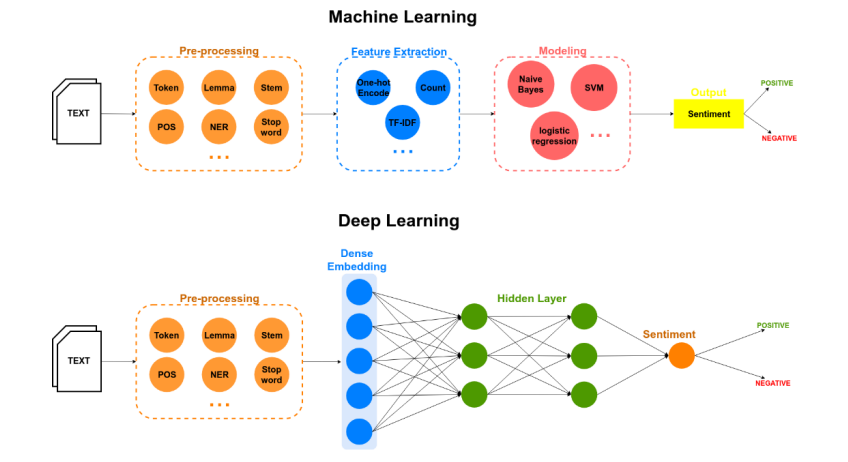


Figura 4: Diferencias entre dos enfoques de clasificación, ML y DL. Parte del discurso (POS); Reconocimiento de entidad nombrada (NER); Término Frecuencia de documento inversa (TF-IDF), tomado de

## Desambiguación del Sentido de la Palabra

Desambiguación del Sentido de la Palabra (WSD por sus siglas en ingles Word Sense Disambiguation) tiene como objetivo identificar el significado semántico subyacente de la información formulada con (posiblemente) diferentes redacciones y estilos sintácticos, para ayudar a identificar la información que es más pertinente para las necesidades del usuario (Tekli, 2016) y encontrar la relación semántica entre dos oraciones, en la Figura 4 se representan los enfoques para WSD; es un tema central de investigación en lingüística computacional y NLP donde la técnica del WSD es aplicable para resolver muchas tareas de procesamiento del lenguaje natural, como resúmenes de textos, respuesta a preguntas, recuperación de información y clasificación de textos (Rahman & Borah, 2022). La razón de su importancia radica en la ambigüedad del lenguaje humano, que es tan generalizado que un gran número de palabras pueden interpretarse de múltiples maneras dependiendo del contexto en el que ocurren. Es considerada por muchos el primer muro en el camino a la comprensión automática de textos y un recurso potencialmente importante en aplicaciones como extracción de información y traducción automática (Navigli, 2012), (Tekli, 2016).

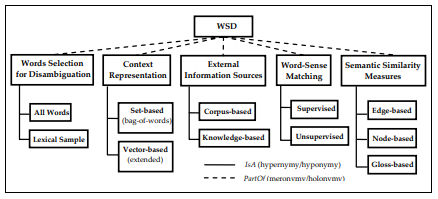


Figura 4: Taxonomía simplificada de las propiedades de los enfoques para WSD, tomado de (Tekli, 2016)

### Tareas generales de WSD

**Seleccionar palabras para la desambiguación**: Existen dos posibles métodos para la selección de palabras objetivo para la desambiguación:

* **todas las palabras/*all-words***, se persigue que el sistema elimine la ambigüedad de todas las palabras en un documento de texto (plano). Aunque es considerado un enfoque de desambiguación total y exhaustivo.
* **muestra léxica/*lexical-sample***, se seleccionan palabras objetivo específicas para la desambiguación (generalmente una palabra por oración). Dichas palabras suelen ser las más ambiguas y por lo general se eligen utilizando métodos de aprendizaje supervisados ​​entrenados para reconocer palabras destacadas en oraciones.

**Identificar y representar contextos de palabras**: Una vez seleccionadas las palabras para la desambiguación, se deben identificar sus contextos para ser utilizados en el proceso. Como WSD se basa en la noción de contexto, las palabras que aparecen juntas en el mismo contexto suelen tener significados relacionados. Se pueden utilizar otras características para describir el contexto, como la información resultante del pre-procesamiento lingüístico, incluidas las etiquetas de parte del discurso (verbo, sujeto, etc.) y relaciones gramaticales (verbo-sujeto, verbo-objeto, etc.).

**Utilizar fuentes de conocimiento de referencia**: Los métodos WSD dependiendo del tipo de fuentes de conocimiento externas de las que dependen, se pueden distinguir como

* **basados ​​en corpus**, se basa en datos, pues implica información sobre palabras previamente desambiguadas y requiere aprendizaje supervisado a partir de corpus etiquetados por sentido (SemCor y OntoNotes) donde las palabras/expresiones se han asociado con un significado semántico explícito, para permitir predicciones de nuevas palabras.
* **basados ​​en conocimiento**, están impulsados ​​por el conocimiento, pues manejan un inventario de sentidos estructurado (y explícito) y/o un depósito de información sobre palabras que puede usarse automáticamente para distinguir sus significados en el texto.

**Asociar sentidos con palabras**: Como paso de cierre en WSD se asocian sentidos con palabras, teniendo en cuenta los contextos de las palabras objetivo, así como hacer referencia al conocimiento externo sobre los sentidos de las palabras. Esto suele verse como una tarea de clasificación del sentido de las palabras. En este sentido, los enfoques WSD pueden clasificarse a grandes rasgos como supervisados ​​o no supervisados. Por un lado, los métodos supervisados implican el uso de técnicas de aprendizaje automático, utilizando muestras, proporcionados como datos de entrenamiento para un algoritmo de aprendizaje; mientras que los métodos no supervisados, suelen estar completamente automatizados y no requieren ninguna intervención humana o fase de entrenamiento.

**Evaluar la similitud semántica entre sentidos**: Los métodos para evaluar la similitud semántica entre conceptos (sentido de las palabras) en una KB (red semántica), con el fin de realizar emparejamiento de sentidos no supervisado y basado en el conocimiento, se puede clasificar como medidas:

* **basadas en bordes**, son los más intuitivos y estiman la similitud como el camino más corto (en bordes o número de nodos) entre los dos conceptos que se comparan.
* **basadas en nodos**, incorporan una fuente de conocimiento adicional: el análisis estadístico de corpus, para aumentar la información que ya está presente en la KB de referencia. Estiman la similitud como la cantidad máxima de contenido de información (es decir, una función de la probabilidad de ocurrencia del concepto, calculada en base a estadísticas de corpus y estructura de KB) que tienen conceptos en común.
* **basadas en definición**, evalúan la similitud semántica a medida que la palabra se superpone entre las glosas de conceptos (palabra sentidos) que se comparan, una glosa que subraya la definición textual describiendo el sentido de una palabra (Tekli, 2016).

### Enfoques

De acuerdo con los autores de varias bibliografías consultadas el número de enfoques para el WSD varía, siendo las más mencionadas:

**WSD supervisado**: este enfoque utiliza métodos de traducción automática para el aprendizaje de clasificación de la palabra objetiva a través del conjunto de datos, es decir, conjuntos de datos codificados como vectores donde los elementos representan características, con un elemento especial que representa la etiqueta (o clase) de sentido apropiada (Navigli, 2012). SupWSD es un sistema de última generación para WSD supervisado. Utiliza clasificadores lineales y una serie de características como etiquetas POS, palabras circundantes, colocaciones locales, incrustaciones de palabras y relaciones sintácticas (Logacheva et al., 2020)

Tabla 5: Técnicas/Algoritmos del enfoque supervisado para WSD, tomado de (Rahman & Borah, 2022) y otros

|  |  |
| --- | --- |
| **Técnicas/Algoritmos** | **Descripción** |
| Algoritmo del árbol de decisión/Desicion Tree | Se utiliza para indicar reglas de clasificación en una estructura de árbol donde el conjunto de datos de entrenamiento es recursivo. Cada nodo hoja denota el sentido de una palabra. El núcleo del árbol de decisión proporciona una pista de una prueba que tiene aplicación en la estimación de elementos del árbol. Las ramas del árbol indican el resultado de la prueba (Bhattacharjee et al., 2020) |
| Colocación basada en clases | Se utilizan tres puntuaciones diferentes de relación de palabras para la colocación: la primera son las relaciones entre nombres de WordNet; el segundo son las clases de similitud de palabras basadas en grupos; y el tercero es el análisis de definiciones de diccionario. |
| Red Neuronal Recurrente/ Recurrent Network Neuronal | Este modelo utiliza un enfoque conexionista en el que las neuronas artificiales se utilizan para el procesamiento de datos. Este algoritmo contribuye principalmente a colección de características de entrada, cuyo objetivo es dividir el conjunto de datos a entrenar en conjuntos de datos que no se superpongan (Bhattacharjee et al., 2020). Se basa en células de memoria a largo plazo (LSTM por sus siglas en ingles Long Short-Term Memory). Este LSTM ayuda a capturar información sobre el orden de las palabras y agrega representaciones de palabras distribuidas (incrustaciones) como características. |
| LSTMembed | Tiene como objetivo aprender incorporaciones de sentidos vinculados a BabelNet, al mismo tiempo el manejo del orden de palabras y uso de incrustaciones previamente entrenadas como objetivo. A pesar de ser probado sólo en inglés, el enfoque se puede adaptar fácilmente a otros idiomas presentes en Babelnet (Loureiro et al., 2021). |
| KNN | Tiene en cuenta los k vecinos más cercanos al nuevo valor a clasificar. Entre los sistemas que emplean esta técnica se encuentran Context2vec, ELMO, BERT-Base, BERT-Large, ALBERT-XXL (Loureiro et al., 2021). |
| BERT | La entrada al modelo consiste en un contexto (una oración que contiene una palabra ambigua) y una glosa (definición de sentido) de WordNet. El par glosa-contexto se concatena mediante un token especial y se clasifica como positivos o negativos. La incorporación de sentidos es una alternativa a modelos tradicionales de vectores de palabras como word2vec, fastText o GloVe, que representan bien palabras monosémicas pero falla en palabras ambiguas (Loureiro et al., 2021). |
| ALBERT | Incorpora dos técnicas de reducción de parámetros que eliminan los principales obstáculos en el escalado de modelos previamente entrenados. El primero es una parametrización de incrustación factorizada. La segunda técnica es compartir parámetros entre capas. Esta técnica evita que el parámetro crezca con la profundidad de la red. Ambas técnicas reducen significativamente el número de parámetros para BERT sin perjudicar gravemente el rendimiento, mejorando así la eficiencia de los parámetros (Lan et al., 2019). |
| RoBERTa | Es un algoritmo mejorado de BERT que propone el entrenamiento del modelo por más tiempo, con lotes más grandes, sobre más datos (Y. Liu et al., 2019). |
| DistilBERT | Se centra en reducir el tamaño de los grande modelos de lenguaje mediante el pre-entrenamiento y destilación de conocimientos (Sanh et al., 2020). |
| LMMS | genera incrustaciones de sentido con cobertura total de WordNet, al tiempo que renuncia a las alternativas de MFS. Posee incorporaciones sensoriales sin acceso al lema o a las funciones de parte del discurso que normalmente se utilizan para restringir los sentidos del candidato. Representar incorporaciones de sentido en el mismo espacio que cualquier incorporación contextual generada a partir del mismo NLM (Natural Language Modelling) previamente entrenado facilita las introspecciones de esos NLM y permite evaluaciones intrínsecas a nivel de token basadas en el rendimiento de WSD k-NN (Loureiro & Jorge, 2019). |
| BEM | Modelo bicodificador construido sobre BERT que está diseñado para mejorar el rendimiento de los sentidos raros y de disparo cero. El sistema representa las palabras objetivo y los sentidos en el mismo espacio de incrustación mediante el uso de un codificador de contexto para representar la palabra objetivo y el contexto circundante, y un codificador de brillo para representar las definiciones de sentido. |
| fastText | Es una extensión del modelo skipgram, donde las representaciones de palabras se aumentan usando ngramas de personajes. A cada ngrama de caracteres se le asocia una representación vectorial, y la representación vectorial de una palabra se obtiene tomando la suma de los vectores de la ngramas de caracteres que aparecen en la palabra. La palabra completa siempre se incluye como parte de los ngramas de caracteres, de modo que el modelo aún aprende un vector para cada palabra (Grave et al., 2018). |
| SMV (Support Vector Machine) | Los cálculos basados ​​en SVM se utilizan para caracterizar modelos en dos categorías diferentes. En el centro de las dos categorías, este cálculo encuentra un hiperplano, tal que el borde de separación entre estas dos categorías se vuelve peso máximo. El orden del modelo de prueba depende del borde del hiperplano, donde reside el modelo experimental (Bhattacharjee et al., 2020). |
| Naïve Bayes | Es un algoritmo clasificador basado en el Teorema de Bayes que ordena registros de contenido con dos limitaciones (Bhattacharjee et al., 2020):   * La probabilidad contingente de un sentido particular de un término y los aspectos más destacados de la circunstancia única. * El máximo de los valores determinados evaluados a partir de las ecuaciones representan la sentido contextual más pertinente. |
| GlossBERT | Logra una mejora significativa al aprovechar la información de definición. Este modelo se beneficia del enfoque de clasificación de pares de oraciones. |
| IMS | Era uno de los sistemas más frecuentes que entrenaba un sentido  clasificador para cada lema en los datos de entrenamiento. En comparación, unificó la desambiguación de palabras en una arquitectura de etiquetado de secuencia única, aliviando el problema de eficiencia. |
| LMMS | Implementa la desambiguación de una manera basada en similitudes. Aprende una incorporación de sentido para cada sentido etiquetado en SemCor y los asigna a una cobertura completa de los sentidos de WordNet utilizando relaciones de sentido (Wang & Wang, 2021). |

**WSD basado en conocimiento** (KB por sus siglas en ingles Knowledge Base): estos métodos explotan recursos de conocimiento (como diccionarios, tesauros, ontologías, etc.) para determinar en un contexto el sentido de las palabras. Cuentan con una cobertura más amplia como ventaja, gracias al uso de grandes cantidades de conocimiento estructurado. Los mejores sistemas basados ​​en el conocimiento de la literatura, como Degree o PageRank personalizado, utilizan WordNet u otros recursos (por ejemplo, BabelNet) para construir un gráfico semántico y explotar las propiedades estructurales del gráfico (ya sea localmente en la oración de entrada o globalmente) para elegir los sentidos apropiados de las palabras en contexto.

Tabla 6: Técnicas/Algoritmos del enfoque basado en conocimiento para WSD, tomado de (Rahman & Borah, 2022) y otros.

|  |  |
| --- | --- |
| **Técnicas o algoritmos** | **Descripción** |
| Lesk | Encuentra palabras similares entre definición de cada sentido con el contexto de la palabra objetivo |
| Lesk extendido | Se incluyen la definición de sentidos relacionados. Para la ponderación de palabras, se utiliza el término convencional frecuencia inversa del documento. Para un mejor análisis, se agrega incrustación de palabras en la versión mejorada de Lesk que ayuda a calcular la similitud entre la definición y el contexto de la palabra objetivo. |
| Basado en grafos | Se utiliza algoritmo de PageRank, que realiza paseos aleatorios por la red para identificar los nodos más importantes (Tripodi & Navigli, 2019) |
| Babelfy | Define una firma semántica para un contexto dado y lo compara con todos los sentidos candidatos para realizar la tarea de desambiguación, incluye todo el documento mientras encuentra su sentido (Tripodi & Navigli, 2019) |
| LSA | Cambia de un enfoque basado en superposiciones a un enfoque basado en vectores y analiza las relaciones semánticas distributivas entre un conjunto de documentos y términos en documentos, luego utiliza conceptos matemáticos de vectores para determinar la similitud como en Lesk mejorado (Orkphol & Yang, 2019) |
| UKB | Es uno de los sistemas relacionados que modelan relaciones sensoriales en contexto. Primero conecta los sentidos en contexto a través de relaciones de sentido de WordNet y opera PageRank personalizado en el gráfico de sentido construido para decidir la importancia del sentido. (Wang & Wang, 2021) |
| Heurísticas | El enfoque utiliza heurísticas, que se prueban para descubrir el significado de la palabra a partir de criterios semánticos específicos.  características. Hay tres tipos diferentes de heurísticas utilizadas mediante un punto de referencia para evaluar WSD marco (Bhattacharjee et al., 2020):   * Un Sentido para cada Discurso; donde una palabra conservará su importancia entre cada uno de sus acontecimientos en un discurso determinado. * Un Sentido para cada Aposición; donde se espera que las palabras más cercanas generen un signo sólido y predecible al contextual sentido de una expresión * Sentido de Frecuencia Máxima; esto funciona según el principio de descubrir todos los sentidos posibles que puede tener una palabra y es esencialmente pertinente que un sentido particular ocurra con más frecuencia que otros. |
| Word2vec | Es un enfoque popular de incrustación de palabras que representa cada palabra en el modelo de espacio vectorial de tamaño fijo a través del modelo skip-gram o de bolsa continua de palabras (CBOW) entrenado por una capa oculta Red neuronal artificial que captura eficazmente similitudes semánticas y sintácticas de palabras de un enorme corpus de texto (Orkphol & Yang, 2019) |

**WSD no supervisado**: son técnicas de inducción de sentidos de palabras destinadas a descubrir sentidos automáticamente basándose en corpus sin etiquetar. No explotan ningún corpus etiquetado manualmente para proporcionar una elección de sentido para una palabra en contexto (Navigli, 2012), (Al-Harbi et al., 2017). El enfoque no supervisado tiene la ventaja de que no necesita un gran corpus anotado y es más práctico en las aplicaciones del mundo real. Esto se puede hacer con la ayuda de un léxico público grande y bien organizado conocido como WordNet, que proporciona un conjunto de sinónimos cognitivos (o synsets) para cada palabra categorizada por etiqueta de parte del discurso donde cada synset viene con una definición (o glosa), ejemplos de uso y su relación con otras palabras. WordNet no proporciona directamente un conjunto de sentidos para cada palabra, sino un conjunto de conjuntos sintéticos donde cada conjunto sintético representa un concepto/sentido. (Orkphol & Yang, 2019)

Tabla 7: Técnicas/Algoritmos del enfoque no supervisado para WSD, tomado de (Rahman & Borah, 2022) y otros

|  |  |
| --- | --- |
| **Técnicas o algoritmos** | **Descripción** |
| Clustering | En la agrupación, la proposición fundamental denota tipos semánticos similares de información que se agrupan en un conjunto. Este es uno de los enfoques no supervisados ​​más destacados utilizado para WSD (Bhattacharjee et al., 2020). |
| Association rules | Las reglas de asociación se utilizan para establecer intrigantes  relación entre variables en grandes conjuntos de datos. Las reglas de asociación también se aplican en WSD (Bhattacharjee et al., 2020). |
| Word Sense Induction (WSI) | Tiene como objetivo agrupar los sentidos de las palabras y no requiere asignar cada grupo a un sentido predefinido, en cambio, los inventarios de sentido de las palabras se inducen automáticamente a partir de los grupos, tratando cada grupo como un sentido único de una palabra (Logacheva et al., 2020). Los enfoques de WSI caen en tres grupos principales:   * **agrupamiento de contexto** consisten en crear vectores que caracterizan los contextos de las palabras y agrupan estos vectores * **agrupación de redes de ego de palabras** se aplica a grafos semánticos. Los nodos de un gráfico semántico son palabras, y los bordes entre ellos denotan una relación semántica que es por lo general evaluado con similitud coseno de las incrustaciones correspondientes. * **sinónimos y los enfoques sustitutos de agrupación** crean vectores que representan sinónimos o sustitutos de palabras polisémicas. Dichos vectores se crean utilizando diccionarios de sinonimia o sustitutos dependientes del contexto obtenidos de un modelo de lenguaje |
| Algoritmo de inducción de sentido SenseGram | Separa a los vecinos más cercanos para inducir la palabra, percibe y construye incorporaciones sensoriales para cada sentido. Él comienza construyendo un ego-grafo (gráfico semántico centrado, en una palabra, en particular) de la palabra y sus vecinos más cercanos. Los bordes entre las palabras denotan su relación semántica, p.e. los dos nodos se unen con una arista si es coseno. El gráfico resultante se puede agrupar en subgrafos que corresponden a los sentidos de la palabra (Logacheva et al., 2020). |
| GPT | Utiliza un enfoque semi-supervisado para tareas de comprensión del lenguaje utilizando una combinación de pre-entrenamiento no supervisado y ajuste supervisado (Radford et al., 2018). |
| XLNet | Maximiza la probabilidad logarítmica esperada de una secuencia w.r.t. todas las permutaciones posibles del orden de factorización (Z. Yang et al., 2020) |
| Transformers-XL | Propone la noción de recurrencia en su red profunda de auto atención (Dai et al., 2019) |

Tras realizar el análisis de los diferentes enfoques que utiliza la técnica de WSD, estudiar sus métodos y consultar la disponibilidad de dichos algoritmos se ha optado por utilizar fasttext debido a la disponibilidad del modelo en español preentrenado donde otros modelos resulta tedioso buscar un modelo preentrenado para el español donde se le realizará las técnicas de Word Sense Disambiguation

Tabla 8: Comparativa de enfoques para la WSD, tomado de (Bhattacharjee et al., 2020), (*Aprendizaje supervisado vs. no supervisado; ¿cuál es mejor?*, s. f.), (*¿Qué es el aprendizaje no supervisado?*, s. f.)

|  |  |  |
| --- | --- | --- |
| **Enfoque** | **Ventajas** | **Desventajas** |
| KB | * Alta precisión * Útiles cuando se dispone de información específica sobre el dominio * Proporcionan explicaciones transparentes basadas en reglas | * Sufren de escasez de supervisión * Dependencia de la calidad y exhaustividad de las bases de conocimiento * Posibilidad de no adaptarse bien a contextos cambiantes o a términos poco comunes |
| Supervisado | * Mejor que ambos enfoques con aplicación * Utiliza conjuntos de datos etiquetados * Evitan la complejidad computacional al no requerir de grandes conjuntos de datos | * Resultados insatisfactorios * Tardan tiempo y requieren de experiencia humana para la validación de etiquetas de entradas y salidas de datos * Se necesitan grandes conjuntos de datos, lo que resulta costoso |
| No supervisado | * No hay ningún requisito de inventario sensorial y corpus anotados por sentido * Descubre patrones ocultos y relaciones entre datos * Útil para tareas como agrupamiento y reducción de dimensionalidad. | * Difícil de implementar y bajo rendimiento * En ocasiones produce resultados completamente erróneos si no es intervenido por humanos en la validación de los resultados * Mayor riesgo de resultados inexactos * Tiempos de entrenamiento más extensos |

## Herramientas y Tecnologías

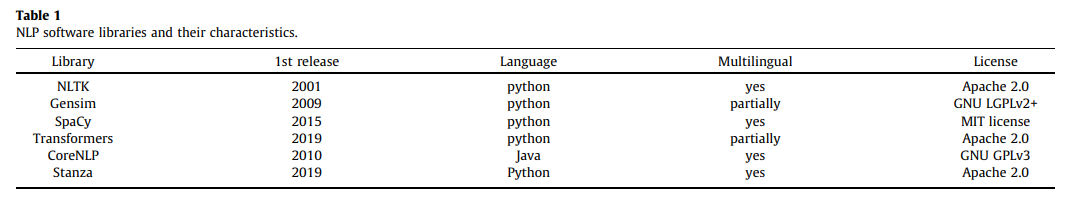
Python se ha convertido en el lenguaje de referencia para muchas aplicaciones de análisis, debido a su simplicidad, limpieza de sintaxis, múltiples bibliotecas de ciencia de datos y extensibilidad (específicamente con C++). Esta simplicidad y extensibilidad han llevado a que la mayoría de los principales marcos de aprendizaje profundo se construyan en Python o adopten interfaces de Python que incluyan extensiones optimizadas para GPU y C++ de alto rendimiento. (Kamath et al., 2019)

**Librerias (va en Herramientas y tecnologias)**

Pandas (https://pandas.pydata.org/) es una implementación popular de código abierto para estructuras y análisis de datos. Lo usaremos para la exploración de datos y algunos procesamientos básicos. • scikit-learn (http://scikit-learn.org/) es un popular código abierto para varios algoritmos y evaluaciones de aprendizaje automático. Lo usaremos solo para muestrear y crear conjuntos de datos, implementaciones de aprendizaje automático de algoritmos lineales y no lineales en nuestro caso de estudio. • Matplotlib (https://matplotlib.org/) es un popular código abierto para visualización. Lo usaremos para visualizar el rendimiento. (Kamath et al., 2019)

**Tools**

NLTK [57]3 (Natural Language Toolkit) es una plataforma líder para crear programas Python para trabajar con datos del lenguaje humano. Proporciona interfaces fáciles de usar para más de 50 corpus y recursos léxicos como WordNet, junto con un conjunto de bibliotecas de procesamiento de texto para clasificación, tokenización, derivación, etiquetado, análisis y razonamiento semántico, contenedores para bibliotecas de PNL de potencia industrial, y un foro de discusión activo. Gensim [58]4 es una biblioteca de Python para modelado de temas, indexación de documentos y recuperación de similitudes con grandes corpus. El público objetivo son las comunidades de PNL y recuperación de información (IR). La biblioteca contiene implementaciones eficientes de algoritmos populares, como el análisis semántico latente (LSA/LSI/SVD), la asignación latente de Dirichlet (LDA), las proyecciones aleatorias (RP) y el proceso jerárquico de Dirichlet (HDP). La biblioteca también incluye varios modelos previamente entrenados de Word2vec. SpaCy [59]5 es una biblioteca industrial para realizar tareas de PNL en Python. La biblioteca está diseñada específicamente para construir sistemas industriales complejos e interopera a la perfección con TensorFlow, PyTorch, scikit-learn, Gensim y el resto del ecosistema de inteligencia artificial de Python. Spacy incluye varias funcionalidades, como tokenización, NER, segmentación de oraciones, etiquetado PoS y análisis de dependencias. La biblioteca también contiene varios vectores de palabras previamente entrenados. Transformers [60]6 proporciona arquitecturas de propósito general (BERT, GPT-2, RoBERTa, XLNet...) para la comprensión del lenguaje natural (NLU) y la generación del lenguaje natural (NLG) con más de 32 modelos previamente entrenados en más de 100 idiomas y profundidad. interoperabilidad entre TensorFlow 2.0 y PyTorch. La biblioteca está curada por Huggingface, una startup centrada en PNL con una gran comunidad de código abierto. La biblioteca expone las API para utilizar muchas arquitecturas de transformadores previamente entrenados y conocidas que se describen en las secciones anteriores. CoreNLP [61]7 es una biblioteca de PNL que abarca los principales módulos y tareas de PNL, es decir, tokenización, etiquetado PoS, análisis, NER, para el lenguaje de programación Java. La biblioteca ofrece soporte para 6 idiomas diferentes. Stanza [62]8 es una colección de herramientas precisas y eficientes para muchos lenguajes humanos. Stanza incluye una interfaz Python para el paquete CoreNLP Java. El conjunto de herramientas está diseñado para ser paralelo entre más de 70 idiomas, utilizando el formalismo de Dependencias Universales. Finalmente, el paquete proporciona herramientas y modelos especializados para tareas clínicas y biomédicas.



Frameworks (va en Herramientas y Tecnologias)

Los marcos de aprendizaje profundo más populares son: TensorFlow, PyTorch, Keras, MXNet, CNTK, Chainer, Caffe2, PaddlePaddle y Matlab. El componente principal de un marco moderno de aprendizaje profundo es la eficiencia en las capacidades de álgebra lineal, ya que esto se aplica al aprendizaje profundo, con soporte para el cálculo de CPU y GPU (el hardware dedicado como los TPU [Jou16] también se está volviendo cada vez más popular).

TensorFlow: TensorFlow es una biblioteca computacional basada en gráficos de flujo de datos. Estos gráficos tienen nodos que representan operaciones matemáticas y aristas que representan tensores que fluyen entre ellos. Escrito en Python, TensorFlow fue desarrollado por el equipo de Google Brain.

Keras: Keras es una biblioteca Python simple y de alto nivel desarrollada para permitir la creación rápida de prototipos y la experimentación. Puede ejecutarse sobre TensorFlow y CNTK, y ahora forma parte de la biblioteca principal de TensorFlow. Keras contiene implementaciones de componentes comunes de redes neuronales y numerosos ejemplos de arquitectura.

PyTorch: PyTorch es un paquete de Python para la creación rápida de prototipos de redes neuronales. Se basa en Torch, un marco computacional extremadamente rápido, y proporciona cálculo de gráficos dinámicos en tiempo de ejecución. PyTorch fue desarrollado por el equipo de investigación de inteligencia artificial de Facebook.

Caffe: Caffe es un marco C++ de alto rendimiento para crear arquitecturas de aprendizaje profundo que pueden admitir de forma nativa la ejecución distribuida y de múltiples GPU. La versión actual, Caffe2, es el backend utilizado por Facebook en producción.CNTK: renamed the Microsoft Cognitive Toolkit, CNTK is a computational framework based on directed graphs. It supports the Python, C#, or C++ languages, and was developed by Microsoft Research.

MXNet: MXNet es un marco computacional de alto rendimiento escrito en C++ con soporte nativo de GPU incubado por el Proyecto Apache.

Chainer: Chainer es un marco basado puramente en Python con capacidad de gráficos computacionales dinámicos definidos en tiempo de ejecución.

PLN

Stanford CoreNLP: un conjunto de herramientas de análisis lingüístico basado en Java para procesar texto en lenguaje natural. CoreNLP fue desarrollado por la Universidad de Stanford.

NLTK: Natural Language Toolkit, o NLTK para abreviar, es un conjunto de bibliotecas de código abierto para el procesamiento simbólico y estadístico del lenguaje natural para inglés. Fue desarrollado por la Universidad de Pensilvania.

Gensim: un conjunto de herramientas de código abierto basado en Python que se centra en el espacio vectorial y el modelado temático de documentos de texto.

spaCy: un conjunto de herramientas de alto rendimiento basado en Python para el procesamiento avanzado del lenguaje natural. SpaCy es de código abierto y está respaldado por Explosion AI.

OpenNLP: un conjunto de herramientas de aprendizaje automático de código abierto para procesar texto en lenguaje natural. OpenNLP está patrocinado por el Proyecto Apache. •

AllenNLP: una biblioteca de investigación de PNL construida en PyTorch.

**Resources**

Natural Language Processing with Deep Learning http://web.stanford.edu/class/cs224n/

Deep Learning for Natural Language Processing <http://www.cs.ox.ac.uk/teaching/courses/2016-2017/dl>

Neural Networks for NLP <http://phontron.com/class/nn4nlp2017/schedule.html>

Deep Learning Specialization <https://www.deeplearning.ai/deep-learning-specialization/>

Deep Learning Summer School <https://vectorinstitute.ai/2018/11/07/vector-institute-deep-learning-andreinforcement-learning-2018-summer-school/>

Convolutional Neural Networks for Visual Recognition <http://cs231n.stanford.edu>

Neural Networks for Machine Learning http://www.cs.toronto.edu/∼hinton/coursera lectures.html • Neural Networks http://info.usherbrooke.ca/hlarochelle/neural networks/content.html • Practical Deep Learning For Coders https://course.fast.ai/ • Intro to Deep Learning with PyTorch <https://www.udacity.com/course/deep-learning-pytorch--ud188>

**Librerias (va en Herramientas y tecnologias)**

Pandas (https://pandas.pydata.org/) is a popular open source implementation for data structures and data analysis. We will use it for data exploration and some basic processing. • scikit-learn (http://scikit-learn.org/) is a popular open source for various machine learning algorithms and evaluations. We will use it only for sampling and creating datasets, machine learning implementations of linear and non-linear algorithms in our case study. • Matplotlib (https://matplotlib.org/) is a popular open source for visualization. We will use it to visualize performance

(Kamath et al., 2019)

# Capítulo II. Análisis y diseño de la propuesta solución

## Introducción

En el presente capítulo se describen los principales elementos que se tendrán en cuenta en el desarrollo de la solución propuesta. El uso de XP como metodología organizará el proceso de trabajo, en dependencia de las fases y artefactos que define XP, entre los que se encuentran las HU y estimación por esfuerzo. En este capítulo quedan definidos los patrones de diseño, la arquitectura, la base de datos y los algoritmos que permitirán a la herramienta detectar y analizar las ambigüedades.

### 2.1 Descripción general de SIDARES

La herramienta informática propuesta permite identificar la existencia de términos ambiguos en requisitos de software. Los requisitos analizados por esta, pueden ser redactados de forma manual o importados desde un documento con extensión (.xlsx,.csv). La herramienta una vez que obtiene y analiza los requisitos, es capaz de mostrar el requisito con la ambigüedad que detectó ya sea léxica o sintáctica, con el propósito de que la palabra o frase ambigua pueda ser corregida por el ingeniero de requisitos. A continuación, la Figura 5 muestra la interfaz de la herramienta propuesta.

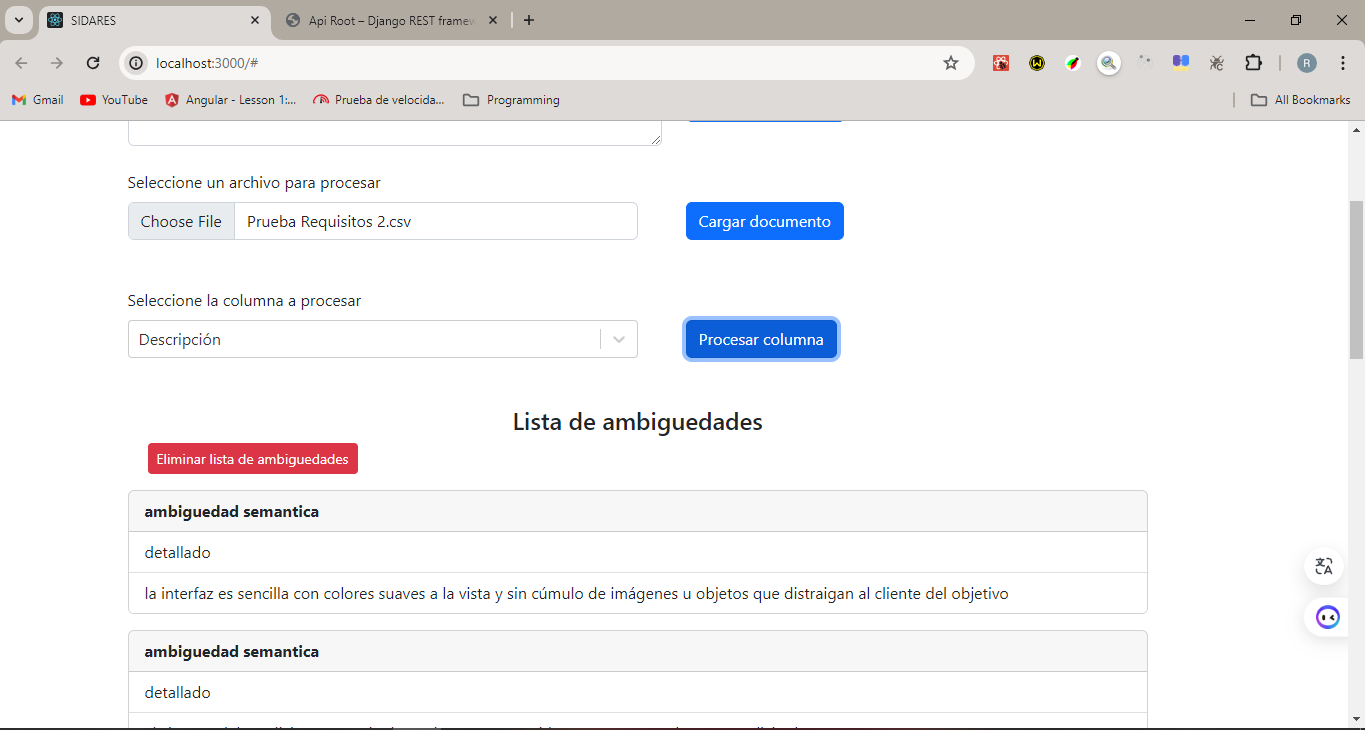


Figura 5: SIDARES (Sistema de Detección de Ambigüedades en Requisitos de Software)

La interfaz de la herramienta está compuesta por un conjunto de componentes visuales. En la parte superior izquierda se encuentra el nombre del sistema. SIDARES, cuenta con los botones “Procesar requisito” y “Procesar columna “, los cuales son los encargados de analizar el requisito y documento de requisitos para así detectar ambigüedades léxicas, sintácticas o semanticas. El botón “Cargar documento” es el que permite importar el documento de requisito para ser posteriormente analizado. En la parte inferior del contenedor principal se encuentra una ventana que muestra los requisitos con las ambigüedades detectadas, además del botón “Eliminar lista de ambigüedades” para vaciar este campo si así lo desea el usuario.

### 2.2 Definición de un conjunto de datos para procesamiento de requisitos en lenguaje natural

Como parte de la propuesta solución se encuentra la elaboración de un conjunto de datos de requisitos de software que servirá como caso de estudio de la herramienta. Para la confección del conjunto de datos, se recopilaron los documentos que contenían los requisitos de software de los diferentes proyectos de cada centro de la UCI, siendo esta la primera etapa del algoritmo Kdd definido en el capítulo 1. Se realizaron dos etapas para el desarrollo del conjunto de datos.

La primera etapa del trabajo consistió en identificar los documentos de requisitos disponibles en la universidad. Se recopilaron de cada uno de los centros de desarrollo de software, los documentos que recogían requisitos funcionales y no funcionales, lo cual permitió recolectar un total de 14 documentos. Todos ellos estaban en un formato uniforme .xlsx y presentaban un contenido estructurado con una distribución de campos donde se destacaban: ‘tipo de requisito’, ‘centro ’y ‘descripción del requisito’.

En una segunda etapa fueron sometidos a un proceso ETL (Extracción-TransformaciónLimpieza) fases o etapas de Kdd. En esta etapa se estructuraron uniformemente y eliminaron ruidos e impurezas que pudieran afectar el futuro análisis del contenido de estos. El resultado fue almacenado en un documento único de formato .csv el cual quedó conformado por un total de 19 357 requisitos entre los que se encuentran 19 139 requisitos funcionales y 216 no funcionales descritos en lenguaje natural. La Figura 6 muestra la estructura del conjunto de datos de requisitos.

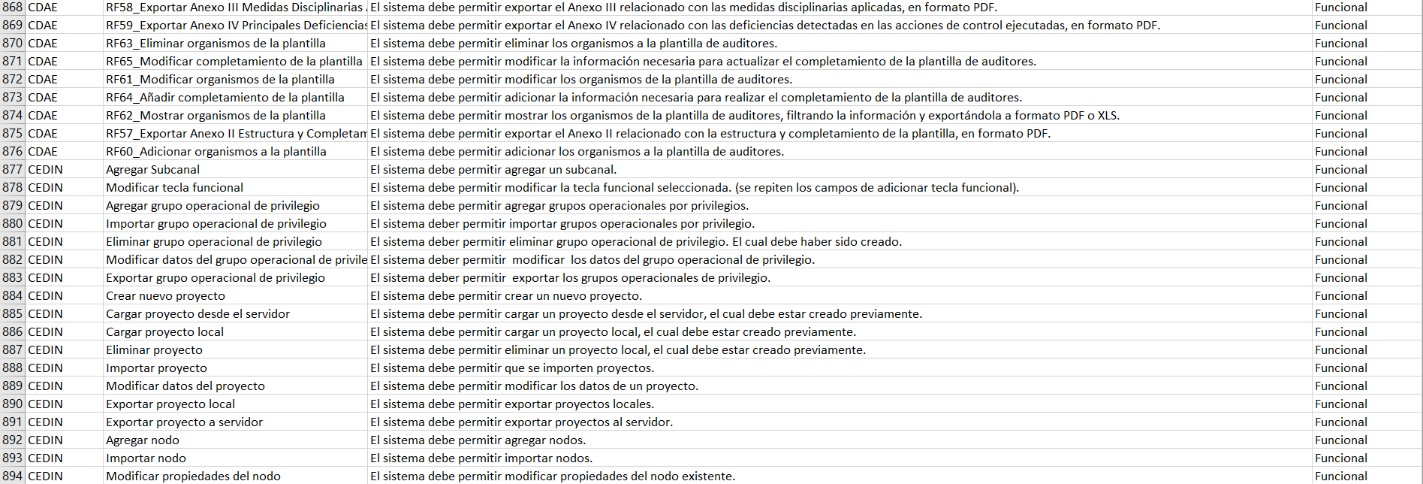


Figura 6: Conjunto de datos de requisitos

El conjunto de datos de requisito está estructurado por cuatro columnas, que contienen la siguiente información: centro, nombre del requisito, descripción y tipo.

### 2.3 Proceso de desarrollo

En el capítulo 1 quedó seleccionado el uso de la metodología XP para guiar el proceso de desarrollo de software de la presente investigación. A continuación, se describe como fueron llevadas a cabo las fases de planificación y diseño. Las fases de prueba y codificación se desarrollarán en el próximo capítulo.

#### 2.3.1 Planificación

En esta fase se describen los requisitos de la herramienta haciendo uso de las HU y se realiza una estimación del esfuerzo necesario en cada una de ellas. Los encargados de dar solución a la problemática se familiarizan con las tecnologías y herramientas que se utilizarán.

## Requisitos de la solución

Los requisitos describen las características de una solución que cumpla con necesidades de las partes interesadas. En reiteradas ocasiones se dividen en requisitos funcionales y no funcionales o de calidad, especialmente cuando los requisitos describen una solución de desarrollo de software.

Para dar respuesta a la problemática se obtuvieron los siguientes **requisitos funcionales** (RF):

**RF1. Importar documento:** el sistema debe permitir importar documento de requisitos de software.

**RF2. Mostrar datos del documento:** el sistema debe mostrar los datos (título, extensión) del documento importado.

**RF3. Redactar requisito:** el sistema debe permitir redactar requisitos de software. **RF4. Mostrar tipo de ambigüedad:** el sistema debe mostrar el tipo de ambigüedad que presente el requisito de software.

**RF5. Detectar ambigüedad:** el sistema debe permitir detectar ambigüedades (léxicas, sintácticas).

**RF6. Eliminar ambigüedad:** el sistema debe permitir eliminar la lista de ambigüedades detectadas en los requisitos.

**RF7. Seleccionar columna:** el sistema debe permitir seleccionar la columna del documento analizar.

Para la investigación se proponen los siguientes **requisitos de calidad** (RC):

## Requisitos de apariencia o interfaz externa

**RC 1:** La herramienta debe contar con una interfaz sugerente que facilite la navegación por ella, es decir que cada opcionalidad le sugiera al usuario la funcionalidad que realiza.

**RC 2:** La letra debe ser visible y legible.

## Requisitos de software

**RC 1:** Para el uso de la herramienta la PC debe contar con cualquier navegador web. **RC 2:** Para el funcionamiento de la herramienta se debe disponer de PostgreSQL 9.2 o superior.

## Requisitos de hardware

**RC 1:** Para el correcto funcionamiento de la herramienta la PC debe contar con 512 MB de memoria RAM.

**RC 2:** Para el correcto funcionamiento de la herramienta la PC debe contar con un procesador de 2.0 GHz o superior.

## Requisito de seguridad

**RC 1:** Si ocurre alguna excepción la herramienta mostrará un mensaje de alerta.

Los requisitos expuestos en el presente epígrafe fueron analizados manualmente, a partir de los criterios de calidad que brida la ISO/IEC/IEEE 29148 y los criterios del lenguaje que propone (Ferrari et al. 2019). Este previo análisis aseguró la calidad de los requisitos y permitió definir requisitos que no poseen términos negativos, ambiguos y vagos.

## Historias de Usuarios

Las HU conforman la parte central de muchas metodologías de desarrollo ágil entre ellas XP.

Para el desarrollo del presente trabajo, se identificaron un total de 7 HU, a continuación, se muestran algunas de ellas, el resto se encuentra en el Anexo 5 del documento:

Tabla 8: HU. Importar documento de requisitos

|  |  |  |
| --- | --- | --- |
| **Historia de usuario** | | |
| Número: 1 | | Nombre de la Historia de Usuario: Importar documento |
| Modificación de la Historia de Usuario: ninguna | | |
| Usuario: Samira Enríquez González | | |
| Prioridad en negocio: alta | Riesgo en desarrollo: medio | |
| Puntos estimados: 5 días | Iteración asignada: 1 | |
| Descripción: El usuario podrá importar un documento con requisitos de software. | | |
| Observaciones:   1. Solo admite documentos con las extensiones (.csv, .xlsx), en caso de cargar un documento con otro formato el sistema debe mostrar una notificación (“formato incorrecto”) 2. En caso de no seleccionar un documento y presionar el botón “Cargar documento” el sistema debe mostrar una notificación (“seleccione un documento”) | | |

Tabla 9: HU. Detectar ambigüedades léxicas ,sintáctica o semántica

|  |  |  |
| --- | --- | --- |
| **Historia de usuario** | | |
| Número: 5 | | Nombre de la Historia de Usuario: Detectar ambigüedades léxicas ,sintáctica o semántica. |
| Modificación de la Historia de Usuario: ninguna | | |
| Usuario: Jonathan Ramírez Reyes | | |
| Prioridad en negocio: alta | Riesgo en desarrollo: alto | |
| Puntos estimados: 10 días | Iteración asignada: 2 | |
| Descripción: El usuario podrá detectar las ambigüedades léxicas y sintácticas presentes en el documento o el requisito redactado, haciendo uso de las técnicas de PLN. | | |
| Observaciones: Debe haber requisitos cargados o escritos para ser analizados. | | |

## Estimación de esfuerzo por historias de usuario

Para el desarrollo de la herramienta propuesta se efectuó una estimación de esfuerzo haciendo uso de la técnica de punto de HU, obteniéndose los siguientes resultados.

Tabla 10: Estimación de esfuerzo por historias de usuario

|  |  |
| --- | --- |
| **Historias de usuario** | **Puntos de estimación (días)** |
| Importar documento | 5 |
| Mostrar datos del documento importado | 5 |
| Detectar ambigüedades léxicas , sintácticas y semánticas | 10 |
| Mostrar tipo de ambigüedad | 3 |
| Redactar requisito | 3 |
| Eliminar ambigüedades | 3 |
| Seleccionar columna analizar | 3 |

La estimación de desarrollo de esta herramienta arrojó como resultado que la implementación debía durar un tiempo total de 32 días aproximadamente 8 semanas, donde 5 días equivale a una semana y 40 horas de trabajo.

## Plan de iteraciones

El plan de iteraciones constituye varias iteraciones sobre la aplicación antes de su entrega. En este plan queda definida la HU que será implementada en cada iteración. A continuación, se describen cada una de las iteraciones propuestas, donde la duración total de iteraciones en días, se obtiene a partir del esfuerzo en días estimados para implementar cada HU.

* Iteración 1: en esta iteración se implementó las HU 1, 2, 3, 6 y 7 las cuales tienen relación a todo lo referido a los requisitos de software tanto redactados como del documento importado.
* Iteración 2: en esta iteración se implementó las HU 4 y 5 las cuales tienen en cuenta las funcionalidades de detectar, mostrar y señalar ambigüedades.

La tabla 11 muestra la duración de cada iteración, así como el orden en que serán implementadas las HU en cada una de ellas según la prioridad asignada.

Tabla 11:Plan de iteraciones

|  |  |  |  |
| --- | --- | --- | --- |
| **Iteraciones** | **Historias de usuario** | **Duración** | **Prioridad** |
| 1 | Importar documento | 5 días | Alta |
| Mostrar datos del documento importado | 5 días | Alta |
|  | Redactar requisito | 3 días | Alta |
| Eliminar ambigüedades | 3 días | Media |
|  | Seleccionar columna analizar | 3 días | Media |
| 2 | Detectar ambigüedades léxicas y sintácticas | 10 días | Alta |
| Mostrar tipo de ambigüedad | 3 días | Alta |

## Plan de entregas

Luego de quedar establecido el plan de iteración, se procede a la elaboración del plan de entrega, con la intención de obtener una estimación realista del tiempo que tomaría la implementación de las funcionalidades que describe cada HU. En la tabla 12 se muestra el plan de entrega.

Tabla 12:Plan de entrega

|  |  |  |
| --- | --- | --- |
| **Producto** | **Iteración 1** | **Iteración 2** |
| Herramienta para analizar ambigüedad en requisitos de software | versión 1.0 del producto | versión 1 del producto |
| Fecha de inicio | 18/6/2022 | 30/6/2022 |
| Fecha de entrega | 29/6/2022 | 22/7/2022 |

### 2.3.2 Diseño

La metodología de desarrollo de software XP propone el diseño de la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto. Los principales elementos del diseño de software que propone XP como artefactos se presentan a continuación.

## Arquitectura de Software

La arquitectura de software es una representación que permite analizar y evaluar la efectividad del diseño para cumplir con los requisitos establecidos, considerar alternativas arquitectónicas en una etapa temprana posibilita hacer cambios al diseño de forma fácil y sencilla, además reduce los riesgos asociados con la elaboración del software.

Para el desarrollo de la presente investigación se utilizó el patrón arquitectónico Modelo Vista Plantilla (MVT, por sus siglas en inglés) que ofrece Django. El empleo de este patrón se debe a que el framework utilizado para el desarrollo de Backend [[3]](#footnote-2) es Django,específicamente el marco de trabajo de Django Rest Framework para la confección de la API REST, además que el lenguaje definido para la construcción de la herramienta es Python.

MVT es un patrón de diseño o arquitectura de diseño que sigue Django para desarrollar aplicaciones web. Es ligeramente diferente del patrón de diseño Modelo-VistaControlador, comúnmente conocido(*Antonio 2020) (Edgardo 2020)*. Este patrón se conforma por tres componentes el modelo, la vista y la plantilla. El modelo es el cual gestiona los datos y está representado por una base de datos. La vista es la que se encarga de recibir solicitudes y envía respuestas. Además, la vista interactúa con el modelo y una plantilla para completar una respuesta, donde la plantilla es básicamente la capa frontal y el componente HTML dinámico de la aplicación. En la Figura 7 se encuentran representados los componentes del patrón arquitectónico a utilizar.

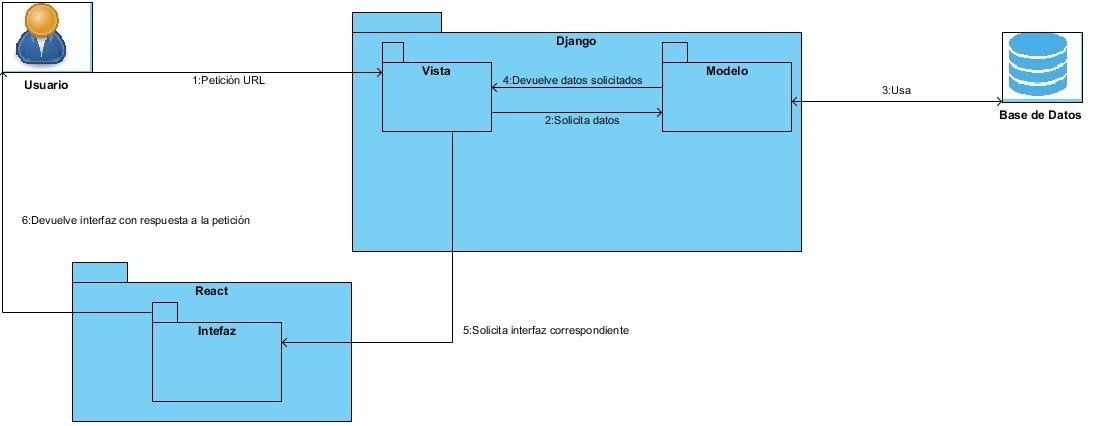


Figura 7: Arquitectura MVP. Elaboración propia

El paquete Modelo contiene la estructura de los datos como el tipo de ambigüedad y su descripción; a su vez utiliza la base de datos para almacenarlos. En el paquete Vista se encuentra las clases principales como AmbigüedadView y FicheroView. Para crear las interfaces con la que interactúa el usuario se utilizó React.

## Patrones de diseño

Un patrón de diseño es una solución a un problema en un contexto particular. Además, facilitan la reutilización de diseños o arquitecturas de software que han tenido éxito y describen el esquema básico para estructurar subsistemas y componentes *(Macario 2018)*. Para el diseño de la solución se emplearon varios patrones de diseño los cuales se describe a continuación.

## Patrones GRASP

Los patrones GRASP (General Responsibility Assignment Software Patterns) describen los principios fundamentales para asignar responsabilidades a los objetos *(Macario 2018)*.

Dentro de los patrones GRASP a utilizar se encuentran:

**Experto:** este patrón ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. El patrón está presente en la clase AmbiguedadesView que contienen todos los métodos e información para tener poca dependencia del resto de las clases.

**Creador:** se encuentra en las clases que tienen la tarea de instanciar objetos de otras. El empleo de este patrón se evidencia en la clase AmbiguedadesView la cual se encarga de crear objetos y almacenarlo.

**Controlador:** este se encarga de coordinar el trabajo de otros. Este patrón se evidencia en la clase Main que brinda Django.

**Alta cohesión:** se utiliza para realizar un diseño que evite la sobrecarga de métodos. La clase tiene alta cohesión si todo lo que hace está bien delimitado dentro del mismo objeto.

El patrón se evidencia en cada de una de las clases de la herramienta.

**Bajo acoplamiento:** es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que, en caso de producirse una modificación en alguna de ellas se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases este patrón se evidencia en cada de una de las clases de la herramienta.

## Patrones GOF

Dentro de los patrones GOF (Gang of four) se encuentran tres grupos de patrones los de creación estructurales y de comportamiento. Se utilizará el siguiente patrón para el diseño de la solución.

**Singleton**: es un patrón de diseño de creación que permite asegurar que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.*(Alexander 2021)*. Este patrón se evidencia en la clase Main de Django.

## Descripción de las tarjetas CRC

Las tarjetas CRC (clase, responsabilidad y colaboración) son una metodología para el diseño de software orientado por objetos creada por Kent Beck y Ward Cunningham.Permite romper con el modo de procedimiento y de pensamiento para apreciar mejor la tecnología de objetos. También permiten que todo el equipo pueda contribuir al diseño del proyecto *(Alejandra 2022)* . A continuación, se describen algunas de las tarjetas, el resto se encuentra en el Anexo 6.

Tabla 13: Tarjeta CRC AmbiguedadesView

|  |  |
| --- | --- |
| **Tarjeta CRC** |  |
| **Clase:** AmbiguedadesView |  |
| **Responsabilidades** | **Colaboración** |
| Detectar ambigüedades léxicas, semánticas y sintácticas a partir de un texto recibido. | Detectar ambigüedades léxicas.  Detectar ambigüedades semánticas.  Detectar ambigüedades sintácticas. |

Tabla 14: Tarjeta CRC AmbiguedadesFicheroView

|  |  |
| --- | --- |
| **Tarjeta CRC** |  |
| **Clase: AmbiguedadesFicheroView** |  |
| **Responsabilidades** | **Colaboración** |
| Detectar ambigüedades léxicas, semánticas y sintácticas a partir de un documento cargado. | Detectar ambigüedades léxicas.  Detectar ambigüedades semánticas.  Detectar ambigüedades sintácticas. |

## Estándares de codificación

Los estándares de código, son parte de las llamadas buenas prácticas. Estas son un conjunto no formal de reglas, que han ido surgiendo en las distintas comunidades de desarrolladores con el paso del tiempo y las cuales, bien aplicadas pueden incrementar notablemente, la calidad del código.*(Arthur 2020)*. El estándar de codificación para el desarrollo de la herramienta definido luego de un estudio realizado previamente es el **CamelCase**. Se decidió el uso de este estándar debido a que reduce el esfuerzo necesario para leer y entender el código fuente, además de que mejora la apariencia de este al no permitir abreviaturas*.(Keefe 2022)*

**CamelCase** es un estilo de escritura que se aplica a frases o palabras compuestas. El nombre se debe a que las mayúsculas a lo largo de una palabra se asemejan a las jorobas de un camello cuando la primera letra de cada una de las palabras es mayúscula.*(Keefe 2022).* Existen dos tipos de CamelCase el UpperCamelCase, cuando la primera letra de cada una de las palabras es mayúscula y el lowerCamelCase, igual que la anterior con la excepción de que la primera letra es minúscula. A continuación, algunas pautas utilizadas del estándar anteriormente mencionado.

* Los nombres de las clases serán en mayúscula, en caso de ser un nombre compuesto las siguientes palabras se escribirán de igual forma.
* Los nombres de los métodos serán con mayúscula, en caso de ser un nombre compuesto, la primera palabra será en minúscula y la siguiente en mayúscula.
* Los identificadores para las variables y los parámetros serán en minúsculas.
* Los nombres de variables o funciones deben ser lo suficientemente descriptivos no más de 15 caracteres.
* De haber comentarios serán en idioma español para una mejor comprensión.

### 2.4 Procesamiento del lenguaje natural

En este trabajo se aplicó una línea de procesamiento del lenguaje natural compuesta por tres técnicas de PLN: 1-) tokenización para dividir el texto en tokens, 2-) eliminación de ruido como, caracteres no deseados o espacios en blanco y números, 3-) clasificación para asignar una etiqueta por parte del discurso (POS), por ejemplo, sustantivo, verbo o pronombre, a cada token de cada frase. La Figura 8 muestra como ocurre el preprocesamiento haciendo uso de las técnicas mencionadas. Como entrada se tiene al documento de requisitos o al requisito escrito manualmente. Luego se pasa a la fase de procesamiento del contenido textual donde se aplican las técnicas de PLN que se muestran en la Figura 8 y definidas en el capítulo 1. Una vez terminado el proceso de PLN se determina que el requisito presenta ambigüedad.

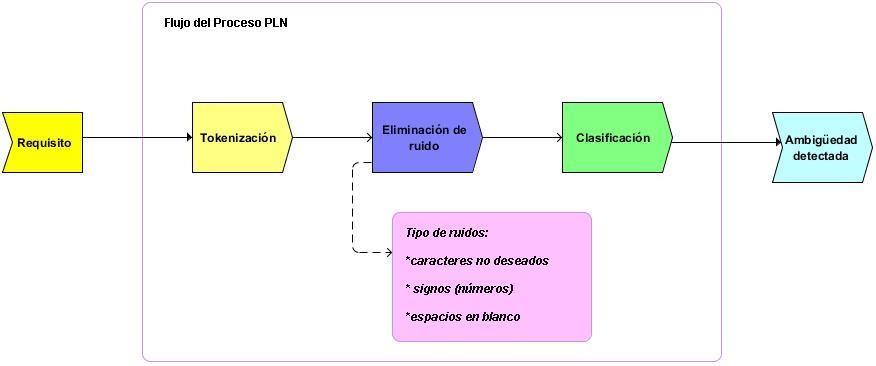


Figura 8: Flujo del proceso PLN

### 2.5 Algoritmos para la detección de ambigüedades

Para la detección de ambigüedad léxica y sintáctica en los requisitos se implementaron 2 algoritmos basados en reglas heurísticas y para la detección de ambigüedad semántica se utilizo un algoritmo de desambiguación del sentido de la palabra (Word Sense Disambiguation) el modelo de calidad obtenido luego del estudio realizado en el capítulo 1. A continuación se presentan las reglas y algoritmos obtenidos.

## Detección de ambigüedad léxica

Para comprobar la presencia de ambigüedad léxica en un requisito, se defino una regla. Sean A conjuntos de términos ambiguos porque tienen más de una interpretación, S un requisito, y sea T (S) cualquier secuencia de palabras del requisito. La regla es la siguiente:

**R1:** ∀a ∈ A, ∀t ∈ T (S), si t = a, marca t como ambigüedad léxica.

Si un requisito tiene asociado un término con más de un significado como el patrón anterior entonces el requisito está en presencia de ambigüedad léxica.

A continuación, el código para la detección de ambigüedad léxica.

|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | Entrada requisito redactado o documento de requisitos | |

1. def AddLexicaAmbiguity **(text)**
2. lexicals = [ ]

|  |
| --- |
| Recorrer lista de palabras ambiguas |

1. for x in ListaPalabrasAmbiguas:

|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | Si la palabra(s) ambigua(s) pertenece a el requisito o documento de requisitos | |

1. if x in text:

1. lexical.append(“Ambiguedad lexica detectada en la cadena " + " " + "( " + x + " )")

|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | Muestra si detecto o no ambigüedad | |

1. return np.unique(lexicals)

## Detección de ambigüedad sintáctica

Para comprobar la presencia de ambigüedad sintáctica en un requisito, se definieron las siguientes reglas.

**R1:** (Token)∗ (CC ) (Token.kind != "punct")∗ (CC) (Token)∗

**R2:** (Token)∗ (CD ) (Token.kind != "punct")∗ (CD) (Token)∗

**R3:** (Token)∗ (P ) (Token.kind != "punct")∗ (P) (Token)∗

**R4:** (JJ) (NN | NNS) (y | o) (NN | NNS)

La regla 1 busca al menos dos apariciones de conjunciones copulativas (y, e, ni, que) en el requisito. Estas conjunciones no deben estar separadas por signos de puntuación (por ejemplo, comas, punto y coma, guiones, etc.). Las comas y otros tipos de puntuación pueden aclarar la estructura sintáctica. La regla 2 por su parte busca la presencia de al menos dos conjunciones disyuntivas (o, u, sea, bien), aunque el funcionamiento es similar a la regla 1. Si un requisito contiene al menos una preposición separable (a, con, de, en) entonces estamos en presencia de la regla 3. Si un adjetivo (JJ) precede a un par de sustantivos singulares (NN) o plurales (NNS), unidos por "y" o "o" estamos en presencia de la regla 4 una ambigüedad sintáctica coordinativa.

El código para detectar ambigüedad sintáctica se presenta a continuación.

|  |
| --- |
| Entrada requisito redactado o documento de requisitos |

**1.**def SyntacticAnalissys(**text**):

**2.**sintactic = []

|  |  |  |
| --- | --- | --- |
| Conjunciones separables | y | preposiciones  61 |

**3.** c = [' y ', ' e ' , ' ni ', ' que ']

cp = [' o ', ' u ', ' sea ', ' bien '] cps = [' a ', ' con ', ' de ', ' en ' ]

**4.**cont = 0

cont2 = 0 Contadores

cont3 = 0

|  |
| --- |
| Tokenizar las palabras |

1. palabras = word\_tokenize(text) print(palabras)

|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | Dividir los requisitos en oraciones | |
| |  | | --- | | Buscar en las oraciones si aparece más de una conjunción copulativa | |

1. for sentence in blob.sentences: for j in sentence: if j == '.' or j == ',' or j == ';':

cont = 0

1. for y in c:

if y in sentence: cont = cont + 1 if cont >= 2: sintactic.append(

'posible ambigüedad sintáctica: El requisito presenta más de una conjunción copulativa

(y, e, ni, que) ')

|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | Buscar en las oraciones si aparece más de una conjunción disyuntiva | |

1. for sentence in blob.sentences: for r in cp: if r in sentence:

cont2 = cont2 + 1 if cont2 >= 2:

sintactic.append(

'posible ambigüedad sintáctica: El requisito presenta más de una conjunción disyuntiva (o, u, sea, bien) ')

|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | Buscar en las oraciones si aparece más de una preposición separable | |

1. for sentence in blob.sentences: for z in cps: if z in sentence:

cont3 = cont3 + 1 if cont3 >= 2:

sintactic.append( 'posible ambigüedad sintáctica: El requisito presenta más de una preposición separable ( a, con, de, en) ')

|  |
| --- |
| Muestra si detecto o no ambigüedad sintáctica |

1. return sintactic

## Detección de ambigüedad semántica

Para comprobar la presencia de ambigüedad semántica en un requisito,se utilizo un algoritmo de desambiguación del sentido de la palabra y un diccionario que contiene varias palabras ambiguas.

|  |
| --- |
| Entrada requisito redactado o documento de requisitos y las posibles palabras que presentan ambigüedades semánticas denominadas target\_words |

El código para detectar ambigüedad semántica se presenta a continuación.

1. def wsd(text, target\_words):

|  |
| --- |
| Se convierte a minúscula el contenido de text y cada vez que exista un espacio se le asignará una posición distinta en la lista. |

1. words = text.lower().split()

3.model = fasttext.load\_model("/content/cc.es.300.bin")

ambiguous\_words = []

|  |
| --- |
| Se inicializa una lista vacía para usarla posteriormente y se carga el modelo de fasttext |

4 . for target\_word in target\_words:

     target\_indices = [i for i, word in enumerate(words) if word ==target\_word.lower()]

if not target\_indices:

            continue

|  |
| --- |
| Busca la presencia de las target\_word o palabras ambiguas del diccionario, en el texto pasado por parámetro, si existe algunas de las palabras en la lista de target\_indices le asigna el indice donde se encontró la palabra y en caso de que no la encuentre continua. |

5. word\_vectors = [model.get\_word\_vector(word) for word in words]

|  |
| --- |
| A partir del modelo pre-entrenado ,mediante iteraciones se vectoriza cada palabra con la función get\_word\_vector de fasttext. |

6. for target\_index in target\_indices:

            # Calcula el vector de contexto para la palabra objetivo

            context\_vector = sum(word\_vectors) - word\_vectors[target\_index]

            print(context\_vector.shape)

            # Encuentra el sentido más cercano de la palabra objetivo

            target\_word\_vector = model.get\_word\_vector(target\_word.lower())

            pairs\_distance\_label=model.get\_nearest\_neighbors(target\_word, k=5)

            labels = [pair[1] for pair in pairs\_distance\_label]

|  |
| --- |
| Itera sobre las posiciones de la palabra objetivo,luego calcula el vector contexto de la palabra, para encontrar el sentido mas cercano de la palabra. |

7.is\_ambiguous = False

            for label in labels:

                if re.match(f'{target\_word.lower()}', label):

                    if is\_ambiguous:

                        ambiguous\_words.append(target\_word)

                        break

                    is\_ambiguous = True

if not is\_ambiguous:

                continue

|  |
| --- |
| Se encarga de verificar si la palabra tiene mas de un sentido y en caso de que tenga mas de un sentido se agrega a la lista de ambiguous\_words, en caso contrario el iterador continua. |

### 2.6 Descripción de la base de datos a utilizar

Los algoritmos que emplean las reglas para el análisis de las ambigüedades sintácticas,semánticas y léxicas presentes en los requisitos hacen uso de una base de datos para almacenar datos del requisito de software o el documento de requisitos. La base de datos está compuesta por las tablas AMBIGUEDADES\_ambiguedad y AMBIGUEDADES\_file.

La tabla AMBIGUEDADES\_ambiguedad contiene el requisito,el tipo y descripción de la ambigüedad mientras que la tabla AMBIGUEDADES\_file contiene el documento de requisitos. Las tablas fueron introducidas por los autores del presente trabajo, con el propósito de hacer posible el funcionamiento de la herramienta propuesta. Una vez que el usuario decide procesar el requisito o el documento de requisito ambas tablas capturan la información correspondiente a ellas. En la siguiente Figura 9 se muestra el diagrama de la base de datos.

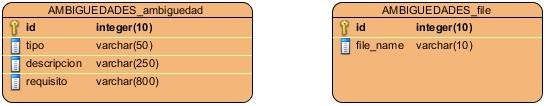


Figura 9: Base de datos

## Conclusiones parciales

* El uso de la metodología XP como guía del proceso de desarrollo de la herramienta permitió realizar un trabajo organizado y estructurado, generando los artefactos correspondientes a las fases de planificación y diseño.
* La recolección y limpieza de los requisitos de software de los proyectos de la UCI permitió la confección de un conjunto de datos que posee 19 357 requisitos de software, que constituye una línea base para futuras investigaciones.
* El empleo del patrón arquitectónico modelo vista plantilla y patrones de diseño, garantizan obtener una solución de software con poca dependencia entre clases, flexible al mantenimiento, cambios y escalabilidad.
* El flujo del proceso de PLN compuesto por las técnicas de tokenización, eliminación de ruido y clasificación, además de los algoritmos que implementan cada una de las reglas heurísticas definidas basadas en el criterio de no ambigüedad, permiten detectar ambigüedades léxica, semántica y sintáctica en los requisitos de software.

# Referencias Bibliográficas

Al-Harbi, O., Jusoh, S., & Norwawi, N. M. (2017). *Lexical Disambiguation in Natural Language Questions (NLQs)* (arXiv:1709.09250). arXiv. http://arxiv.org/abs/1709.09250

Alhogail, A., & Alsabih, A. (2021). Applying machine learning and natural language processing to detect phishing email. *Computers & Security*, *110*, 102414. https://doi.org/10.1016/j.cose.2021.102414

Alshemali, B., & Kalita, J. (2020). Improving the reliability of deep neural networks in NLP: A review. *Knowledge-Based Systems*, *191*, 105210.

*Aprendizaje supervisado vs. No supervisado; ¿cuál es mejor?* (s. f.). Alteryx. Recuperado 27 de marzo de 2024, de https://www.alteryx.com/es/glossary/supervised-vs-unsupervised-learning

Arora, C., Sabetzadeh, M., Briand, L., & Zimmer, F. (2017). Automated Extraction and Clustering of Requirements Glossary Terms. *IEEE Transactions on Software Engineering*, *43*(10), 918-945. https://doi.org/10.1109/TSE.2016.2635134

ASALE, R.-, & RAE. (s. f.). *Lengua | Diccionario de la lengua española*. «Diccionario de la lengua española» - Edición del Tricentenario. Recuperado 20 de abril de 2023, de https://dle.rae.es/lengua

Ashfaq, F., & Bajwa, I. S. (2021). Natural language ambiguity resolution by intelligent semantic annotation of software requirements. *Automated Software Engineering*, *28*(2), 13. https://doi.org/10.1007/s10515-021-00291-0

Ashfaq, F., Bajwa, I. S., Kazmi, R., Khan, A., & Ilyas, M. (2021). An Intelligent Analytics Approach to Minimize Complexity in Ambiguous Software Requirements. *Scientific Programming*, *2021*, e6616564. https://doi.org/10.1155/2021/6616564

Audi, R. (s. f.). *Diccionario Akal de filosofia* (Second Edition). Cambridge University Press, Ediciones Akal, S. A., 2004. Recuperado 18 de febrero de 2024, de https://biblioteca.multiversidadreal.com/index.php?page=12&id=1789

Bano, M. (2015). Addressing the challenges of requirements ambiguity: A review of empirical literature. *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, 21-24. https://doi.org/10.1109/EmpiRE.2015.7431303

Bhatia, J., Breaux, T. D., Reidenberg, J. R., & Norton, T. B. (2016). A theory of vagueness and privacy risk perception. *2016 IEEE 24th International Requirements Engineering Conference (RE)*, 26-35. https://ieeexplore.ieee.org/abstract/document/7765508/

Bhattacharjee, K., ShivaKarthik, S., Mehta, S., Kumar, A., Phatangare, S., Pawar, K., Ukarande, S., Wankhede, D., & Verma, D. (2020). Survey and Gap Analysis of Word Sense Disambiguation Approaches on Unstructured Texts. *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, 323-327. https://doi.org/10.1109/ICESC48915.2020.9155947

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). *Enriching Word Vectors with Subword Information* (arXiv:1607.04606). arXiv. https://doi.org/10.48550/arXiv.1607.04606

Bruscato, A. M., & Baptista, J. (2021). THE RESOLUTION OF AMBIGUOUS ANAPHORA IN PORTUGUESE, ENGLISH AND SPANISH (A PILOT STUDY). *Alfa: Revista de Linguística (São José Do Rio Preto)*, *65*, e13626.

Carreras, A. H., Cuevas, A. S., & González, A. H. (2023). Método de extracción automática de requisitos de software a partir de información textual no estructurada. *Revista Cubana de Transformación Digital*, *4*(1), Article 1.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., & Salakhutdinov, R. (2019). *Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context* (arXiv:1901.02860). arXiv. https://doi.org/10.48550/arXiv.1901.02860

Dang, N. C., Moreno-García, M. N., & De la Prieta, F. (2020). Sentiment Analysis Based on Deep Learning: A Comparative Study. *Electronics*, *9*(3), Article 3. https://doi.org/10.3390/electronics9030483

Dawood, O. S. (2017). From requirements engineering to uml using natural language processing–survey study. *European Journal of Industrial Engineering*, *2*(1), pp-44.

Deng, L., & Liu, Y. (2018). *Deep learning in natural language processing*. Springer. https://books.google.es/books?hl=es&lr=&id=y\_lcDwAAQBAJ&oi=fnd&pg=PR5&dq=deep+learning+algorithms+and+NLP&ots=ajpwg2WTg6&sig=n8TE9H-2IUxXnAhKSkLXMGyn5CI

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (arXiv:1810.04805). arXiv. https://doi.org/10.48550/arXiv.1810.04805

Espinosa Mateo, G. (2023). *Detección de ambigüedad léxica en requisitos de software en español mediante técnicas de Aprendizaje Automático*.

Ezzini, S., Abualhaija, S., Arora, C., & Sabetzadeh, M. (2022). Automated handling of anaphoric ambiguity in requirements: A multi-solution study. *Proceedings of the 44th International Conference on Software Engineering*, 187-199. https://doi.org/10.1145/3510003.3510157

Ferrari, A., Donati, B., & Gnesi, S. (2017). Detecting Domain-Specific Ambiguities: An NLP Approach Based on Wikipedia Crawling and Word Embeddings. *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, 393-399. https://doi.org/10.1109/REW.2017.20

Ferrari, A., & Esuli, A. (2019). An NLP approach for cross-domain ambiguity detection in requirements engineering. *Automated Software Engineering*, *26*(3), 559-598. https://doi.org/10.1007/s10515-019-00261-7

Ferrari, A., Gori, G., Rosadini, B., Trotta, I., Bacherini, S., Fantechi, A., & Gnesi, S. (2018). Detecting requirements defects with NLP patterns: An industrial experience in the railway domain. *Empirical Software Engineering*, *23*(6), 3684-3733. https://doi.org/10.1007/s10664-018-9596-7

Ferrari, A., Spoletini, P., & Gnesi, S. (2016a). Ambiguity and tacit knowledge in requirements elicitation interviews. *Requirements Engineering*, *21*(3), 333-355. https://doi.org/10.1007/s00766-016-0249-3

Ferrari, A., Spoletini, P., & Gnesi, S. (2016b). Ambiguity cues in requirements elicitation interviews. *2016 IEEE 24th International Requirements Engineering Conference (RE)*, 56-65. https://ieeexplore.ieee.org/abstract/document/7765511/

Frattini, J., Unterkalmsteiner, M., Fucci, D., & Mendez, D. (2024). *NLP4RE Tools: Classification, Overview, and Management* (arXiv:2403.06685). arXiv. https://doi.org/10.48550/arXiv.2403.06685

Gervasi, V., Ferrari, A., Zowghi, D., & Spoletini, P. (2019). Ambiguity in Requirements Engineering: Towards a Unifying Framework. En M. H. ter Beek, A. Fantechi, & L. Semini (Eds.), *From Software Engineering to Formal Methods and Tools, and Back: Essays Dedicated to Stefania Gnesi on the Occasion of Her 65th Birthday* (pp. 191-210). Springer International Publishing. https://doi.org/10.1007/978-3-030-30985-5\_12

Grave, E., Bojanowski, P., Gupta, P., Joulin, A., & Mikolov, T. (2018). *Learning Word Vectors for 157 Languages* (arXiv:1802.06893). arXiv. https://doi.org/10.48550/arXiv.1802.06893

Gupta, A. K., Deraman, A., & Siddiqui, S. T. (2019). A survey of software requirements specification ambiguity. *ARPN Journal of Engineering and Applied Sciences*, *14*(17), 3046-3061.

Harris, Z. S. (1954). Distributional Structure. WORD, *10*(2-3), 146-162. https://doi.org/10.1080/00437956.1954.11659520

Hiltunen, L. (2020, septiembre 17). *Reducing structural ambiguity in natural language software requirements specifications* [Diplomityö]. Laturi.Oulu.Fi. https://oulurepo.oulu.fi/handle/10024/16840

Hussen Maulud, D., M. Zeebaree, S. R., Jacksi, K., M.Sadeeq, M. A., & Hussein Sharif, K. (s. f.). *State of Art for Semantic Analysis of Natural Language Processing | Qubahan Academic Journal*. Recuperado 2 de febrero de 2024, de https://journal.qubahan.com/index.php/qaj/article/view/44

IBM. (s. f.). *¿Qué es el procesamiento del lenguaje natural (PLN)? | IBM*. Recuperado 27 de noviembre de 2023, de https://www.ibm.com/mx-es/topics/natural-language-processing

Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets*, *31*(3), 685-695. https://doi.org/10.1007/s12525-021-00475-2

Kamath, U., Liu, J., & Whitaker, J. (2019). *Deep Learning for NLP and Speech Recognition*. Springer International Publishing. https://doi.org/10.1007/978-3-030-14596-5

Khurana, D., Koli, A., Khatter, K., & Singh, S. (2023). Natural language processing: State of the art, current trends and challenges. *Multimedia Tools and Applications*, *82*(3), 3713-3744. https://doi.org/10.1007/s11042-022-13428-4

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019, septiembre 26). *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. arXiv.Org. https://arxiv.org/abs/1909.11942v6

Landolt, S., Wambsganss, T., & Söllner, M. (2021). A taxonomy for deep learning in natural language processing. *HICSS*, 1-10. https://www.alexandria.unisg.ch/bitstreams/1c0aad38-2ed6-4963-97d5-ee3bb9e4cde5/download

Lauriola, I., Lavelli, A., & Aiolli, F. (2022). An introduction to Deep Learning in Natural Language Processing: Models, techniques, and tools. *Neurocomputing*, *470*, 443-456. https://doi.org/10.1016/j.neucom.2021.05.103

Liu, H., & Lang, B. (2019). Machine learning and deep learning methods for intrusion detection systems: A survey. *applied sciences*, *9*(20), 4396.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach* (arXiv:1907.11692). arXiv. https://doi.org/10.48550/arXiv.1907.11692

Logacheva, V., Teslenko, D., Shelmanov, A., Remus, S., Ustalov, D., Kutuzov, A., Artemova, E., Biemann, C., Ponzetto, S. P., & Panchenko, A. (2020, marzo 14). *Word Sense Disambiguation for 158 Languages using Word Embeddings Only*. arXiv.Org. https://arxiv.org/abs/2003.06651v1

Loureiro, D., & Jorge, A. (2019). *Language Modelling Makes Sense: Propagating Representations through WordNet for Full-Coverage Word Sense Disambiguation* (arXiv:1906.10007). arXiv. https://doi.org/10.48550/arXiv.1906.10007

Loureiro, D., Rezaee, K., Pilehvar, M. T., & Camacho-Collados, J. (2021). Analysis and Evaluation of Language Models for Word Sense Disambiguation. *Computational Linguistics*, *47*(2), 387-443. https://doi.org/10.1162/coli\_a\_00405

Martínez, O. G. E., Reyes, S. V., & González, A. M. (2021). Uso de técnicas de procesamiento de lenguaje natural para la detección de requerimientos de software: Use of natural language processing techniques for software requirements detection. *South Florida Journal of Development*, *2*(5), 7323-7335. https://doi.org/10.46932/sfjdv2n5-072

Meta. (s. f.). *FastText*. Recuperado 29 de abril de 2024, de https://ai.meta.com/tools/fasttext

Moreno, A. (2017, octubre 17). Procesamiento del lenguaje natural ¿qué es? - IIC. *Instituto de Ingeniería del Conocimiento*. https://www.iic.uam.es/inteligencia/que-es-procesamiento-del-lenguaje-natural/

Navigli, R. (2012). A Quick Tour of Word Sense Disambiguation, Induction and Related Approaches. En M. Bieliková, G. Friedrich, G. Gottlob, S. Katzenbeisser, & G. Turán (Eds.), *SOFSEM 2012: Theory and Practice of Computer Science* (pp. 115-129). Springer. https://doi.org/10.1007/978-3-642-27660-6\_10

Orkphol, K., & Yang, W. (2019). Word Sense Disambiguation Using Cosine Similarity Collaborates with Word2vec and WordNet. *Future Internet*, *11*(5), Article 5. https://doi.org/10.3390/fi11050114

Peer, J., Mordecai, Y., & Reich, Y. (2024, marzo 10). *NLP4ReF: Requirements Classification and Forecasting: From Model-Based Design to Large Language Models*.

Porto Dapena, J. Á. (2018). *Sobre ambigüedad y vaguedad en los diccionarios*. https://doi.org/10.25145/j.refiull.2018.36.017

*¿Qué es el aprendizaje no supervisado? | IBM*. (s. f.). Recuperado 27 de marzo de 2024, de https://www.ibm.com/mx-es/topics/unsupervised-learning

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). *Improving language understanding by generative pre-training*. https://www.mikecaptain.com/resources/pdf/GPT-1.pdf

Rafiq, U., Bajwa, S. S., Wang, X., & Lunesu, I. (2017). Requirements Elicitation Techniques Applied in Software Startups. *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 141-144. https://doi.org/10.1109/SEAA.2017.73

Rahman, N., & Borah, B. (2022). An unsupervised method for word sense disambiguation. *Journal of King Saud University - Computer and Information Sciences*, *34*(9), 6643-6651. https://doi.org/10.1016/j.jksuci.2021.07.022

Ramírez Reyes, J., & Enríquez González, S. de las M. (2022). *SIDARES: herramienta de procesamiento del lenguaje natural para la detección de ambigüedad léxica y sintáctica en requisitos de software*.

Riaz, M. Q., Butt, W. H., & Rehman, S. (2019). Automatic detection of ambiguous software requirements: An insight. *2019 5th International Conference on Information Management (ICIM)*, 1-6. https://ieeexplore.ieee.org/abstract/document/8714682/

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). *DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter* (arXiv:1910.01108). arXiv. https://doi.org/10.48550/arXiv.1910.01108

Sarker, I. H. (2021). Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN Computer Science*, *2*(3), 160. https://doi.org/10.1007/s42979-021-00592-x

Seleey, D. A., Triana, C. E. P., & López, D. C. C. (2022). INGENIERÍA DE REQUERMIENTOS E INTELIGENCIA ARTIFICIAL: UNA REVISIÓN DE LA LITERATURA. *REVISTA COLOMBIANA DE TECNOLOGIAS DE AVANZADA (RCTA)*, *1*(39), Article 39. https://doi.org/10.24054/rcta.v1i39.1395

Shreda, Q. A., & Hanani, A. A. (2021). Identifying Non-functional Requirements from Unconstrained Documents using Natural Language Processing and Machine Learning Approaches. *IEEE Access*, 1-1. https://doi.org/10.1109/ACCESS.2021.3052921

Simón, A., García Bermúdez, R., Del, M., Cruz, R., & Párraga Valle, J. E. (2018). *A Review of Automatic Requirements Extraction from Natural Language Text*. 47-58.

*Stemming*. (1994, 2024). MathWorks. https://la.mathworks.com/discovery/stemming.html

*Stop word removal - Natural Language Processing: Python and NLTK [Book]*. (s. f.). Recuperado 13 de marzo de 2024, de https://www.oreilly.com/library/view/natural-language-processing/9781787285101/ch02s07.html

Tekli, J. (2016). An Overview on XML Semantic Disambiguation from Unstructured Text to Semi-Structured Data: Background, Applications, and Ongoing Challenges. *IEEE Transactions on Knowledge and Data Engineering*, *28*(6), 1383-1407. https://doi.org/10.1109/TKDE.2016.2525768

Term Extraction. (s. f.). *FiveFilters.Org*. Recuperado 13 de marzo de 2024, de https://www.fivefilters.org/term-extraction/

Toro, A., & Peláez, L. E. (2016). Ingeniería de Requisitos: De la especificación de requisitos de software al aseguramiento de la calidad. Cómo lo hacen las Mipymes desarrolladoras de software de la ciudad de Pereira. *Entre Ciencia e Ingeniería*, *10*(20), 117-123.

Tripodi, R., & Navigli, R. (2019). Game Theory Meets Embeddings: A Unified Framework for Word Sense Disambiguation. En K. Inui, J. Jiang, V. Ng, & X. Wan (Eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (pp. 88-99). Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1009

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). *Attention Is All You Need* (arXiv:1706.03762). arXiv. http://arxiv.org/abs/1706.03762

Vázquez Pérez, S. (2023). *Resolución de la ambigüedad semántica mediante métodos basados en conocimiento y su aportación a tareas de PLN.* https://rua.ua.es/dspace/bitstream/10045/11456/1/Tesis\_vazquez.pdf

Wang, M., & Wang, Y. (2021). Word Sense Disambiguation: Towards Interactive Context Exploitation from Both Word and Sense Perspectives. En C. Zong, F. Xia, W. Li, & R. Navigli (Eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (pp. 5218-5229). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.acl-long.406

Yang, H., de Roeck, A., Gervasi, V., Willis, A., & Nuseibeh, B. (2011). Analysing anaphoric ambiguity in natural language requirements. *Requirements Engineering*, *16*(3), 163-189. https://doi.org/10.1007/s00766-011-0119-y

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2020). *XLNet: Generalized Autoregressive Pretraining for Language Understanding* (arXiv:1906.08237). arXiv. https://doi.org/10.48550/arXiv.1906.08237

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E.-V., & Batista-Navarro, R. T. (2021). Natural Language Processing for Requirements Engineering: A Systematic Mapping Study. *ACM Computing Surveys*, *54*(3), 55:1-55:41. https://doi.org/10.1145/3444689

Zhou, Z.-H. (2021). *Machine Learning*. Springer Nature.

1. Tomáš Mikolov es un informático checo que trabaja en el campo del aprendizaje automático. [↑](#footnote-ref-0)
2. https://fasttext.cc [↑](#footnote-ref-1)
3. Backend es el entorno donde se ejecutan uno o varios programas informáticos que almacenan, asegura ,procesan y analizan datos en forma continua para luego generar resultados(información), que es representada en forma visual por las Frontend developers (Fernando 2018) [↑](#footnote-ref-2)