



TESINA DE LICENCIATURA

Título: Construcción semiautomática de un documento LEL utilizando técnicas de procesamiento de lenguaje natural.

Autores: Guillermo Federico Carrilao Avila.

Director: Dr. Leandro Antonelli, Dr. Waldo Hasperué.

Carrera: Licenciatura en Sistemas.

Resumen

Los analistas deben consumir y analizar una gran cantidad de información sobre el dominio y requerimientos de un proyecto para construir los diferentes documentos de ingeniería de software que se utilizan a lo largo del ciclo de vida de desarrollo. La calidad de estos documentos son de vital importancia para el éxito de un desarrollo e implica un gran esfuerzo por parte de los analistas. Por esta razón, esta tesis propone una herramienta que sea capaz de tamizar y sintetizar toda la información de un dominio de manera tal que asista a los analistas y facilite su trabajo. En este trabajo se busca facilitar el esfuerzo de construir documentos LEL, los cuales son glosarios que especifican de manera detallada cada elemento de un dominio (símbolos) y sus posibles relaciones entre sí (impactos), con el objetivo de ser un nexo comunicativo entre los analistas y los clientes.

Para lograr esto se implementó una solución semi-automatizada para crear documentos LEL a partir de texto ingresado por el usuario, donde la información que se toma como entrada debe contener detalles de requerimientos o de dominio de una aplicación.

Finalmente el trabajo muestra los resultados del algoritmo de aprendizaje automático utilizado y el análisis comparativo de los resultados esperados con los obtenidos por la herramienta, donde se detalla el porqué de los errores y aciertos de los mismos.

Palabras Claves

Léxico Extendido del Lenguaje, Procesamiento de Lenguaje Natural, Algoritmos de Aprendizaje Automático.

Conclusiones

Los procesos automatizados de recolección de información en datos de tipo texto pueden ser muy conflictivos debido a la gran complejidad que poseen las estructuras sintácticas del idioma Español. Sin embargo, se han encontrado buenos resultados por parte de la herramienta, donde se realizó una comparativa entre los resultados esperados y los dispuestos por la herramienta y se llegó a la conclusión de que la mayoría de los elementos no encontrados se dieron por la falta de funcionalidades que todavía la herramienta no contempla y no por fallos de la misma.

Trabajos Realizados

Se implementó una herramienta que es capaz de identificar símbolos e impactos (elementos de un glosario LEL) a partir de información de tipo texto ingresada por un usuario. Para su implementación se utilizaron técnicas de procesamiento de lenguaje natural, reglas heurísticas y algoritmos de aprendizaje automático, donde se utilizó un framework web para disponer una interfaz amigable para el usuario. Adicionalmente este trabajo final se destaca en ser una 'novel tool', ya que es el primer desarrollo en el idioma español para generar estos documentos a partir de entrada de tipo texto.

Trabajos Futuros

Esta tesina sienta las bases de una arquitectura basada en el uso de reglas heurísticas, técnicas de procesamiento de lenguaje natural, y algoritmos de aprendizaje automático para la obtención y derivación de información. Este trabajo debe verse como el punto de partida para enriquecer cada uno los módulos mencionados en la obtención de glosarios LEL. Desde el punto de vista del procesamiento de lenguaje natural, el proceso de identificación de sujetos y en especial cuando el texto posee estructuras gramaticales complejas, es un gran punto de mejora e investigación a futuro. Como herramienta de asistencia automática es ideal la implementación de una API que permita su uso en procesos más complejos o para ser utilizada por otras herramientas.

Adicionalmente a medida que la herramienta sea utilizada, se espera que los resultados obtenidos de su uso sirvan de retroalimentación para el algoritmo de aprendizaje automático y de esta manera mejorar el output gradualmente.



UNIVERSIDAD NACIONAL
DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE GRADO

**Construcción semiautomática de un
documento LEL utilizando técnicas
de procesamiento de lenguaje natural.**

Autor:

Guillermo Federico Carrilao Avila

Directores:

Dr. Leandro Antonelli

Dr. Waldo Hasperué

Agradecimientos

Agradezco en primer lugar a mi familia, que sin ellos nada de esto hubiese sido posible y fueron la base de todo.

A mis amigos, que siempre estuvieron acompañándome durante todo este proceso con mates, charlas y risas.

A mis directores, que me acompañaron durante todo el proceso de desarrollo y escritura de esta tesina con recomendaciones, consejos y palabras de apoyo.

Al grupo de seguridad informática de la facultad, que me ayudaron a encontrar mi camino profesional.

Y por último a todas aquellas personas que me enseñaron algo alguna vez.

Índice general

1. Introducción	7
1.1. Motivación	7
1.2. Objetivos	9
1.3. Contribución	9
1.4. Estructura de la tesina	10
2. Background	11
2.1. Léxico extendido del lenguaje	11
2.1.1. Introducción	11
2.1.2. Definición	12
2.1.3. Categorías de símbolos	12
2.1.4. Confección de un LEL	13
2.1.5. Conclusión	14
2.2. Procesamiento de lenguaje natural	14
2.2.1. Introducción	14
2.2.2. Definición	15
2.2.3. Uso de procesamiento de Lenguaje Natural	16
2.2.4. Normalización de texto, Lemmatization y Stemming	17
2.2.5. Tokenización	19
2.2.6. Bigramas y trigramas	20
2.2.7. Parts of Speech Tagging (POS Tagging)	20
2.2.8. Dependencias gramaticales	21
2.2.9. Conclusión	22
2.3. Aprendizaje automático	22
2.3.1. Introducción	22
2.3.2. Aprendizaje automático supervisado	23
2.3.3. Aprendizaje automático no-supervisado	23
2.4. Categorización de texto	23
2.4.1. Introducción	23
2.4.2. Definición	24

2.4.3.	Ponderación de las palabras	24
2.4.4.	Enfoque Naive Bayes	26
2.4.5.	Conclusión	28
3.	Estado del Arte	29
3.1.	Introducción	29
3.2.	Derivaciones	30
3.2.1.	Diagrama de clases	30
3.2.2.	Casos de usos	34
3.2.3.	Modelos entidad relación	37
3.2.4.	Glosarios de términos	39
3.3.	Resumen de investigaciones	41
3.4.	Conclusión	46
4.	Herramienta propuesta	47
4.1.	Introducción	47
4.2.	Análisis de la estructura LEL	47
4.3.	Arquitectura de la plataforma	48
4.3.1.	Vista general de la plataforma	49
4.3.2.	Python como lenguaje de programación	49
4.3.3.	Plataforma web	49
4.3.4.	Procesamiento de lenguaje natural: Spacy y NLTK	50
4.3.5.	Aprendizaje automático: Sklearn	50
4.3.6.	Base de datos MySQL	51
4.3.7.	Repositorio central de multi-lenguaje	51
4.4.	Esquema de extracción de información	52
4.5.	Diseño de la herramienta	53
4.6.	Implementación y flujo de la herramienta	57
4.6.1.	Datos de entrada	57
4.6.2.	Módulo de reglas heurísticas	58
4.6.3.	Uso de módulos de reglas heurísticas y reestructuración de sen- tencias	63
4.6.4.	Normalización de la entrada	65
4.6.5.	Estructuración de texto: Obtener Sujetos y Objetos Principales	66
4.6.6.	Interacción del usuario con los símbolos	69
4.6.7.	Búsqueda de relaciones	70
4.6.8.	Búsqueda de nociones y retroalimentación	74

5. Experimentación	77
5.1. Construcción del clasificador de Naive Bayes	77
5.1.1. Introducción	77
5.1.2. Muestras de aprendizaje	78
5.1.3. Preparación de la información para el clasificador de Bayes	79
5.1.4. Desbalanceo de datos y submuestreo (downsampling)	82
5.1.5. K-Fold Cross Validation	83
5.1.6. Resultados obtenidos	86
5.2. Resultados de la herramienta	86
5.2.1. Introducción	86
5.2.2. Resultados de la herramienta comparados con los documentos LEL	87
5.3. Conclusión de resultados	90
6. Conclusiones	91
Apéndice	99
A. Manual de uso	100
A.1. Instalación	100
A.1.1. Docker	100
A.1.2. Docker Compose	100
A.1.3. Instalación de imagen	100
A.2. Uso de la herramienta	102

Capítulo 1

Introducción

Este capítulo busca interiorizar al lector acerca de las problemáticas principales que aborda esta tesina, definiendo cuáles son los objetivos a obtener, la motivación por la cual se quiso desarrollar esta herramienta y la contribución que se espera tanto en el ámbito académico como en el laboral.

1.1 Motivación

En los últimos años la necesidad de automatizar o reproducir el comportamiento humano ha crecido a pasos agigantados, la búsqueda de reducir costos, esfuerzo y tiempo combinado con la evolución de las distintas tecnologías y prestaciones informáticas han sido disparadores importantes para que esto ocurra [1][2][3].

Aunque la ingeniería de software es una rama importante en todo desarrollo informático, la automatización de ciertos procedimientos que ocurren reiteradas veces en un flujo de trabajo normal, aún no han sido investigados o implementados en profundidad [1][4][5][6]. Esta tesina surge de la necesidad de contar con herramientas que elaboren documentos que faciliten las especificaciones de requerimientos [2][6]. Dichos documentos comúnmente se desarrollan manualmente a partir del análisis de información en formato tipo texto por parte de un experto o un equipo de ingeniería de software, por lo tanto, la utilización de técnicas de procesamiento de lenguaje natural se acopla perfectamente en este marco para estructurar dicha información [5][7]. Sin embargo, con la utilización de una muestra amplia de documentos es necesario contar con técnicas que nos ayuden a identificar qué información es importante y cuál no [6][8].

El procesamiento del lenguaje natural involucra un conjunto de técnicas que permiten de una manera automática a las computadoras a entender, interpretar y mani-

pular el lenguaje humano. Generalmente se basa en herramientas que son capaces de estructurar textos escritos en lenguaje natural, tales como emails, páginas web, etc. [1][5][7][9][10]. Por otro lado el aprendizaje automático tiene como finalidad utilizar técnicas o algoritmos que permitan que las computadoras puedan aprender a detectar patrones de un problema determinado. La base para el aprendizaje se basa en mejorar a partir de las experiencias, es decir, se busca encontrar algoritmos capaces de procesar muestras de datos para generar modelos. Estos modelos resultantes deben ser capaces de generalizar comportamientos e inferencias para un conjunto más amplio de datos [6][8].

A partir de lo mencionado, en este trabajo de fin de grado se propone la elaboración semiautomática del Léxico Extendido del Lenguaje, (Language Extended Lexicon, LEL)[1][11][12] que se explicará brevemente a continuación.

La etapa de relevamiento de requerimientos es un área clave para el éxito de un proyecto. En particular construir una especificación de requerimientos de software (ERS) adecuada es esencial, ya que constituyen un medio de comunicación entre los miembros del equipo de desarrollo con el resto de los stakeholders, y es necesario que el equipo de desarrollo posea los requerimientos correctos para construir el producto [1][11][12].

Un problema recurrente que se produce en cada especificación de requerimientos es que el equipo técnico maneja un lenguaje distinto al de los stakeholders. Esta diferencia de lenguaje provoca un desentendimiento entre las partes, generando especificaciones de requerimientos incorrectas, ambiguas o que no satisfagan las expectativas de los stakeholders [11][12].

A partir de los inconvenientes mencionados surge el Léxico Extendido del Lenguaje, el cual es un glosario que permite describir el lenguaje del dominio de la aplicación utilizando el lenguaje natural [11]. Por consiguiente, los stakeholders y analistas, emplean este glosario para complementar conocimientos y de esta manera generar una homogeneidad en la comunicación entre ambas partes [11][12].

La principal motivación de este trabajo de fin de grado es reducir el esfuerzo y tiempo que requiere elaborar un LEL, debido a que la generación de estos documentos requieren una exhaustiva lectura, análisis y comprensión de un dominio [1]. En la actualidad no existe alguna aproximación o investigación acerca de la creación semiautomática o automática de LEL en ningún idioma. Sin embargo existen implementaciones con herramientas de procesamiento de lenguaje natural para generar especificaciones de requerimientos en Inglés [13][9][10], tales como casos de uso, historias de usuarios, que se podrán usar como puntapié inicial para investigar y descubrir

herramientas, encontrar soluciones a problemas recurrentes y detectar problemas que aún no han sido resueltos [6].

1.2 Objetivos

El objetivo principal de esta tesina consiste en desarrollar una herramienta con el propósito de asistir a los analistas de sistemas en el proceso de capturar y describir el lenguaje propio de un dominio. Dicha herramienta tiene como salida elementos que facilitan la creación y modelado de un documentos LEL utilizando como entrada documentos de texto no estructurados. Para el presente desarrollo se utilizaron técnicas de procesamiento de lenguaje natural y algoritmos de aprendizaje automático para llevar a cabo el procesamiento de texto y capturar información que sirvan para la construcción de un documento LEL. Dicho objetivo se llevó a cabo a través de los siguientes subobjetivos específicos:

- Investigar publicaciones existentes acerca de la elaboración semiautomática de especificaciones de requerimientos y dominios.
- Analizar y comparar el corpus de entrada más adecuado teniendo en cuenta la disponibilidad de los mismos en un entorno real.
- Analizar herramientas de procesamiento de texto de lenguaje natural para el lenguaje español.
- Analizar algoritmos de aprendizaje automático para mejorar la salida de la aplicación.
- Implementar una aplicación que genere de manera semiautomática documentos LEL dado la entrada documentos de texto no estructurados.

1.3 Contribución

La presente tesina pretende contribuir con:

- Semi-automatización en la construcción de glosarios LEL's.
- Feedbacks positivos por expertos o alumnos de postgrado donde puedan acotar sugerencias, mejoras e ideas dentro de lo posible para obtener un producto de mejor calidad.

- El resultado conseguido va a variar de acuerdo a la calidad de entrada de datos y por supuesto a la cantidad de datos que se tiene como guía para realizar el sistema. Sin embargo, con esta tesina espera desplegar el uso de algoritmos de aprendizaje automático y procesamiento de lenguaje natural en el resto de los campos de la ingeniería de software, como así en todas las áreas de informática donde el trabajo del analista se vuelve tedioso y repetitivo.

1.4 Estructura de la tesina

Explicada la motivación y los objetivos abordados por esta tesina, el siguiente capítulo [Background](#) describe detalladamente los conocimientos necesarios para poder comprender cada una de las aristas desarrolladas en este trabajo. El capítulo [Estado del Arte](#) hace énfasis en las investigaciones realizadas que se utilizaron como punto de partida teórico para desarrollar la estructura de la herramienta. Los capítulos [Herramienta propuesta](#) y [Experimentación](#) describen la implementación de la herramienta y las pruebas realizadas para validar los resultados de la misma, respectivamente. Por último, el capítulo [Conclusiones](#) detalla cada una de las conclusiones realizadas a partir de la implementación y resultados de la herramienta.

Capítulo 2

Background

Este capítulo describe las distintas aristas que se necesitan para comprender el desarrollo realizado. La primera sección describe el tipo de documento a elaborar de manera automática, los componentes más importantes del mismo, y un análisis sobre cada uno de estos elementos. Las siguientes secciones describirán las técnicas de procesamiento de lenguaje natural y algoritmos de aprendizaje automático utilizadas para la implementación de este trabajo.

2.1 Léxico extendido del lenguaje

2.1.1. Introducción

El Léxico Extendido del Lenguaje (LEL) es una técnica para especificar el dominio de una aplicación, en otras palabras, el contexto en el que una aplicación o sistema trabaja [11]. Este tipo de documentos surge por la necesidad de unificar los conceptos y las ideas que tienen el equipo experto y los stakeholders sobre un dominio. Esta técnica cuando es empleada tiene como resultado un gran beneficio para aquellos usuarios que tienen habilidades orientadas a la ingeniería de software ya que pueden aprovechar mejor las características intrínsecas de este documento, sin embargo para aquellas personas sin estas habilidades es de gran ayuda debido a que funciona como un puente conceptual del dominio entre los stakeholders y el equipo técnico. Por lo general este tipo de documento se lo confecciona antes de realizar una especificación de requerimientos ya que su objetivo es poder mejorar e identificar el conocimiento acerca del contexto de la aplicación, escenarios, preocupaciones generales, dando como resultados especificaciones más precisas. La construcción del mismo puede ser realizada por un experto que sea parte del equipo técnico o puede ser realizada entre todos los stakeholders que formen parte del proyecto, en este capítulo se verán las

distintas técnicas y modalidades para su construcción.

2.1.2. Definición

El Léxico Extendido del Lenguaje (LEL) es un glosario que tiene por finalidad describir el lenguaje del contexto de la aplicación [12]. Su objetivo principal es describir ciertas palabras o frases peculiares de la aplicación que sirvan para poder comprender el contexto de la misma. LEL proviene de una idea simple de “entender el lenguaje del problema sin preocuparse por entender el problema” [12]. De esta manera, el analista es capaz de reconocer y comprender el lenguaje de la aplicación, y lo utilizará como base el conocimiento adquirido para escribir requerimientos. El LEL es un glosario en el cual se definen símbolos (términos o frases), y cada símbolo se define a través de dos atributos: la noción y los impactos. La noción describe la denotación, es decir, describe las características intrínsecas y sustanciales del símbolo. Y por su parte, los impactos describen la connotación, es decir, un valor secundario que adopta por asociación con un significado estricto [12]. Existen dos principios que se deben seguir al describir símbolos: el principio de circularidad y el principio de vocabulario mínimo. El principio de circularidad establece que durante la descripción de los símbolos se debe maximizar el uso de otros símbolos descritos en el LEL. Por su parte, el principio de vocabulario mínimo complementa el principio de circularidad y establece que en las descripciones se debe minimizar el uso de símbolos externos al LEL y los símbolos que se utilicen deben tener una definición clara, unívoca y no ambigua. Estas características permiten comprender el lenguaje de la aplicación de una manera simple, y también hacen que el LEL pueda ser visto como un grafo. Los símbolos se deben categorizar en una de cuatro categorías básicas con el fin de especializar la descripción de los atributos. Las cuatro categorías básicas son: sujeto, objeto, verbo y estado. Los sujetos se corresponden con elementos activos dentro del contexto de la aplicación, mientras que los objetos se corresponden con elementos pasivos. Por su parte, los verbos son las acciones que realizan los sujetos sobre los objetos. Finalmente, los estados representan las situaciones en las que se pueden encontrar los sujetos o los objetos previo y luego de realizar las acciones.

2.1.3. Categorías de símbolos

La descripción de la noción y los impactos se debe ajustar a la categoría del símbolo que se debe describir. Para los sujetos, la noción debe describir las características que debe satisfacer para poder considerarse como tal. Por su parte, los impactos, deben describir las acciones que realizan. Para los objetos, la noción debe describir los atributos constitutivos del objeto, mientras que los impactos deben describir las acciones que se realizan sobre ellos. Para los verbos, la noción debe describir el objetivo

o fin que persiguen, mientras que los impactos deben describir los pasos necesarios para cumplir con la acción. Finalmente, la noción de los estados debe describir la situación que representa, mientras que los impactos deben describir las acciones que se pueden realizar a partir de ese estado y el nuevo estado al que se puede acceder. La tabla 2.1 resume la información[12].

Categoría	Noción	Impacto
Sujeto	Características y condiciones que el sujeto debe satisfacer	Acciones que el sujeto realiza
Objeto	Atributos o características constitutivas del objeto	Acciones que se realizan sobre los objetos
Verbo	Objetivo o fin que el verbo persigue	Pasos necesarios para cumplir con el objetivo
Estado	Situación que el verbo representa	Acciones posibles de realizar desde el estado y nuevo estado al que se arriba.

Tabla 2.1: Categorías de símbolos y template para describir cada categoría

2.1.4. Confección de un LEL

La construcción de un LEL puede realizarse de varias maneras, siempre teniendo en cuenta la dimensión de la aplicación o para que está siendo usado. El proceso tradicional para construir un LEL consiste en dos actividades: identificar y describir los símbolos que son realizados exclusivamente por un ingeniero o un analista de requerimientos. Sin embargo, esto permite solo permite la subjetividad de una persona, a partir de aquí surge una de las técnicas propuestas por los doctores Leandro Antonelli, Gustavo Rossi y Alejandro Oliveros, la cual se basa en construir un LEL de manera colaborativa entre los stakeholders de la aplicación. [14].

En este enfoque colaborativo se propone las mismas actividades que en el proceso tradicional: identificar y describir los símbolos, pero también se agrega una actividad social común: la cual es expresar «me gusta» a una expresión que define un símbolo.

En este enfoque de colaboración no incluye ningún analista y las actividades son realizadas directamente por los stakeholders interesados. Por lo tanto, la identificación y la descripción de los símbolos también ocurre de manera colaborativa, y diferentes personas cooperan para definir un símbolo. Por ejemplo, una persona identifica un símbolo y otra incluye una expresión para describirlo. Luego, otra persona diferente incluye una expresión más para describir el símbolo. Finalmente, alguien más revisa el símbolo y su definición y puede validar que le gusta la manera en la que está expresado. Entonces continuamente se está trabajando de manera colaborativa y validando cada símbolo definido [14].

Para identificar y describir símbolos existen múltiples referencias a seguir en cuanto a buenas prácticas, esto permite que sin importar cual sea la técnica utilizada para confeccionar un LEL, haya como resultado un documento LEL completo y poco ambiguo que sirva de guía para realizar una especificación de requerimientos posteriormente[12].

2.1.5. Conclusión

Como se pudo ver en esta sección, un documento LEL mejora ampliamente el conocimiento acerca del dominio para todos los stakeholders, a su vez esto influye directamente sobre las especificaciones de requerimiento, ya que con un conocimiento adquirido acerca de una aplicación y su dominio, los requerimientos especificados serán más completos y menos ambiguos. Pero un LEL no solo se utiliza para mejorar el conocimiento del dominio, si no que ya existen técnicas estudiadas acerca de cómo utilizar estrategias de derivación para transformar los símbolos y sus descripciones en requerimientos de software directamente, ya sea una historia de usuario o un caso de uso [11]. También, puede ser usado para estimar el tamaño de una aplicación utilizando como referencia un documento LEL [1]. Por lo tanto, este trabajo final toma más relevancia en cuanto a las posibilidades que existen luego que se confecciona un LEL para luego derivarlo en otro tipo de documento. Si se puede construir automáticamente un documento que tiene tanta influencia sobre etapas posteriores de una ingeniería de software, esto permitirá que luego se pueda avanzar sobre estas posibles derivaciones, también de manera semiautomática o automática.

2.2 Procesamiento de lenguaje natural

2.2.1. Introducción

El lenguaje es un sistema usado de manera intencional por los humanos para comunicar y razonar. El lenguaje natural (LN) es el lenguaje normalmente usado por

una comunidad para transmitir o expresar algo, ya sea información o emociones. El lenguaje natural es transversal a todas las áreas de estudio, y desde hace tiempo que se analiza la idea de procesar dicha información de manera automática. Por ejemplo en la ingeniería de software la gran mayoría de pedidos de requerimientos están escritos en LN, y dichos requerimientos están escritos por una persona no experta. En un flujo de trabajo normal, dichos requisitos tienen que ser analizados por un especialista que tiene que ser capaz de comprender y relacionar los conceptos de un dominio específico para generar un SRS, un LEL, o cualquier documento que se necesite en un proyecto. Debido a que el LN es altamente informal requiere una gran intervención del experto para realizar un diseño de una solución software [15]. Dada la naturaleza ambigua del LN y los diversos factores que pueden afectar la interpretación de los textos escritos en LN suele ser un trabajo bastante costoso de realizar.

Un documento LEL surge específicamente del Lenguaje Natural y sin tener algún documento técnico de respaldo, de hecho la idea principal es que se realice antes de tener un documento SRS ya que ayuda a la construcción del mismo. El analista encargado de la construcción del LEL debe analizar, interpretar y razonar información escrita en Lenguaje Natural por parte de un cliente o los stakeholders de un proyecto, y a partir de ese trabajo construye el documento. Entonces, el procesamiento de lenguaje natural surge de la necesidad de procesar documentos de tipo y obtener conocimiento de manera automática. Tomando como base la ingeniería de software, el PLN es una gran herramienta que puede ayudar a construir automáticamente un documento que se necesite a partir de Lenguaje Natural.

2.2.2. Definición

Cuando se habla de procesar un lenguaje, se habla de realizar una traducción de la versión de un texto desde una lengua a otra. De este modo, para ejecutar un procesamiento de lenguaje natural (PLN) se requiere la transformación del texto en una representación semántica que se pueda utilizar para razonar, tomar decisiones y ejecutar tareas específicas. Esta representación se logra por medio de procesos de parseo o construcción de un árbol de análisis a partir de una gramática. Si la gramática es sintáctica, por medio de dicho árbol de análisis se genera información sobre las categorías gramaticales de las palabras y la función sintáctica asociada (por ejemplo: identificación del sujeto, verbo, predicado y complementos). Mientras que, si la gramática es semántica, el árbol de análisis ya es bastante próximo a la representación lógica que permite el razonamiento y la ejecución. El PLN como disciplina busca desarrollar programas computacionales que sean capaces de ejecutar actividades relacionadas con la comprensión, análisis y producción de textos o discursos escritos en lenguaje natural, de una manera similar a como lo hace el ser humano. Por su parte, el lenguaje

se estudia desde diferentes disciplinas orientadas hacia la lingüística, la cual a su vez estudia todos los hechos y fenómenos relacionados con el LN. Este subcampo de la Ciencia de la Computación se encarga de formular e investigar métodos y técnicas informáticas capaces de procesar y analizar grandes cantidades de datos en lenguaje natural. La información existente en Internet y el conocimiento en general se describen en lenguaje natural. El lenguaje natural es una forma de comunicación imprecisa y ambigua. Su comprensión depende del contexto de cada situación y la manera en que las palabras son usadas, es decir que está conformada por diferentes variables tales como cultura, lugar, o como por ejemplo la edad de las personas que utilizan dichas palabras o expresiones. Dicho lenguaje no es más que el medio utilizado por los seres humanos para comunicarse en forma oral, gestual o escrita. El lenguaje revela muchos aspectos: pensamientos, creencias, sentimientos, etc. Desde un punto de vista computacional, el lenguaje natural es el punto de partida hacia lenguajes estructurados interpretables por una máquina. Así, el PLN está fundamentado en la Lingüística Computacional, la cual es concebida desde la lingüística aplicada. El término lingüística computacional se refiere al campo interdisciplinario entre la lingüística, fonética, ciencias de la computación, ciencias cognitivas, inteligencia artificial, lógica formal (Clegg, 2008) y se orienta a la construcción de modelos de lenguajes “entendibles” por las computadoras [16]. Ambas disciplinas tienen al lenguaje natural como objeto de estudio pero considerándolo desde diferentes enfoques. Si bien inicialmente el NLP estaba relacionado con las gramáticas formales y la búsqueda de analizadores sintácticos, en la actualidad incluye otras tantas áreas, como la extracción de conocimiento o la recuperación de información.

2.2.3. Uso de procesamiento de Lenguaje Natural

Un enfoque utilizado para trabajar con las herramientas de procesamiento de lenguaje natural fue la Extracción de Información, la cual consiste en obtener información estructurada automáticamente de documentos no estructurados para que luego puedan ser leídos por una máquina [17]. Este enfoque consiste en buscar entidades dentro de un texto, y relaciones a partir de estas entidades y a partir de esto generar nueva información estructurada. La arquitectura estudiada para este trabajo de fin de grado fue basada en varios autores, ya que todos proponen una idea similar de cómo orientar una arquitectura que procese y estructure una información de tipo texto [3] [18] [17].

Cada una de las burbujas de la arquitectura 2.1, se explican a lo largo de esta sección. Resumiendo, el texto ingresa a un proceso de estructuración con el objetivo de poder encontrar entidades, para luego encontrar relaciones entre ellas, y de esta manera tener las herramientas para generar nuevo conocimiento y tener una estructura de la información que hay en el texto.

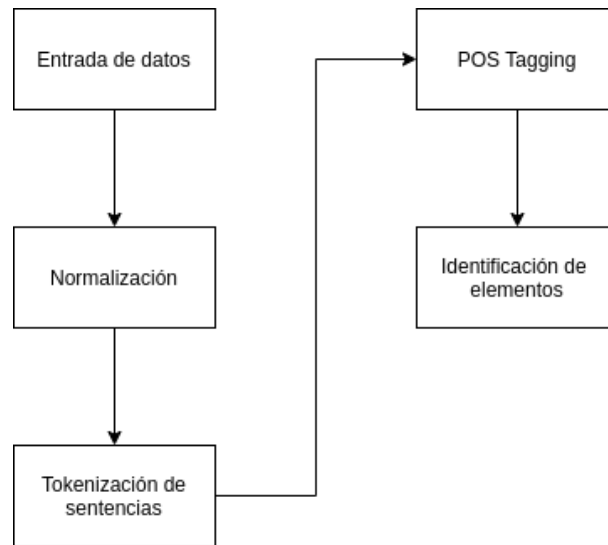


Figura 2.1: Flujo de extracción de información

2.2.4. Normalización de texto, Lemmatization y Stemming

La primera etapa utilizada para la extracción de información en el enfoque de esta tesina fue normalizar las palabras o el texto que se tiene como entrada. Esto permite tener una única forma canónica con la que se realiza un procesamiento. Adicionalmente en esta sección se explica la utilización de Lemmatizadores y Stemmers que son dos técnicas muy utilizadas para evitar la presencia de falsos positivos en la salida de resultados.

Normalización de texto

No existe una manera universal de normalizar una entrada de texto, ya que siempre depende de las características del mismo, ya sea por el nivel de formalidad o estructura que tenga, pero se basa principalmente en unificar la manera en la que se representan las palabras. A su vez, un proceso de normalización también puede tener procesamientos en el cual se eliminan palabras o expresiones que no aportan una relevancia en el texto. Las siguientes tareas son comunes en la mayoría de los procesos de normalización [3]:

- Convertir todas las letras a minúsculas o mayúsculas
- Convertir números en palabras o eliminar números
- Eliminar signos de puntuación, acentos y otros signos.

- Eliminar espacios en blanco
- Transformar abreviaciones
- Eliminar palabras que no aporten semántica a una sentencia (stopwords)
- Eliminar letras repetidas

Todas estos tipos de transformaciones garantizan que si dos o más palabras significan lo mismo pero que están escritas de distintas maneras puedan ser representadas de una manera única dentro un programa. Para dar un ejemplo, podríamos identificar que se encuentra la palabra «USA» y «Estados Unidos», pero para un programa convencional son palabras distintas pero semánticamente son lo mismo. Otro ejemplo puede ser la informalidad que presenta el lenguaje natural, en el cual «Productos» «producto» «productoss» son exactamente la misma palabra si se realiza un proceso de normalización adecuado.

Para concluir, el proceso de normalización en cada programa es distinto ya que los requerimientos de cada procesamiento puede requerir que no se eliminen ciertos elementos para obtener mejores resultados. A continuación hay un ejemplo de cómo puede ser un proceso de normalización convencional:

Sentencia: «*Necesitamos usuarios Administradores para el sistemaaa!!!*»

Sentencia post-normalización: «necesitamos usuarios administradores para el sistema»

Stemming

El Stemming es una técnica para eliminar afijos de una palabra, teniendo como resultado la raíz «stem» de la palabra. Por ejemplo, la raíz de las palabras agregar o agregando es «agreg», un buen algoritmo de stemming sabe que el sufijo «ando» o «ar» puede eliminarse. Stemming es más comúnmente utilizado por los motores de búsqueda para indexar palabras. En lugar de almacenar todas las formas de una palabra, un motor de búsqueda puede almacenar sólo las raíces obtenidas de un Stemming, lo que reduce en gran medida el tamaño del índice mientras que aumenta la precisión de recuperación. [3].

Sentencia: «*Requerir Requiriendo Requerimos*»

Sentencia post-stemming: «requer requer requer»

Lemmatization

La lematización es muy similar a la técnica de Stemming, pero es más parecida a la sustitución de sinónimos. Un lemma es una palabra raíz, en oposición a un raíz «stem». Entonces, a diferencia del Stemming, siempre te queda una palabra válida que significa lo mismo o que es el origen de dicha palabra. Está implementado a través de algoritmos de aprendizaje automático que derivan a una palabra raíz a través de información aprendida anteriormente. Es un proceso que consume muchos más recursos que el Stemming, pero cada uno se utiliza para distintos enfoques, cuando la forma de la palabra importa en el resultado final, un Lemmatizador suele ser útil [3].

Sentencia: *«Ellos se registrarán y comprarán productos.»*

Sentencia post-lemmatization: *«Ellos se registrar y comprar productos.»*

2.2.5. Tokenización

La tokenización es una de las primeras técnicas utilizadas de PNL para estructurar textos. La tokenización es el proceso de tokenizar o dividir una cadena de texto en una lista de tokens. Uno puede pensar en el token como partes, como por ejemplo una palabra es un token en una oración, y una oración es un token en un párrafo. Por lo general cualquier herramienta de procesamiento de lenguaje natural, contiene métodos para tokenizar una estructura de texto. Adicionalmente muchos de ellos realizan un procesamiento más profundo sobre cada token, por ejemplo en los tokens de una oración, se puede encontrar con que la herramienta te busca la definición de la misma, busca sinónimos, antónimos, o relaciones entre palabras. Los tokenizadores están lejos de ser algo trivial, debido a la variedad de palabras que existen o los criterios que existen para dividir lógicamente un párrafo en sentencias. Es decir, existen palabras que tienen una complejidad más allá de estar separada por espacios en blanco en una oración, como por ejemplo la palabra «hispano-americano», dependiendo del tokenizador que se utilice esto puede desembocar en dos tokens o en uno. Luego para las sentencias también existen criterios que separarán una oración, por ejemplo por el uso del signo de exclamación, y otros solo separarán a través de puntos seguidos o puntos y aparte [3]. Entonces es importante saber definir qué tipo de herramientas se utiliza para cada parte del procesamiento. A continuación se deja un ejemplo de cómo se tokeniza una oración, donde cada división entre cada palabra significa un token de la oración:

Oración: *«El usuario registrado compra productos»*

Tokenización:

El	usuario	registrado	compra	productos
----	---------	------------	--------	-----------

2.2.6. Bigramas y trigramas

Se llama n -grama a una subsecuencia de n elementos consecutivos en una secuencia dada. En el caso del lenguaje natural los elementos son las palabras y los más utilizados en el área son n -gramas con $n=2$ y $n=3$, denominados bigramas y trigramas respectivamente. Los bigramas y trigramas son frecuentemente utilizados para buscar colocaciones en un texto. Las colocaciones son dos o más palabras que aparecen frecuentemente juntas, como por ejemplo «Buenos Aires» [3]. Adicionalmente las colocaciones se utilizan para identificar elementos que se repiten continuamente, y que dependiendo de su recurrencia se puede inferir una cierta relevancia dentro de un texto.

2.2.7. Parts of Speech Tagging (POS Tagging)

El Parts of Speech (POS) significa asignar clases gramaticales a cada una de las palabras de una oración de lenguaje natural. Asignar una etiqueta POS a cada palabra de un texto de manera manual puede llevar mucho tiempo, lo que da como resultado la existencia de varios enfoques para automatizar el trabajo. Por lo tanto, el etiquetado POS automatizado es una técnica para automatizar el proceso de anotación de categorías léxicas. El proceso toma una palabra o una oración como entrada, asigna una etiqueta POS a la palabra o a cada palabra en la oración y produce el texto etiquetado como salida. La importancia de estas etiquetas es la gran cantidad de información que dan sobre una palabra y sus vecinos. El etiquetado de POS se puede utilizar en aplicaciones de Texto a voz, recuperación de información, análisis, extracción de información, investigación lingüística para corpus y también se puede utilizar como un paso intermedio para tareas de nivel superior de PLN, como análisis, análisis semántico, traducción y muchos más [17], lo que hace que el etiquetado POS sea una función necesaria para cualquier aplicación basada en PLN. El etiquetado POS tiene su importancia en el procesamiento posterior de texto, como el análisis, ya que facilita el procesamiento al adjuntar una clase a una palabra, y también se usa ampliamente para el análisis de texto lingüístico. El etiquetado POS es una tarea de desambiguar palabras, porque intrínsecamente son ambiguas debido a que dependen directamente del contexto en el cual están empleadas y como se las usa. En la figura 2.2 se puede observar el procesamiento que realiza una herramienta de POSTagging convencional.

Como se ve en el ejemplo de la figura POS Tagging 2.2, una herramienta a través de distintas técnicas como pueden ser árboles de decisión o redes neuronales etiquetan

Los	chicos	estudian	en	la	escuela
DET	NOUN	VERB	ADP	DET	NOUN

Figura 2.2: Ejemplo de POS Tagging

de acuerdo al contexto de la oración distintas clases gramaticales a las palabras de una sentencia. Dependiendo de la herramienta a veces la nomenclatura con la que etiquetan puede variar, sin embargo existe una referencia para seguir, que es el Esquema de Universal Dependencies. Universal Dependencies es un framework para tener una anotación consistente de gramática en diferentes idiomas. En el cual contiene etiquetas para generalizar verbos, adjetivos, sustantivos, adverbios, y demás clases que son transversales en la mayoría de los idiomas [19].

2.2.8. Dependencias gramaticales

A su vez, como hay procesos que realizan una clasificación entre las palabras de una oración, existe una clasificación para las relaciones entre esas palabras. Las cuales son llamadas relaciones gramaticales, las cuales expresan a través de etiquetas, cual es la relación que existe entre dos palabras. Los argumentos de estas relaciones consisten en un elemento de cabecera y un elemento dependiente [17]. Como sucede en el POS Tagging, las relaciones de dependencias también siguen la referencia del Esquema de Universal Dependencies [19].

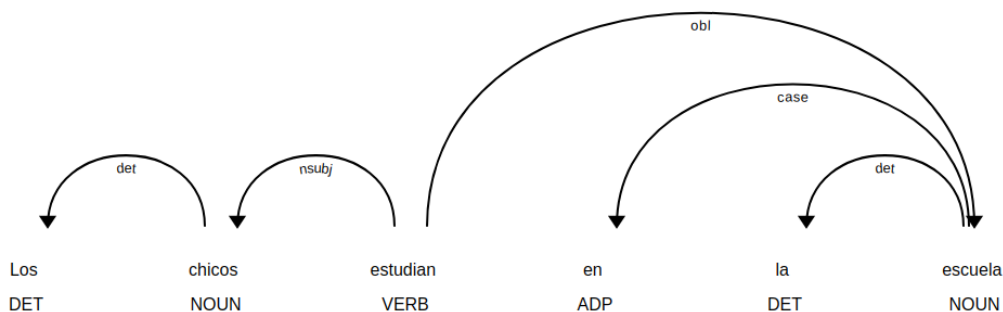


Figura 2.3: Ejemplo de Dependencias

Como se puede ver en la imagen 2.3, se puede observar como la palabra «chicos», representada por tag de NOUN (sustantivo) es el sujeto nominal (nsubj) del verbo

(VERB) «estudian» y la «escuela» es el objeto principal en la que el verbo interactúa a través de la relación OBJ, que representa el objeto directo. Entonces, el desafío para darle un sentido a estas dependencias es poder encontrar automáticamente ciertas relaciones que sirvan para dar un contexto o para poder encontrar información útil en estas relaciones.

2.2.9. Conclusión

La conformación intrínseca de un documentos LEL, que cuenta con sus cuatros respectivos símbolos, sujeto, objeto, verbo y estados, se ve altamente relacionado con las categorías existentes en el lenguaje natural del español por lo tanto también se verá reflejado en la manera de clasificar y etiquetar a las palabras por parte del POS Tagging que se puede realizar con las herramientas. Adicionalmente la posibilidad de poder relacionar esos símbolos con las relaciones de dependencias en una sentencias también nos da un marco para trabajar y para encontrar relaciones entre los símbolos dentro de un texto.

2.3 Aprendizaje automático

2.3.1. Introducción

El aprendizaje automático representa un gran campo de utilidad en la informática, estadística, probabilidad, inteligencia artificial, psicología, neurobiología y muchas otras disciplinas. El aprendizaje automático se trata de crear algoritmos que permitan que la computadora aprenda a identificar atributos o información de acuerdo a una entrada de datos. El aprendizaje es un proceso de encontrar regularidades estadísticas u otros patrones de datos [20]. Con el aprendizaje automático, los problemas se pueden resolver simplemente construyendo un modelo que sea una buena representación de un conjunto de datos seleccionado. Los algoritmos de aprendizaje automático se crean para poder representar el enfoque humano de aprender alguna tarea. Hoy en día a diferencia de hace algunos años, muchos de los algoritmos de aprendizaje automático se han desarrollado, actualizado y mejorado, y el desarrollo reciente en aprendizaje automático se convierte en la capacidad de aplicar automáticamente una variedad de cálculos matemáticos complejos a grandes datos, que calcula los resultados mucho más rápido. Se utiliza el aprendizaje automático en aplicaciones para reconocer patrones, aprender de la experiencia, abstraer nueva información de los datos u optimizar la precisión y eficiencia de su procesamiento y salida. Los algoritmos de aprendizaje automático pueden ser supervisado o no-supervisados aunque hay autores que también cuentan con otras clasificaciones [21], la mayoría de los artículos

los reorganiza en algoritmos supervisados o no supervisados. La principal diferencia que hay entre estas dos clases es la existencia de etiquetado en la información con la que aprenden estos algoritmos. En las siguientes secciones se explica con detalles la diferencia y la similitudes entre ellos.

2.3.2. Aprendizaje automático supervisado

Los algoritmos de aprendizaje automático supervisados intentan predecir y clasificar un atributo predeterminado a partir de una entrada de información. Su precisión y su error de clasificación, también con otras medidas de performance son dependientes de la cantidad de atributos predeterminados que la maquina fue capaz de predecir o clasificar correctamente. En el aprendizaje supervisado, los algoritmos trabajan con datos etiquetados (labeled data), con la cual generan funciones que dadas variables de entrada (input data), les asigna una etiqueta de salida adecuada. El algoritmo se entrena a partir de datos etiquetados y así aprende a relacionar que inferencia tienen los datos de entrada con la etiqueta asignada, de esta manera son capaces de asignar una etiqueta de salida a un nuevo valor, es decir, predice el valor de salida. [22].

2.3.3. Aprendizaje automático no-supervisado

Por el lado de los algoritmos de aprendizaje automático no-supervisado implica reconocimientos de patrones sin tener la necesidad de tener un atributo como objetivo. Estos algoritmos pueden ser adecuados para identificar información para luego etiquetarla y utilizarla para implementar tareas de aprendizaje supervisadas [22]. El aprendizaje no supervisado tiene lugar cuando no se dispone de datos etiquetados para el entrenamiento. Sólo conocemos los datos de entrada, pero no existen datos de salida que correspondan a una determinada entrada. Por lo tanto, a partir de cierta información y la estructura de la misma, intenta encontrar algún tipo de organización que simplifique el análisis.

2.4 Categorización de texto

2.4.1. Introducción

La clasificación de objetos o elementos se encuentra en el día a día de la inteligencia humana. Decidir qué letra, palabra o imagen se ha presentado a nuestros sentidos, reconocer rostros o voces, clasificar un correo de email, asignar ciertas calificaciones a tareas. Todos estos son ejemplos de asignación de una categoría a una entrada de datos. Es decir que todo el tiempo somos capaces a través de nuestros sentidos de

poder clasificar a un objeto en particular[17]. Muchas tareas de procesamiento del lenguaje natural implican clasificación, de esta manera a partir de información de tipo texto es posible asignarle una clase específica. En esta sección se presenta el algoritmo de Bayes, el cual sirve para la categorización de texto, que es la tarea de asignar una etiqueta o categoría a una entrada de tipo texto.

2.4.2. Definición

Esta tesina se enfocó en una tarea común de categorización de texto. La categorización de texto puede ser la orientación positiva, negativa u otra que una persona expresa hacia algún objeto. Una revisión de una película, libro o producto en la web expresa el sentimiento del autor hacia el producto, mientras que un texto de una revista o político expresa el sentimiento hacia un candidato o acción política. La versión más simple acerca de la clasificación de texto es una tarea de clasificación binaria, y las palabras utilizadas durante una opinión suelen ser determinantes importantes para obtener esta clasificación[17]. Por lo tanto el objetivo de la clasificación es tomar una observación de un elemento, medir sus características y de esta manera poder clasificar la observación en alguna clase perteneciente a un conjunto de clases definidas anteriormente. Este tipo de clasificación por lo general se realiza a través de algoritmos de aprendizaje automático supervisado, los cuales tienen un conjunto de datos de observaciones de entrada, cada uno asociado con alguna salida correcta. El objetivo del algoritmo es aprender a mapear desde una nueva observación a una salida correcta. De esta manera los clasificadores que sirven para entradas de tipo de texto, aprenden a partir de las palabras que se utilizan y de las etiquetas asignadas, es decir que las palabras tienen distintas preponderancias que se relacionan con las categorías finales.

2.4.3. Ponderación de las palabras

Para representar un documento a la hora de utilizar un algoritmo de aprendizaje, podemos representarlo como un vector d de elementos.

$$d = (p_1, p_2, \dots, p_n)$$

Donde p_i representa el peso del término i del documento [23], para calcular estos pesos existen varios enfoques, donde principalmente se dividen en dos ramas:

- Aquellos que calculan el peso de las palabras por la cantidad de ocurrencias que tengan en un documento, por ejemplo la Ponderación de Frecuencia de Término (TF por sus siglas en inglés).

- Y por otro lado aquellos que determinan que si hay palabras que aparecen frecuentemente en todas las muestras, tiende a bajarle la ponderación de las mismas, este tipo de caso se lo puede conocer como Ponderación de Frecuencia de Término \times Frecuencia de Documento Inverso (TF \times IDF por sus siglas en inglés).

Ponderación de Frecuencia de Término (TF)

La ponderación de Frecuencia de Término (Term Frequency) es una manera simple de ponderar palabras de un documento. En este método el peso de una palabra en un documento es igual al número que aparece este término en el documento [23].

$$p_i = tf_i \quad (2.1)$$

Donde tf_i representa la frecuencia del término i en un documento específico.

Ponderación de Frecuencia de Término \times Frecuencia de Documento Inverso (TF \times IDF)

La ponderación de Ponderación de Frecuencia de Término \times Frecuencia de Documento Inverso (Term Frequency \times Inverse Document Frequency) es una de las ponderaciones más usadas para determinar el peso de los términos de un documento [23]. En este enfoque, el peso de un término i en un documento es asignado proporcionalmente a la cantidad de veces que un término aparece en el documento e inversamente proporcional al número de documentos del corpus en el que aparece este término. Es decir que tiene en cuenta que si una palabra aparece en la mayoría de los documentos de un corpus, es probable que esta palabra no tenga un inferencia importante en una categoría de un documento, por ejemplo en un libro de informática, la palabra programación aparecerá múltiples veces, pero si procesamos 200 libros de otras áreas, este tipo de ponderación, preponderará a la palabra programación en el documento de informática. A continuación se encuentra la expresión al peso de un término i de un documento.

$$p_i = tf_i \times \log\left(\frac{N}{n_i}\right) \quad (2.2)$$

Donde tf_i es la frecuencia del término, N la cantidad de documentos y n_i la cantidad de documentos donde el término aparece.

La comparación entre el enfoque de TF y TF \times IDF parece bastante obvia debido a la simpleza en la que resuelve la ponderación de TF, el enfoque TF \times IDF resuelve de una manera más real el peso de cada palabra en los documentos de un corpus, obteniendo mejores resultados a la hora utilizar un algoritmo de aprendizaje automático.

2.4.4. Enfoque Naive Bayes

El clasificador ingenuo de Bayes (Naive Bayes) es un modelo probabilístico que dado un documento utiliza las probabilidades conjuntas de cada término de la muestra para calcular la probabilidad condicional de todas las clases $c \in C$ [17] [23]. Para esta tesina se hizo utilizó este enfoque utilizando el Modelo Multinomial de Naive Bayes que utiliza la regla de Bayes que permite dividir cualquier probabilidad condicional dentro de tres probabilidades [17]:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \quad (2.3)$$

Pero calcular la probabilidad condicional de que ocurra un documento dado cada uno de los términos del mismo puede ser bastante costoso más cuando hablamos de un gran número de parámetros y un amplio corpus de información para que los algoritmos de aprendizaje automático aprendan. Por lo tanto se realizan dos suposiciones simplificadoras para solucionar este problema. La primera es que determina que la muestra o el documento son una bolsa de palabras, esto quiere decir que realmente no importa el orden en el que las palabras están ordenadas dentro del documento, sino del peso que tienen las mismas en una sentencia. La segunda suposición es que cada elemento de un documento es independiente del otro, lo cual simplifica y aumenta las operaciones computacionales comparados con otros clasificadores no ingenuos.

A partir de las suposiciones vistas, en los siguientes párrafos se explica cómo se transforma la regla de bayes 2.3 a partir de estas suposiciones y cómo influye en la complejidad de las ecuaciones. En una primera instancia si estamos hablando de un documento y un conjunto de categorías al cual ese documento puede pertenecer, la regla de bayes se puede expresar de la siguiente manera:

$$P(c_j|d) = \frac{P(d|c_j)P(c_j)}{P(d)} \quad (2.4)$$

Donde c_j es la probabilidad de que la categoría j ocurra dado un documento d_i . Entonces para calcular la categoría estimada de un nuevo documento, se debe calcular la probabilidad de que ocurran todas las clases $c \in C$. Un clasificador ingenuo bayesiano elige aquella categoría \hat{c} que tenga la más alta probabilidad de ocurrir. Para calcular $P(c_j|d)$, tanto $P(c_j)$ como $P(d|j)$ tiene que ser estimados a partir de los conjuntos de documentos para entrenar el modelo. Si analizamos la ecuación 2.4 podemos concluir que $P(d)$ va hacer igual para todas las categorías donde estemos calculando la probabilidad de que ocurra dado este documento, por lo tanto podemos eliminar este factor de la ecuación. La probabilidad de que una categoría ocurra, es decir $P(c_j)$, puede ser estimada con los documentos del conjunto de entrenamiento de la siguiente manera:

$$P(\hat{c}_j) = \frac{\sum_{i=1}^N y(d_i, c_j)}{N} \quad (2.5)$$

Donde N es la cantidad de documentos del conjunto de entrenamiento, y la función y se compone de la siguiente manera:

$$f(d_i, c_j) = \begin{cases} 1 & d_i \in c_j \\ 0 & d_i \notin c_j \end{cases} \quad (2.6)$$

Es decir que calcula la probabilidad de que ocurra una clase dados los documentos de entrenamiento que pertenecen a dicha clase. Por último se encuentra el parámetro $P(d|c_j)$ que según el modelo ingenuo bayesiano que se utilice se calcula de una manera distinta. En la próxima sección se ve cómo se calcula este parámetro en un modelo multinomial de naive bayes.

Modelo Multinomial de Naive Bayes

En el modelo multinomial un documento d es una secuencia ordenada de términos pertenecientes al espacio de términos T . La suposición que realiza el modelo ingenuo de Bayes es que la probabilidad de que ocurra cada término de un documento es independiente del contexto del término, de la posición y la longitud del documento. Entonces, cada documento d_i es representado por una distribución multinomial de los términos con el número de eventos independientes igual a la longitud del documento d_i . Por lo tanto para calcular la probabilidad de que ocurra un documento d_i dada su categoría c_j puede ser aproximada de la siguiente manera[17]:

$$P(d_j|c_j) \approx \prod_{i=1}^{|d_i|} P(w_i|c_j) \quad (2.7)$$

Donde $|d_i|$ es el número de términos en el documento d_i y w_i es el término i -ésimo que aparece en el documento d_i . Por lo tanto la estimación de $P(d_i)$ es reducida a estimar cada $P(w_i|c_j)$ independientemente. La siguiente estimación Bayesiana es usada para $P(w_i|c_j)$ [23]:

$$\hat{P}(w_i|c_j) = \frac{1 + TF(w_i, c_j)}{|T| + \sum_{w_k \in T} TF(w_k, c_j)} \quad (2.8)$$

Aquí, $TF(w_i, c_j)$ es el número total de veces que el término w_i ocurre en el conjunto de documentos de entrenamiento pertenecientes a la categoría c_j . La sumatoria de términos en el denominador está para el número total de ocurrencias en el conjunto de documentos de entrenamientos perteneciente a la categoría c_j . Y por último se encuentra el espacio de términos T [23].

2.4.5. Conclusión

Con lo visto en las características de los algoritmos de aprendizaje automático supervisados y el clasificador ingenuo bayesiano, se llega a la conclusión que con la finalidad de poder encontrar características y componentes de un documento LEL, con estos tipos de clasificadores se puede modelar un clasificador que con aprendizaje guiado por expertos en LEL, pueda indicar si un componente, compuesto por palabras/símbolos, cumple con lo necesario para ser parte de un LEL.

Capítulo 3

Estado del Arte

3.1 Introducción

Para desarrollar la estructura general del proyecto de esta tesina, se llevaron a cabo varias investigaciones para ver el entorno en el cual se encontraba este tópico y de esta manera analizar cuales podrían ser las contribuciones que este trabajo final podía aportar.

Dentro de los objetivos de la investigación del estado del arte se encontraba buscar trabajo similares al propuesto, en este caso debido a que la arista creada por un documento LEL es relativamente nueva no se han encontrado artículos o enfoques propongan una derivación de este documento a partir de texto en lenguaje natural o requerimientos escritos en lenguaje natural. Por lo tanto se procedió a buscar otro tipos de derivaciones que tengan un factor en común con este documento LEL. Para ello se realizó un enfoque sobre 4 modelos utilizados en el proceso de ingeniería de software, el primero es el modelo de diagrama de clases UML, que representa la estructura de un sistema mostrando las clases del mismo, sus atributos, operaciones (o métodos), y las relaciones entre los objetos. El segundo son los modelos de casos de uso UML que consisten en identificar actores, ya sea personas o sistemas, e identificar los diferentes casos de usos en los que estos actores están involucrados. El tercer modelo es el Modelo de Entidad Relación, que se utiliza para describir como esta conformada una base de datos, con toda la complejidad que esto implica. Y por último, aunque no es un modelo, es el enfoque más parecido a un documento LEL, ya que se trata de la extracción de términos conformando glosarios sobre lenguaje natural. Se lo considera un termino a todo concepto relevante de un documento que se utilice para definir el dominio del mismo, los términos por lo general suelen ser sustantivos pero también se pueden ver representados como verbos.

Es importante aclarar, que los artículos encontrados durante esta investigación trabajan sobre el idioma inglés, y las estructuras gramaticales de este idioma, por lo tanto resultó un desafío, trasladar ciertas ideas o conceptos al idioma español teniendo en cuenta que hay una diferencia desfavorable con respecto a las reglas existentes en el español para conformar sentencias. No se han encontrado artículos en español que realmente ayuden a comprender ciertas problemáticas tanto a nivel de implementación o a nivel conceptual de cómo crear una herramienta como la propuesta. Por lo tanto a través de este capítulo se analizará cada uno de estos enfoques, haciendo énfasis en la complejidad que han alcanzado a lo largo de los años, los resultados y conclusiones de cada investigación.

3.2 Derivaciones

3.2.1. Diagrama de clases

Definición

Los diagramas de clases, por lo general conocidos como diagramas de clases UML, son diseños estáticos que sirven para representar gráficamente softwares orientados a objetos, donde se tiene en cuenta la representación de las clases, sus métodos, sus atributos, así mismo sirven para representar la relación que tienen estas clases entre sí. La teoría de orientación a objetos define un número de relaciones entre objeto que incluyen a: agregación, composición, herencia, y asociación, donde cada una de estas relaciones implica una complejidad distinta en cuanto a su significado, estas relaciones también son contempladas en un diagrama UML [24]. Por lo tanto, estos diagramas implican una gran complejidad al tener que representar clases, comportamiento y adicionalmente tener en cuenta la relación o la complejidad existente entre cada componente del diagrama de clases, que muchas veces es una cuestión subjetiva que define el analista encargado de realizar este diagrama. En las próximas secciones se aborda la evolución de complejidad en el intento de conseguir modelos UML u Orientados a objetos de manera automática, y la incorporación de ciertas tecnologías a medida que se va complejizando el proceso.

Investigaciones

Los primeros trabajos relacionados con la extracción de modelos UML, nos lleva al año 2006, donde los enfoques se basaron en extraer estos modelos a partir de requerimientos escritos en lenguaje natural (RLN) utilizando técnicas básicas de procesamiento de lenguaje natural (PLN). Uno de los primeros trabajos interesantes fue

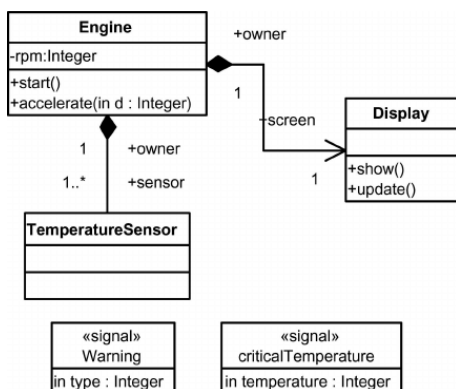


Figura 3.1: Ejemplo de diagrama de clases UML

desarrollado por los autores Anandha Mala y G.V Uma, que propusieron extraer elementos orientados a objetos a partir de un documento de requerimientos (SRS). El núcleo de este trabajo se basó principalmente en la utilización de técnicas de procesamiento de lenguaje natural, donde el módulo más importante se encargaba del preprocesamiento de la información no estructurada a través de técnicas conocidas como tokenización de sentencias, POS Tagging, y Chunkers para dividir sintácticamente una entrada de tipo texto. Una vez que la herramienta obtenía cierta información a partir de los chunkers y el POS Tagging, la misma tenía en cuenta reglas heurísticas para la obtención de clases, atributos de clases, y métodos realizando una derivación directa a partir de los elementos sintácticos pertenecientes al idioma. Por ejemplo, trasladaba sustantivos como clases, y verbos como métodos, adicionando reglas que se debían cumplir para que ocurran estas derivaciones. Más allá del modelo poco complejo que planteó esta herramienta, la misma propuso la utilización de un módulo interesante en el idioma inglés, que se basó en resolver las referencias de pronombres, esto significa, que el módulo era capaz de identificar y relacionar que sujetos se está haciendo referencia implícitamente en alguna parte del texto. Según los descrito en la conclusión del proyecto, la herramienta fue capaz identificar todas las clases y métodos que estaban definidos, pero al contar con un procesamiento poco complejo, obtuvo más elementos de los que debía y donde se tuvo que utilizar la intervención humana para filtrarlos [25].

Con el pasos de los años y la evolución de investigaciones y tecnologías, el procesamiento necesario para la extracción de elementos UML a través de PLN apuntaba a una mayor complejidad y a la incorporación de otras técnicas, tales como la utilización de stemmers o lemmers, de reglas heurísticas más complejas, y la incorporación de nuevas tecnologías para la identificación de ciertos patrones que sobrepasaban al PLN convencional. En este punto en el año 2010, el autor Rodina Ahmad, propone un

enfoque de extracción automático de diagramas de clases a partir de requerimientos textuales utilizando como núcleo central la utilización de PLN y reglas heurísticas para obtener todos los elementos posibles de un diagrama de clase UML. Por lo tanto, esta herramienta también utilizó las técnicas básicas para encontrar elementos sintácticos en un texto pero se diferenció principalmente de los primeros artículos de derivaciones, por incorporar la búsqueda de relaciones entre las clases de un sistema, lo cual connota una complejidad adicional importante, que sumado a las clases, atributos y métodos conforman todos los elementos de un diagrama de clases UML. Y también, por la incorporación de dos componentes relevantes, que fueron la utilización de corpus de información, como WordNet, que permitió analizar propiedades semánticas de las palabras tales como jerarquías y relaciones, que se las pudo utilizar para unificar semánticamente palabras que puedan llegar a significar lo mismo. Y por último, la incorporación de ontologías de dominio, que se utilizó para refinar los resultados de las reglas heurísticas planteadas, es decir que se utilizó el conocimiento previo dado un dominio particular para identificar elementos tales como clases y métodos [26]. Un trabajo similar al descrito es el de los autores Letsholo, Zhao, Chioasca, que también propusieron una herramienta para extraer un diagrama de clases UML a partir de requerimientos escritos en LN, a través del mismo núcleo de PLN, con la diferencia que identificaron atributos y métodos de una clase mediante la utilización de las dependencias sintácticas que existían entre las palabras. Es decir que utilizaron funcionalidades de una librería, en este caso Stanford Parser, que les indicaba que inferencias sintácticas tenía una palabra sobre otras. Sin embargo este trabajo, solamente se basó en reglas heurísticas a partir de estas dependencias para identificar los elementos, lo cual impactó directamente sobre la precisión de los resultados (71 % de recall, 60 % de precisión, y 138 % de overspecification) [27].

De esta manera también hubo otros trabajos que propusieron sistemas automáticos basándose solamente en reglas heurísticas y en PLN, lo cual ya empezaba a denotar que estas herramientas necesitaban una complejidad adicional para llegar a mejores resultados. Es el caso de los autores Tripathy y Rath que a principio del 2014 proponían una herramienta bastante rudimentaria la cual se basaba exclusivamente en reglas heurísticas donde no hicieron pruebas de precisión de la misma [28], pero que a fines del 2014 realizaban un importante avance en la complejidad de los módulos para extraer estos diagramas. En este nuevo artículo describen un flujo de trabajo mejor segmentado y a su vez más complejo. Donde utilizaron el núcleo de PLN para extraer los elementos sintácticos más importantes, incorporaron Stemmers para unificar los sustantivos y eliminar palabras duplicadas dentro del texto, y agregaron un módulo estadístico en el cual utilizaban la frecuencia de cada sustantivo encontrado dentro del texto para determinar si era una clase candidata. De esta manera se incorporó otra arista a tener en cuenta para seleccionar elementos «importantes» dentro de un texto.

Ya que el resto de los elementos de este diagrama dependen de las clases, la precisión de los resultados está directamente relacionado con la precisión de detectar las clases del sistema. Luego esta herramienta propuesta realiza un procesamiento para identificar el resto de los elementos a través de reglas heurísticas, para luego graficar el diagrama. Este trabajo tiene un 100 % de recall, 93 % de precisión y un 9 % de overspecificacion, por lo tanto el enfoque estadístico para identificar las clases del sistema fue totalmente exitoso. Es de gran relevancia, tener en cuenta las conclusiones que hace este trabajo, ya que enuncia que para mejorar aún más los resultados la utilización de técnicas de machine learning iba a ser necesaria [29]. Aunque todo apuntaba a una incorporación de tecnologías para realizar estas derivaciones en los años siguientes a este último artículo no se han encontrado grandes cambios relacionados con las implementación de estas derivaciones, es el caso del artículo de Sharma, Srivastava y Biswas (2015), los cuales proponen otra herramienta que consigue estas derivaciones a través de reglas heurísticas [30]. Sin embargo se encontró un artículo escrito por Narawita y Vidanage, del año 2017, que realizan un cambio de importante en los módulos utilizados para implementar esta derivación. La cual esta conformada de tres componentes principales la primera es PLN que realiza un procesamiento similar a las demás herramientas para identificar elementos del texto, luego realiza un enfoque orientado a reglas heurísticas para identificar los elementos del diagrama, en este caso clases, atributos y métodos. Pero utiliza un nuevo módulo que es el de algoritmos de aprendizaje automático supervisado de tipo clasificador, el cual se utilizó para filtrar y mejorar la salida de los elementos candidatos que surgieron de las reglas. Para realizar el aprendizaje de este clasificador, se utilizó corpus de clases y acciones de UML validos, de esta manera el algoritmo es capaz de identificar que elementos tienen más probabilidad de ser una clase en un diagrama. Luego la herramienta utiliza un factor importante, que es el uso de la intervención humana para seguir filtrando y mejorando los resultados conseguidos entre los tres módulos previamente mencionados. Por último vuelve a utilizar el módulo de aprendizaje automático para conseguir las relaciones existentes entre las clases del diagramas, para completar todos los elementos del diagrama. Este artículo menciona que consiguieron aproximadamente un 70 % de precisión en la obtención de los elementos. Esto dispara un debate el cual parece sugerir que mientras más complejo sea una herramienta de estas características, menos precisión se va conseguir por la cantidad de filtros que se van adicionando, y que las herramientas más simples suelen conseguir todos los elementos pero con sobre-especificación general, es decir que van a encontrar elementos que no se encuentran relacionados con el diagrama en cuestión [31].

3.2.2. Casos de usos

Definición

Una definición precisa para representar un caso de uso, puede ser que, un caso de uso describe el comportamiento del sistema bajo condiciones cuando el sistema responde a un requisito desde uno de sus stakeholders, llamado actor primario. Por lo general, los casos de uso son utilizados de ayuda para etapas tempranas de desarrollo, clasificar requerimientos, y guiar en el resto del proceso de desarrollo. Entonces, estos casos de usos terminan describiendo que es lo que el sistema debe hacer desde la perspectiva del usuario. Estos comportamientos son capturados a través de diagramas y son independientes de los detalles de implementación. Los componentes principales de un caso de uso se dividen en dos, el primero es el actor, que son los usuarios humanos que utilizan el sistema, y por el otro lado los casos de uso, que sería el componente que describe el comportamiento de una acción, estos elementos son relacionados a través de asociaciones, que son representados con una línea recta que une al actor y al caso, teniendo connotación que el actor es el que realiza el comportamiento descrito por el caso de uso. Luego existen relaciones entre componentes, tal como la generalización de actores y casos de usos utilizados para representar jerarquías entre los elementos. Por último existen las relaciones de extensión e inclusión entre casos de usos. Por ejemplo, en el caso de extensión, se utiliza para indicar que un caso de uso llama a otro para circunstancias especiales del mismo para completar su objetivo, y se utiliza la inclusión cuando un caso de uso depende de otro para conseguir su objetivo [32].

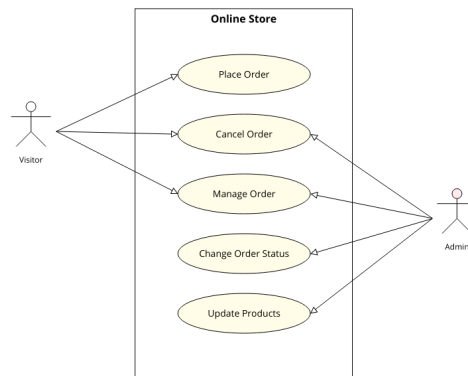


Figura 3.2: Ejemplo de casos de uso

Investigaciones

Los desarrollos relacionados con casos de usos comenzaron prácticamente de manera paralela con las investigaciones de diagramas de clases debido a que la información necesaria para generar de manera automática casos de usos sirve luego para generar diagramas de clases y viceversa. Sin embargo el caso de uso suele realizarse anteriormente que el diagrama de clases y los diagramas de clases suelen ser más complejos con respecto a la información que se representa. Aunque ambos diagramas tengan propósitos y enfoques muy distintos esto no impide obtener y encontrar una relación directa entre los elementos de casos de uso y diagramas de clases, es decir que esto posibilita que encontrar información para uno puede significar encontrar información para el otro. Dicho esto, uno de las primeras investigaciones que se ha encontrado acerca de casos de usos surge en el año 2009, donde investigadores de la Universidad de Limerick, proponen una herramienta llamada UMGAR (UML Model Generator from Analysis of Requirements), la cual tenía como objetivo generar modelos UML, en el cual se contemplan los casos de usos, a través del procesamiento de requerimientos escritos en lenguaje natural (RLN). Para realizar dicho procesamiento, se utilizaron dos módulos importantes, el primero es transversal a todas las investigaciones de este capítulo, que son las técnicas y el procesamiento de lenguaje natural, tales como tokenización de sentencias y palabras y POS Tagging. Luego para identificar los casos de uso se propuso 8 reglas de reestructuración de sentencias, donde la idea era identificar todas aquellas reglas donde sintácticamente no respete el formato: Sujeto-Predicado-[Objeto], y reescribirlas automáticamente en dicho formato. En caso de que una sentencia no respete el formato y no pueda ser reescrita, la herramienta pedía al usuario una re-escritura de la sentencia, esto colabora a una mayor obtención de información ya que permitía tener mapeado todas las sentencias que se tenga en el texto ingresado. Luego la herramienta realizaba una derivación directa de sujetos y objetos como actores y el predicado como caso de uso. El artículo no muestra resultados de precisión de la herramienta, pero al no incluir de módulos de filtros o estadísticos que eliminen falsos positivos, es probable que en esta etapa de la herramienta, haya habido un alto overspecification [33].

Avanzando en los años, se realiza un gran salto al año 2017, donde tres autores proponen una herramienta automatizada solamente para encontrar diagramas de casos de usos a partir de RLN. La cual propone varios módulos interesantes a tener en cuenta. La primera parte de esta herramienta es similar a la mayoría de las herramientas descritas en este capítulo, la cual consiste en dividir sintácticamente el texto de entrada y generar una estructura para obtener información. Un módulo innovador que se propuso en esta herramienta fue implementar un módulo corrección gramatical y escritura, la cual consistía en detectar si había algún error de escritura o de gra-

mática, y de esta manera reportar un mensaje al usuario para que corrija este error. Adicionalmente, se utilizó un enfoque para analizar semánticamente las oraciones, es decir, analizar e identificar recursos cohesivos tales como referencias, comparación o adición. Por último para identificar los elementos del caso de uso (actores, casos de uso, relaciones) utiliza reglas heurísticas para realizar las correspondientes derivaciones. Los resultados de las evaluaciones resaltan un buen número de precisión con un 80 % y un recall del 96 % en promedio con un total de 5 casos de estudio y 10 documentos en cada caso [34]. Aunque este artículo contiene una estructura similar a las vistas hasta ahora, ese mismo año, investigadores de la Universidad de Siegen (Alemania), proponen una herramienta en la cual la identificación de actores y casos de usos sea a través de algoritmos de aprendizaje automático de tipo supervisado. Por lo cual, tuvieron que enseñar a una algoritmo en particular, en este caso el algoritmo de Naive Bayes, como reconocer casos de uso y actores a través de conocimiento identificado previamente. Por lo tanto, la herramienta contiene dos módulos importantes, PLN para identificar los elementos sintácticos y Machine Learning para identificar los elementos del diagrama. Entra en discusión en este artículo el dominio con el cual se realiza el aprendizaje de la máquina debido a que en distintos dominios se puede tener distintos actores y distintos casos de uso. Por lo tanto el foco con el que se deben realizar las pruebas debe estar relacionado con el corpus de información utilizado para realizar el aprendizaje. Por último el enfoque propuesto tuvo un exactitud del 77 %, un recall del 80 % y una precisión del 70 % [35]. En este mismo marco entra un artículo que se ha hablado en las derivaciones a diagramas de clases, ya que también se utilizó para derivar casos de uso, en el cual utiliza algoritmos de aprendizaje automático supervisados para identificar casos de usos válidos y a su vez identificar las relaciones entre casos de usos y actores, en las que contempla las relaciones de inclusión, extensión y generalización. Abarcando por completo todos los elementos de un diagrama de casos de uso [31]. Siguiendo el hilo de trabajos cronológicamente y relacionado directamente con la complejidad de los mismos, se ha encontrado un artículo más, en el cual se sigue el cambio de tecnologías empleadas en los últimos años para resolver el proceso de conseguir automáticamente casos de uso a partir de lenguaje natural. El artículo de la Universidad de Jordania (2019), propone un flujo de trabajo particular, donde se basan en una metodología de aprendizaje constante, donde no hay filtros automáticos, de lo contrario proponen que se obtengan casos de uso a partir del análisis sintáctico básico de los RLN y a partir de ello, medir la satisfacción del usuario con respecto a estos resultados y con esta información realizar un aprendizaje de un algoritmo clasificador, es decir que la idea principal de este artículo es que la precisión de esta metodología va a estar 100 % relacionada con el uso de la herramienta y el feedback de la misma y de esta manera ir mejorando los resultados [36].

3.2.3. Modelos entidad relación

Definición

Los modelos de entidad relación (MER) son ampliamente usados para diseñar bases de datos y análisis de sistemas. Formalmente, se puede decir que un diagrama o modelo de entidad relación es una representación gráfica del dominio de un problema que se está modelando. Por lo tanto, este modelo asiste al encargado de diseñar la base de datos para identificar la información y las reglas que serán representadas y usadas en una base de datos. Es una representación independiente de la implementación y facilita la comunicación entre los stakeholders y los analistas. Está compuesto por varios elementos, el primero de ellos son las «Entidades» que son los elementos principales de este modelo y son aquellos problemas de dominio donde el usuario tiene que guardar información, luego existen los atributos, que representan propiedades que pueden ser de entidades o de relaciones. Las relaciones es otro elemento importante, la cual tiene el objetivo de asociar entidades y describe una acción de significado entre las mismas. Al igual que en los modelos vistos hasta ahora, el MER también posee varios tipos de relaciones en los que se destacan las relaciones que denotan una conjunción de los datos que se comparten entre las entidades, como las relaciones uno a uno, uno a muchos y por último muchos a muchos. Luego están las relaciones que permiten distinguir tipos de entidades y jerarquizar las mismas, también llamadas relaciones de herencia, las cuales se pueden dividir en dos tipos: Generalización y Especificación [37].

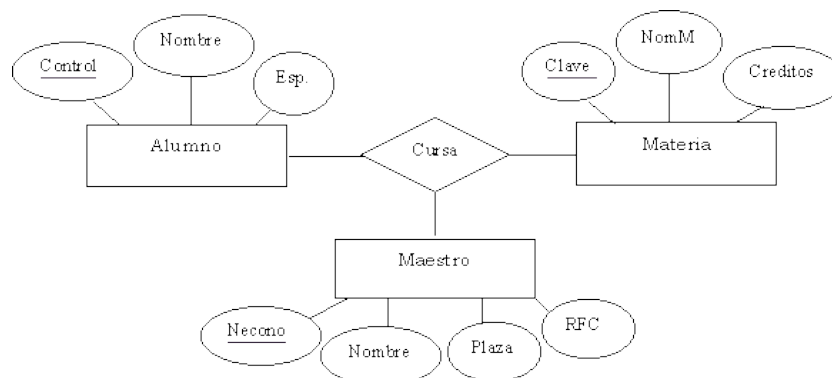


Figura 3.3: Ejemplo de modelo entidad relación

Investigaciones

Debido a que no se han encontrado muchos artículos relacionados a derivaciones de RLN o LN a modelos entidad-relación, esta sección se dividirá en dos partes,

relacionadas por la complejidad de los artículos y adicionalmente por el momento cronológico en el que se propusieron. Los primeros enfoques encontrados se dan en el año 2015 propuestos por la Universidad de Mumbai (India) y la Universidad de Mutah (Jordania), en los cuales se realizó un trabajo bastante similar, donde explican técnicas con las cuales se puede derivar RLN hacia modelos ER. En dichos artículos se tuvo como núcleo central el procesamiento de lenguaje natural para obtener información sintáctica de cada sentencia de un texto. Luego ambos trabajos proponen una derivación directa de los elementos sintácticos encontrados a partir del POS Tagging a los elementos del MER [38] [39]. Ambos coinciden que los sustantivos son posibles entidades candidatas, los verbos posibles relaciones, que para identificar las claves primarias de las entidades se puede buscar adverbios que indiquen una unicidad de un elemento, o que un atributo también podría derivar de un sustantivo siempre y cuando tenga relación sintáctica con las entidades en cuestión. Más allá de las similitudes, los autores de la Universidad Mumbai realizaron una investigación más compleja acerca de las reglas heurísticas a desarrollar para conseguir dichos elementos, también es importante mencionar que dentro de estas reglas contemplan las cardinalidades que se pueden obtener de las relaciones comprendidas entre las entidades.

Avanzando en los artículos encontrados, se llega directamente al año 2018, donde investigadores de la Universidad de Moratuwa, proponen un enfoque compuesto de dos módulos principales para obtener información, donde el módulo de PLN se mantiene al igual que los artículos anteriores para identificar los elementos de un texto dado (RLN), pero agregan un módulo de algoritmos de aprendizaje automático de tipo supervisado para clasificar automáticamente los 4 elementos (Entidades, Relaciones, Atributos y Cardinalidades) de un MER. Este módulo recibe como input las sentencias del texto que fueron procesadas en el módulo anterior, y tiene como output cada elemento identificado en cada categoría. El artículo hace énfasis que los atributos son recuperados con una lógica adicional, que es utilizando ontologías y web mining, esto se debe a que los atributos muchas veces dependen del dominio en el que se está trabajando. Por lo tanto, este enfoque deja de lado absolutamente las reglas heurísticas delegando todo el procesamiento de detección a ML [40], teniendo un 100 % de confianza sobre este módulo. Para validar en cierto punto esta confianza, los investigadores presentan una tabla con los distintos clasificadores de texto usados, donde todos superan el 94 % de precisión con respecto a identificar entidades, pero se destaca el clasificador de Naives Bayes a tener una mejor precisión con respecto a la identificación de atributos y sub-entidades con una precisión mayor al 98 %.

Siguiendo con el año 2018, se ha encontrado un último artículo que se expuso en la Conferencia Internacional de Machine Learning e Ingeniería de datos, donde los investigadores proponen una herramienta que se compone de 3 módulos, donde el primero es el común a todos (PLN), que se utiliza para obtener los elementos

sintácticos del texto. Pero el segundo módulo se diferencia de todos los vistos hasta ahora, ya que para identificar las entidades, atributos y relaciones a partir del POS Tagging, utilizaron bases de datos específicas para cada dominio, es decir que para identificar cada elemento del MER para un dominio en particular, se tuvo que tener en cuenta una base de datos que contenga una tabla por cada tipo de elemento donde especificaba que palabras entraban en esa categoría, por ejemplo dado un sustantivo, la herramienta busca si se encuentra relacionado con una palabra de la tabla entidades que contiene esa base de datos, en caso de que así sea, ese sustantivo pasa a ser una entidad. De esta manera, identifica la categoría de cada elemento del texto utilizando la base de datos prácticamente como una ontología. Y por último, se implementó un módulo interesante que es la detección de pronombres cercanos, es decir dado un pronombre, identificaron cual es el sujeto al que se está haciendo referencia, de esta manera se consigue una flexibilización de las estructuras en la que se escriben las sentencias para obtener información [41].

3.2.4. Glosarios de términos

Definición

Los glosarios de términos consiste en todos los conceptos de dominios especificado en un documento con sus respectivas definiciones. Estos glosarios de términos suelen crearse en una primera instancia y luego capturar los requerimientos utilizando la terminología definida. Un glosario asegura que diferentes stakeholders tengan la misma interpretación de requerimientos y se utilice durante todas las etapas del ciclo de desarrollo de software. Adicionalmente un glosario también pueden ser utilizados para derivar conceptos hacia clases y métodos o cualquier necesidad de una especificación de requerimientos [42].

Investigaciones

No se han encontrado muchas investigaciones en los últimos años, pero los artículos que se han investigado ha servido para tener una noción de las problemáticas que pueden existir en un contexto tan subjetivo que es encontrar los conceptos importantes dentro de un documento o dentro de RLN. El primer artículo encontrado en los últimos 10 años desarrollado por investigadores de la India, proponen un sistema de detección de términos a partir de un documento de requerimientos, en el que dicho sistema está compuesto por tres módulos principales. El primer módulo se basó en el pre-procesamiento del texto, que consistió en detectar las frases nominales de cada sentencia encontrada en el texto y de eliminar palabras que no aporten un valor agregado a la sentencias, también conocidas como stopwords. Una vez pasada la in-

formación por este módulo de estructuración, la herramienta propuso un módulo que determine la unicidad de términos. Este módulo consiste en determinar que elementos son palabras compuestas tales como sustantivos con adjetivos, que unidades están relacionadas por conectores de conjunción como las palabras «y» u «o», y por último identificar los verbos encontrados como posibles términos. El último módulo y más importante de este desarrollo propone filtrar cuales unidades no pueden ser considerados términos. En este caso divide las técnicas utilizadas en dos. Por un lado, utiliza las técnicas lingüísticas donde tiene como objetivo eliminar aquellas unidades donde los sustantivos sean abstractos, verbos auxiliares, o referencias a pronombres. Y por el otro, utiliza técnicas estadísticas donde evalúa si unidades identificadas en etapas anteriores pueden tener alguna ambigüedad en la misma ya sea por un error algorítmico o por una falla en la escritura. En caso de encontrar términos no ambiguos los mismos son utilizados para desambiguar al resto de las unidades. Con la conjunción de estas dos partes, el módulo identifica los términos finales del glosario, donde especifica que tuvieron un 85% de precisión en promedio para detectar las lista de entidades y la lista de acciones que conforman el glosario [42]. Continuando con las investigaciones, en el año 2017 investigadores de la IEEE y colaboradores, proponen una manera de extraer glosarios términos de manera automática y realizar un agrupamiento semántico y sintáctico entre cada uno de estos términos. El enfoque esta compuesto por tres partes, la primera es el núcleo de NLP, donde los autores afirman que el 99% de los términos en glosarios técnicos se encuentran en frases nominales (FN). Por lo tanto para conseguir dichas FN realizan un procesamiento a través de text chunking que permite separar sintácticamente sentencias en forma de arboles, teniendo de manera simple la frase nominal de una sentencia. No utilizan ningún filtro estadístico para obtener términos, sino que cualquier sustantivo encontrado en una frase nominal es considerado un término. Luego realiza un procesamiento para encontrar similitudes sintácticas y semánticas entre sustantivos o FN, para luego poder relacionarlas y agruparlas en lo considerado como un cluster de términos. Este trabajo en particular se basa principalmente en la clusterización de términos, por lo tanto no hay un gran detalle o complejidad basado en la extracción términos más allá de lo explicado [43]. Y por último se encuentra un trabajo desarrollado por investigadores de la Universidad Tecnológica de Berlin (Alemania), que proponen un extracción de glosarios de términos automática a partir de especificaciones de requerimientos de gran escala. Este trabajo para identificar los términos de un glosario, propone un enfoque híbrido entre dos enfoques actuales, el primero es identificar todas aquellas frases nominales, la cual afirman, al igual que el artículo anterior, que allí se encuentran el 99% de los términos relevantes, en este caso utilizan la librería NLTK y los text chunkers que provee dicha herramienta, y el segundo enfoque, en este caso estadístico, trata de que los términos relevantes suelen estar relacionados con la frecuencia en el que se encuentran en un documento, adicionalmente disponen de un corpus con un dominio

totalmente distinto, donde si un término frecuente más en este documento de dominio distinto que en el documento por el cual se encontró, el mismo filtra, ya que esto significaría que el término encontrado no es realmente relevante para este documento. Por último los resultados que pasaron los filtros serían considerados los términos del glosario perteneciente a esa especificación de requerimientos. Los resultados de este trabajo, tienen un 75 % de recall y un 73 % de precisión donde identifican 258 términos de glosarios de un subconjunto de 100 requerimientos [44].

3.3 Resumen de investigaciones

A continuación se describe de manera resumida las investigaciones vistas durante este capítulo, teniendo en cuenta el año de publicación, tecnologías empleadas y resultados obtenidos:

Nombre	Derivación	Tecnologías	Resultados	Año
Automatic Construction of Object Oriented Design Models [UML Diagrams] from Natural Language Requirements Specification	Texto a Modelo UML	Procesamiento de Lenguaje Natural	Cuenta con sobre-especificación en los resultados donde el 12 % de las clases y el 7,4 % de los métodos son eliminadas manualmente.	2006
An Automated Tool for Generating UML Models from Natural Language Requirements	Texto a Caso de usos	Procesamiento de Lenguaje Natural	Incorporación de reglas heurística para identificación de casos de usos y no realiza pruebas de precisión	2009
Class diagram extraction from textual requirements using Natural language processing (NLP) techniques	Texto a Modelo UML	Procesamiento de Lenguaje Natural y Ontologías de dominio	Se encontró con éxito elementos del diagrama, sin realizar pruebas y sin encontrar relaciones entre clases	2010

Nombre	Derivación	Tecnologías	Resultados	Año
Automatic Extraction of Glossary Terms from Natural Language Requirements	Texto a Glosario de términos	Procesamiento de Lenguaje Natural, técnicas estadísticas y lingüísticas	Obtiene términos principalmente desde sustantivos y verbos, no realiza pruebas de precisión.	2013
TRAM: A Tool for Transforming Textual Requirements into Analysis Models	Texto a Modelo UML	Procesamiento de Lenguaje Natural y reglas heurísticas	Obtiene resultados concretos con un 37 % de precisión, 138 % de sobre-especificación y un 71 % de recall	2013
Requirement Analysis using Natural Language Processing	Texto a Modelo UML	Procesamiento de Lenguaje Natural	Obtiene resultados excelentes con un recall del 100 %, precisión del 93 % y sobre-especificación del 9 %	2014
Application of Natural Language Processing in Object Oriented Software Development	Texto a Modelo UML	Procesamiento de Lenguaje Natural	Extracción de clases y métodos solamente sin realizar pruebas de precisión	2014
Generating ER Diagrams from Requirement Specifications Based On Natural Language Processing	Texto a Modelo Entidad Relación (MER)	Procesamiento de Lenguaje Natural	Obtiene elementos simples de un MER, enunciando limitaciones con respecto a los distintos dominios. No realiza pruebas de precisión	2015

Nombre	Derivación	Tecnologías	Resultados	Año
Techniques to automatically generate Entity Relationship Diagram	Texto a Modelo Entidad Relación (MER)	Procesamiento de Lenguaje Natural y reglas heurísticas	Obtiene con éxito la mayoría de los elementos de un MER con la incorporación de reglas. No realiza pruebas de precisión	2015
From Natural Language Requirements to UML Class Diagrams	Texto a Diagrama UML	Procesamiento de Lenguaje Natural y reglas heurísticas	Obtiene con éxito los elementos con una gran sobre-especificación, precisión del 50 % y un recall del 100 %.	2015
Automated Extraction and Clustering of Requirements Glossary Terms	Texto a Glosario de términos	Procesamiento de Lenguaje Natural	Presente mejora con respecto a herramientas previas con precisión del 60 %	2017
An Automated Use Case Diagrams Generator From Natural Language	Texto a Casos de Uso	Procesamiento de Lenguaje Natural	Presenta mejoras con el uso de la interacción de un especialista aunque no posee pruebas de precisión	2017

Nombre	Derivación	Tecnologías	Resultados	Año
UML Generator – Use Case and Class Diagram Generation from Text Requirements	Texto a Casos de Uso y Diagramas de clases	Procesamiento de Lenguaje Natural, reglas heurísticas y Aprendizaje Automático	Una de las primeras herramientas en presentar una combinación entre procesamiento de lenguaje natural y aprendizaje automático, con las cuales logra conseguir resultados aunque sin tomar pruebas de precisión	2017
Automated Use Case Diagram Generation from Textual User Requirement Documents	Texto a Casos de Uso	Procesamiento de Lenguaje Natural y Aprendizaje Automático	Obtienen resultados muy positivos con una exactitud del 77 %, un recall del 80 % y una precisión de 80 % donde deja en responsabilidad de identificar elementos a módulos de Aprendizaje Automático	2017

Nombre	Derivación	Tecnologías	Resultados	Año
Generating Entity Relationship Diagram from Requirement Specification based on NLP	Texto a Modelo Entidad Relación	Procesamiento de Lenguaje Natural y Aprendizaje Automático	Nuevamente dejan en responsabilidad de identificar elementos a módulos de Aprendizaje Automático y consideran que los resultado con respecto a la exactitud de los modelos fueron óptimos.	2018
Automated Generation of E-R Diagram from a Given Text in Natural Language	Texto a Modelo Entidad Relación	Procesamiento de Lenguaje Natural y Aprendizaje Automático	El trabajo se basa principalmente en dar elementos a través de sentencia simples y se propone a trabajo a futuro, sentencias escritas con más complejidad, no realizan pruebas.	2018
Automatic Glossary Term Extraction from Large-Scale Requirements Specifications	Texto a Glosario de términos	Procesamiento de Lenguaje Natural	Se pudieron conseguir una gran cantidad de términos en los conjuntos de requerimientos con un 74 % de recall y una precisión del 73 %	2018

Nombre	Derivación	Tecnologías	Resultados	Año
Generate use case from the requirements written in a natural language using machine learning	Texto a Casos de usos	Procesamiento de Lenguaje Natural y Aprendizaje Automático	Se analizan las limitaciones dadas por las herramientas para el lenguaje inglés pero consiguiendo una precisión de 72 % y un recall del 69 %	2019

Tabla 3.1: Resumen de las investigaciones realizadas

3.4 Conclusión

Analizando cada uno de las secciones vistas, se puede realizar un camino marcado de cuales fueron los avances tecnológicos y análisis incorporados a lo largo de los años para conseguir de manera automática o semiautomática los modelos planteados a partir de lenguaje natural o requerimientos escritos en lenguaje natural. Se puede decir, que durante los primeros años donde se empezó a implementar estas extracciones los módulos más simplistas se basaban en conseguir la información a través de módulos enteramente de PLN y con reglas heurísticas simples para filtrar la información. Luego, se fueron incorporando análisis más complejos con respecto a los filtros realizados sobre la el módulo de PLN. Donde surgen reglas heurísticas muchos más complejas y abarcativas, entra en juego los análisis estadísticos para recuperar información relevante. Pero se ve claramente, que en los últimos años la incorporación de Machine Learning fue necesaria y positiva tanto para filtrar elementos de los modelos como para directamente utilizar estos algoritmos para conseguir los resultados finales. Por lo tanto, se puede pensar que un flujo de implementación óptimo, puede tender a utilizar PLN como módulo de preprocesamiento, reglas heurísticas para buscar ciertos patrones sintácticos, modelos estadísticos para obtener información relevante y algoritmos de aprendizaje automático para obtener los resultados finales o para filtrar aquellos que se consideran correctos.

Capítulo 4

Herramienta propuesta

4.1 Introducción

Este capítulo describe detalladamente la planificación y desarrollo de la herramienta propuesta en esta tesina. En las primeras secciones se describen las problemáticas existentes en el procesamiento de lenguaje natural, el análisis de una estructura convencional de LEL, las tecnologías utilizadas, el uso de las mismas y la implementación final de la herramienta.

4.2 Análisis de la estructura LEL

Esta sección describe cuales son fueron los factores más importantes que se tuvieron en cuenta para desarrollar la herramienta. Para ello se realizó un análisis entre las relaciones que existen entre un documento LEL y el idioma español. Un documento LEL tiene una estructura general bien definida, donde dicha estructura se utiliza para definir el contexto de una aplicación. Los elementos sintácticos utilizados en un LEL se los denominan «Símbolos». Estos símbolos nos sirven para definir varios tipos de elementos en un dominio. Los símbolos se definen cuatro categorías básicas: los sujetos, objetos, verbos y estados. Cada uno de estos símbolos se definen a través de dos atributos: impactos y nociones [12]. Los impactos definen la connotación de ese símbolo y las nociones de las características intrínsecas dentro de un dominio en particular.

La finalidad de esta herramienta es encontrar lo distintos símbolos del texto y los impactos relacionados entre cada uno de ellos. Las nociones no fueron abordadas por la fuerte dependencia que tiene con un dominio en particular y esta herramienta fue hecha para cualquier tipo de dominio. Para encontrar los distintos símbolos resultó

importante buscar las relaciones entre la sintaxis del idioma español y los elementos de un LEL. El primer elemento de esta categoría, que son los *sujetos*, el cual corresponde con el mismo nombre y semántica que se encuentra en el análisis sintáctico de oraciones del idioma español, son aquellos elementos que realizan determinadas o que se describen, dichas acciones realizadas en un LEL se los denominan *verbos*. Los verbos son unos de los desafíos más importantes de este análisis ya que deben ser analizadas con una perspectiva de la diátesis del idioma español.

La diátesis es un rasgo gramatical que describe el numero de argumentos verbales que necesita un verbo para que una sentencia tenga sentido. Por ejemplo: El cliente agrega productos al carro de compra, es un buen ejemplo de como son necesarios los argumentos «productos» y «carro de compra» para que la sentencia tenga un sentido. También existen aquellas oraciones donde no hay necesariamente un objeto directo al cual la acción esta afectando, como por ejemplo: El cliente esta comprando. En esos casos se descartó de utilidad o de información este tipo de sentencias, ya que no realizan un aporte a la estructura de un LEL. El análisis sintáctico se realizo sobre toda sentencia donde un sujeto realiza una acción sobre un objeto directamente e indirectamente. Los objetos directamente e indirectamente accionados por los sujetos son los símbolos denominados *objetos*. En algunos casos estas acciones provocan ciertos cambios sobre el objeto, a estos cambios se los llama *estados*. Los estados definen un antes y un después en un objeto luego de que un sujeto determinado realice un acción sobre ellos.

Si bien el español se caracteriza por tener poca organización entre sus elementos sintéticos, esta tesina se enfoca en la utilización de sentencias con cierta estructura. Sentencias que tengan coherencia y cohesión entre cada uno de los elementos. Es importante remarcar que una vez encontrado los símbolos, por como esta compuesta la estructura LEL, es posible encontrar los distintos impactos de cada sujeto. La herramienta despliega cada uno de los impactos, ya sea de la manera *sujeto + verbo + objeto*, o con un estado resultante luego de una acción: *sujeto + verbo + objeto + estado* [12].

4.3 Arquitectura de la plataforma

La siguiente sección presenta la arquitectura de la plataforma, la cual se compone de herramientas multiplataforma de software libre. Además se describe como se realiza la interacción entre cada uno de los componentes.

4.3.1. Vista general de la plataforma

A continuación se representa cada uno de las herramientas utilizadas y como interactúan entre si para conformar la plataforma final.

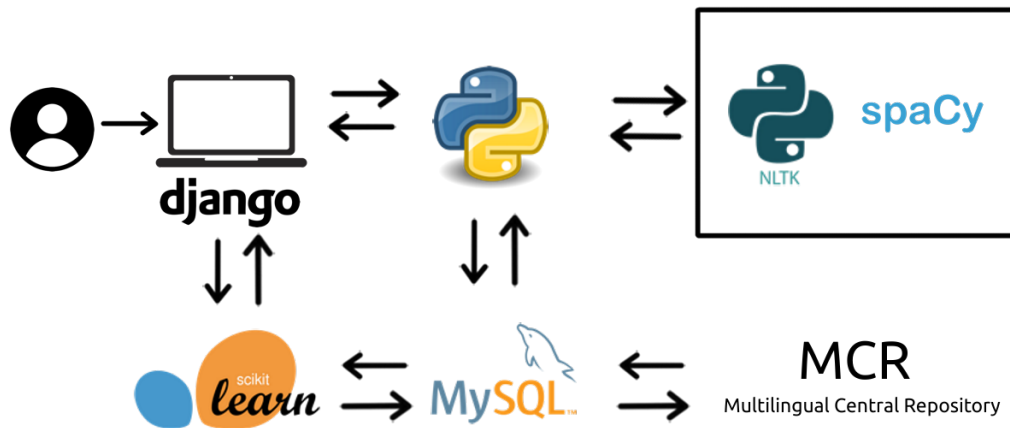


Figura 4.1: Arquitectura del desarrollo

4.3.2. Python como lenguaje de programación

Como lenguaje de programación se eligió Python, por tener la característica de ser un lenguaje de alto nivel con una curva de aprendizaje pequeña, hoy en día es un lenguaje que esta en auge y es el que mayor crecimiento ha tenido en los últimos años, adicionalmente se acopla con facilidad con herramientas de análisis de texto y de aprendizaje automático. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional[3]. La elección de este lenguaje se debió al encontrar una gran cantidad de herramientas de que realizan procesamiento de lenguaje natural que correspondían con la funcionalidad que se necesitaba al comienzo de esta tesina[3][45].

4.3.3. Plataforma web

Dentro de las plataformas web soportadas por Python, Django es sin duda la más utilizada para realizar desarrollos simple y rápidos, para garantizar al usuario una mejor experiencia con la herramienta, se pensó en un plataforma web, por la características de multiplataforma que poseen dichos tipos de sistema. Para implementarla se utilizó Django. Es un framework de aplicaciones web gratuito y de código abierto escrito en Python. Un framework web es un conjunto de componentes que te ayudan

a desarrollar sitios web más fácil y rápidamente[46]. Este framework es un desarrollo bastante maduro y que predomina con excelencia en el ambiente de programación web sobre Python. Adicionalmente la utilización de un Framework que cumple con un patrón Modelo Vista Controlador, el cual fue un factor importante a la hora de elegirlo debido que durante los años de carrera, el MVC es un patrón visto, estudiado y trabajado, disminúyenos aun más la curva de aprendizaje.

4.3.4. Procesamiento de lenguaje natural: Spacy y NLTK

Utilizar Python se debió a la gran cantidad de herramientas que trabajan con procesamiento de lenguaje natural, Spacy y NLTK son herramientas open-source que vienen teniendo un crecimiento agigantado en los últimos años, con la automatización de procesamiento de datos de tipo texto. Las herramientas Spacy y NLTK son utilizadas a través de todo el desarrollo desde la primera interacción con la entrada del usuario hasta en la salida final, por lo tanto la utilización de las mismas era fundamental para poder realizar esta tesina. Las herramientas Spacy se utilizó por la sencillez a la hora de analizar sentencias de tipo texto y por el análisis sintáctico que realiza sobre las mismas. Por otro lado, NLTK (Natural Language Tool Kit) provee funciones de tokenización de oraciones y palabras con una resolución muy simple de utilizar [3] [46]. Se puede concluir que Spacy se utilizó para procesamientos un poco más complejos ligados a la sintaxis del idioma español y NLTK para realizar procesamientos más referidos a la algoritmia de la herramienta, como recorrer oraciones o palabras.

4.3.5. Aprendizaje automático: Sklearn

Para cualquier herramienta de automatización es importante recolectar la mayor cantidad de retroalimentación con usuarios posible, debido a que cualquier de proceso automatizado contiene márgenes de errores. Este proyecto contaba con la pretensión de poder tener un feedback por parte del usuario y poder mejorar la salida medida que se utiliza el sistema. La maquina de aprendizaje automático utilizada en este proyecto utiliza el modelo matemático de Bayes el cual se utiliza para poder reconocer las salidas por la aplicación que fueron útiles para los usuarios, aprender de ello y mejorar de manera continua. Este modelo se implementó simulando interacciones por el usuario por expertos en LEL, de esta manera ya se tiene por default un aprendizaje automático acerca de las salidas LEL. Como el modelo Bayes se debe volver a entrenar si existen datos nuevos, el aprendizaje debe ser planeado en caso de tener esta plataforma puesta en producción. La librería utilizada para realizar es sklearn, la cual ya cuenta con varios modelos matemáticos implementados en el cual esta Bayes,

de esta manera cuando el usuario interactúa con la plataforma para dar su opinión por la salida LEL, estas elecciones son guardadas a través de una base de datos SQL.

4.3.6. Base de datos MySQL

La utilización de un DBMS para almacenar ciertos resultados y para información que se utilizará en este proyecto es fundamental. La elección de una base de datos relacional de MySQL, se dio por la simplicidad de la misma condicionado por la baja complejidad de los datos ingresados. Adicionalmente Django contiene una integración transversal con MySQL, donde permite configurar ORM (Mapeo Objeto-Relacional) de una manera sencilla, esto no solo garantiza una facilidad al trabajar con objetos en alto nivel si no que garantiza seguridad a nivel base de datos. El aprendizaje a través de la carrera de base de datos relacionales similares a MySQL fue una influencia en esta elección. La decisión de su utilización se vio definitiva cuando se halló una base de datos en este DBMS que contenía un repositorio completo del lenguaje español.

4.3.7. Repositorio central de multi-lenguaje

Para mostrar algunos tipos de nociones básicas de verbos y acciones que son transversales a varios dominios de aplicación, se necesitó una integración con un glosarios de palabras, que nos permita encontrar definiciones, relaciones y semántica. Por lo tanto se investigó distintos tipos de glosarios, donde se hallaron MultiWordNet, WordNet y The Multilingual Central Repository (MCR) [47][48][49]. Todos estos glosarios cuentan con una gran base de datos para consultar la semántica, los hipónimos, los hiperónimos de cualquier palabra del idioma español, per la mayoría de las definiciones de palabras estan en el idioma inglés, esto fue un limitante crucial ya que la herramienta está desarrollada para utilizarse en el idioma español, ante esta eventualidad se investigaron varias alternativas, teniendo en cuenta las siguientes problemáticas. WordNet y MultiWordnet son glosarios en el idioma inglés con búsqueda de palabras en español bastante completos pero solo son accesibles desde sitios web por lo tanto la incorporación a gran escala de definiciones y relaciones a la plataforma se vuelve imposible de escalar, descartando todos aquellos que sean repositorios de consulta web. En el caso del repositorio MCR, cuenta con un sitio web para consultar pero además cuenta con una base de datos SQL libre para descargar. Esta característica genera notables ventajas, las consultas de definiciones se pueden realizar de manera de local y rápida. Además permite poder modificar la base a las necesidades de este proyecto. Aunque la base de datos se encuentra en su mayoría en inglés, se creó una alternativa menos convencional, que sería traducir en español todas esa definiciones. Realizar este procesamiento de manera manual llevaría muchísimo tiempo y ante un eventual crecimiento de este repositorio no sería escalable. Por lo tanto se realizó una automa-

tización utilizando una librería Google Translate de Python donde se comunica con la API de Google Translate para obtener las distintas traducciones, en este procesamiento se tradujo 146261 definiciones que contiene en inglés este repositorio[50]. Se creó un script de Python, con el cual se conecta a la base de datos en inglés, recupera la definición de una palabra y envía el un requerimiento a la API de Google para buscar su traducción, para que luego se inserte en la base de datos local como una nueva definición. De esta manera luego de 24 horas de ejecución se tuvo una base de datos completa de manera local de definiciones de palabras en el idioma español. Esto permitió tener una mejora muy importante en el tiempo de respuesta, generó que el desarrollo sea mucho más escalable y consiguiendo el objetivo de contar con nociones importantes de ciertas palabras.

4.4 Esquema de extracción de información



Figura 4.2: Esquema de extracción de información

El esquema de extracción de información está separado en varias etapas, las cuales se representan en la figura 4.2. La etapa inicial consiste en disponer al usuario una entrada de datos de tipo texto para procesar y conseguir símbolos de un LEL a través de esta información. Debido a que el lenguaje en español presenta muchísimas

variables de escritura, no todas las sentencias están escritas en pos activa en referencia a un sujeto en particular, se dispuso de varias reglas heurísticas para controlar y reestructurar las sentencias que son ingresadas por el usuario, esto resultó de gran utilidad debido a que permite una flexibilización para el usuario a la hora de ingresar información. Luego la herramienta a través de distintas técnicas PLN estructura la información de manera tal de poder encontrar elementos sintácticos en cada sentencia del texto ingresado por el usuario. Para resaltar aquellos elementos que son de importancia, se analizó filtros estadísticos que encuentre una relación entre la frecuencia en que ocurren las palabras y la importancia de las mismas en el texto. Esto sirve para encontrar dos símbolos importantes, por los cuales se pueden inferir los demás. Estos son los sujetos y los objetos, tal como lo indica su nombre estos son similares o análogos a los elementos sintácticos encontrados en el lenguaje español. Una vez que la herramienta encuentra estos elementos, se los dispone al usuario para que elija, agregue o quite elementos según la visión de analista que tiene y el LEL que espera como resultado.

Una vez definidos los símbolos principales la herramienta procederá a buscar otros símbolos relacionados con los mismos, tales como las acciones y los estados, en este proceso se realiza un procesamiento a través de técnicas de lenguaje natural y adicionalmente interviene el módulo de machine learning para filtrar símbolos que, según conocimiento previo, no resultarían adecuados para un documento LEL. Por último, el usuario tendrá la opción de considerar cuales impactos o conjunto de símbolos le parecen adecuados, estas decisiones serán guardadas para luego mejorar el algoritmo de aprendizaje automático.

4.5 Diseño de la herramienta

Para realizar las interacciones con el usuario se propone un sitio web con un patrón de Modelo Vista Controlador, donde cada una de estas partes interactúa con múltiples herramientas. Empezando con el modelo, utilizando la configuración básica de Django y sus respectivos ORM, para comunicarse con el servidor MySQL para obtener información necesaria para la herramienta. Las vistas por otro lado, son los componentes que crean el nexo entre el usuario y la lógica interna de la herramienta. Estas vistas crean una interfaz más amigable para que el usuario pueda interactuar, a través de ellas el usuario es capaz de poder ingresar entradas de datos de una manera simple y fácil. Y por último están los controladores que realizan todo el procesamiento de la herramienta, aquí se encuentra el mayor de uso de componentes, como las librerías de Procesamiento de Lenguaje Natural, Modelos de aprendizaje automático y la comunicación con los modelos, es el núcleo de la herramienta.

El diseño consta de un entorno simplista y con pocas interacciones para el usuario, más allá de las etapas de procesamiento ejemplificadas en la figura 4.2 la herramienta consta cuatro vistas principales donde divide las distintas etapas del procesamiento que realiza la plataforma, donde cada una de ellas tiene componentes visuales para poder avanzar hacia la siguiente etapa o volver en caso de haber cometido un error o querer realizar una modificación. Las cuatro etapas se pueden describir en: Ingreso de información, Procesamiento y recolección de símbolos, Búsqueda de impactos y por último la exportación de la misma.

En la primera de etapa del ingreso de información, se modela una vista simple con un entrada de texto, adicionalmente cuenta con ciertos criterios y recomendaciones para que el usuario tenga en cuenta para el ingreso de dicha información. En este momento solo interactúa la parte de visual de Django pero este es el primer disparador para comenzar el procesamiento y estructuración de texto.

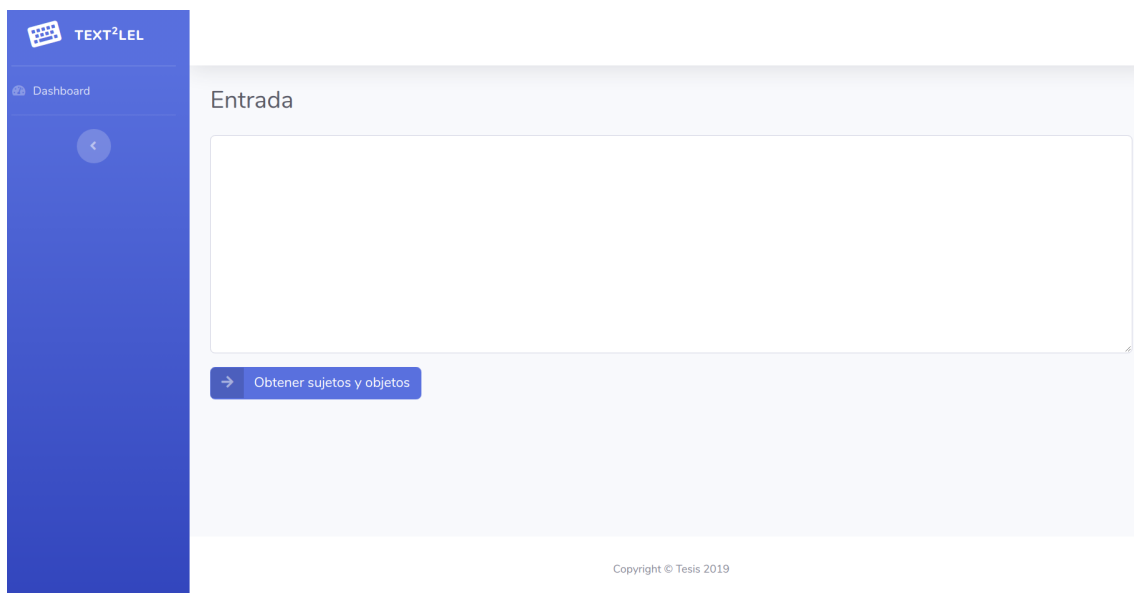


Figura 4.3: Primera etapa - Entrada de información del sistema

En la segunda parte, se utilizan el núcleo de procesamiento de lenguaje natural para procesar el texto y obtener los símbolos correspondientes a esta etapa. Luego el sistema muestra los datos encontrados en dicha etapa, permitiendo al usuario poder realizar alta, baja y modificación de los símbolos encontrados, en esta etapa solamente se tienen en cuenta los sujetos y objetos. Esta etapa es fundamental para que el usuario tenga en cuenta cual fue la información relevante encontrada en la información

ingresada. Adicionalmente este puede modificar salida de la herramienta para que la siguiente etapa se amolde más a sus necesidades.

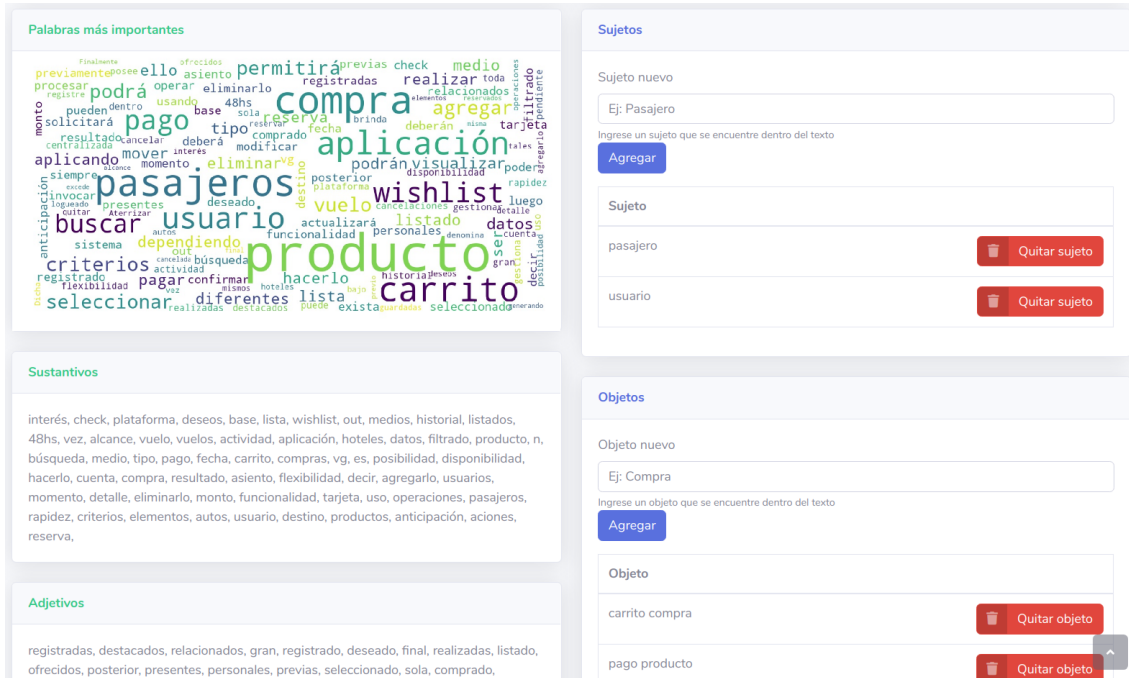


Figura 4.4: Segunda etapa - Vista de objetos y sujetos encontrados

La tercer etapa, con más procesamiento e importancia de la herramienta consiste en utilizar los símbolos definidos en la etapa anterior, utilizando nuevamente el núcleo de NLP para obtener los símbolos faltantes, verbos y estados. Para cada uno de estos símbolos se utiliza el Repositorio Central de Multi-Lenguaje (MCR) para conseguir las nociones de los símbolos encontrados. Para finalizar esta etapa de procesamiento se utiliza el modelo de aprendizaje automático para determinar la fiabilidad de ciertos conjuntos de símbolos (impactos), la construcción de dicho modelo se explicará en los siguientes párrafos.

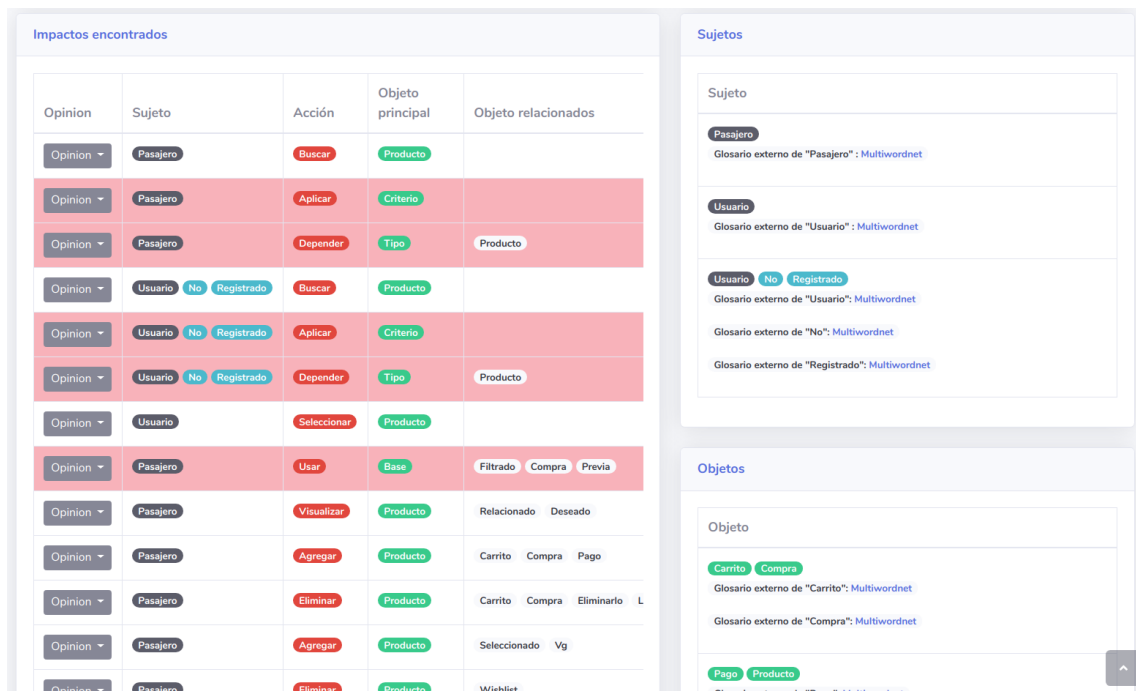


Figura 4.5: Tercera etapa - Impactos encontrados

Como resultado final y cuarta etapa, la aplicación tiene los distintos símbolos (sujetos, objetos, verbos, estados) y con sus respectivas nociones e impactos resultantes de la relaciones entre los distintos símbolos. Para permitir al usuario utilizar los resultados provistos por la plataforma, el sistema tiene la opción de exportar los resultados en un formato amigable para que sea fácil de trasladar a un documento.

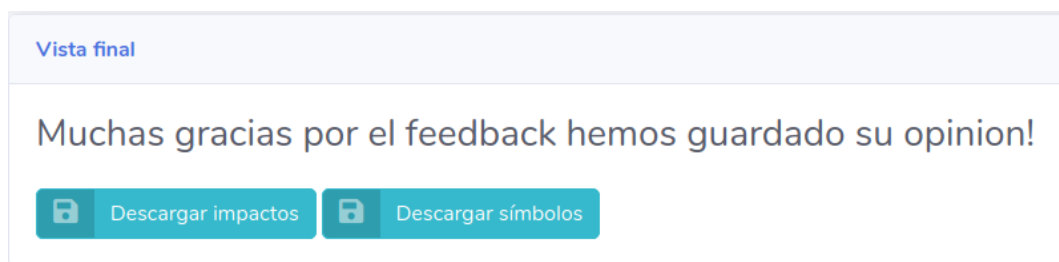


Figura 4.6: Cuarta etapa - Exportación de resultados

Teniendo en cuenta que un sistema automatizado como esta herramienta tiene falsos positivos, se mitiga varios de estos resultados teniendo feedback adecuado con el usuario permitiéndole en todo momento poder cambiar resultados o agregar nueva información, esta retroalimentación se utilizara para generar mejores resultados

tanto para el usuario como para el sistema, gran parte de las acciones determinantes en el sistema son guardadas en la base de datos para poder utilizar algoritmos de aprendizaje automático utilizando la librería Sklearn y poder aprender de estas agregaciones.

Para generar un LEL con más información, una vez encontrado todos los símbolos pertenecientes a los distintos impactos por parte del usuario, el sistema dispone de la base de datos del Repositorio central de multi-lenguaje con los significados de gran parte de las palabras del idioma español, de la cual se extraerá las distintas nociones de cada símbolo, de esta manera se busca asemejarse con los resultados de un LEL convencional, sin embargo esto esta sujeto totalmente a revisiones ya que el significado y el peso de las palabras esta intrínsecamente relacionado al dominio y al contexto en cual está siendo empleado. Tanto los resultados del feedback con el usuario como el repositorio central multi-lenguaje se almacenan dentro de base de datos relacionales permitiendo una rápida respuesta ante consultas.

4.6 Implementación y flujo de la herramienta

En esta sección se describe con detalle cada componente implementado en la herramienta, interrelacionando las tecnologías utilizadas con el flujo de procesamiento que el usuario realiza.

4.6.1. Datos de entrada

El usuario final dispone a través del sitio web implementado en Django una entrada donde puede ingresar datos de tipo texto que permite que esa información llegue al núcleo de la aplicación para su procesamiento. Los datos de entrada para este desarrollo toman una importancia más que significativa debido a que la calidad y cantidad que tengan responderá directamente con la calidad de la salida del desarrollo. Como este proyecto surge a partir del interrogante de poder hacer un LEL de una manera más eficiente y rápida se cuenta con varios documentos de LEL's creados por expertos en la materia y de alumnos de postgrado de la carrera del Máster de Ingeniería de software. A su vez se utilizaron especificaciones de requerimientos del mismo dominios de los LEL's para suministrar información un poco más definida acerca los activos que aparecen en cada dominio específico.

Es importante destacar que la utilización del procesamiento de lenguaje natural es limitado con respecto a identificar componentes de manera inteligente en un texto poco estructurado, por lo tanto la utilización de oraciones semi estructurada son

necesarias para poder identificar activos importantes tales como los sujetos y los componentes a los cuales ellos tan asociados. Por lo tanto la entrada de tipo texto deberá cumplir con los siguientes requerimientos.

- Se debe ingresar un documento que especifique requerimientos funcionales de un sistema.
- El texto debe contar con estructuras sintácticas bien definidas con respecto a sujeto y predicado.

Teniendo en cuenta las consideraciones mencionadas, el usuario luego que ingresa una entrada de tipo texto para analizar, la herramienta pasa al primer procesamiento denominado «Reestructuración de sentencias» a través del módulo de reglas heurísticas que será explicado en la siguiente sección.

4.6.2. Módulo de reglas heurísticas

Las reglas heurísticas en procesamiento de lenguaje natural permite como dice su nombre reglamentar e identificar que tipos de sentencias son válidas a nivel sintáctico o también a nivel semántico. Teniendo en cuenta esto, se analizó la posibilidad de identificar aquellas sentencias que no tienen una estructura simple, como por ejemplo una oración que este en voz pasiva y que el núcleo de la sentencia no sea el sujeto que termina utilizando una aplicación. Para esto se analizó un módulo de spacy [51], llamado Rule-Based Matching, el cual permite obtener matcheos de sentencias de acuerdo a reglas que pueden estar orientadas a palabras literales, a tags sintácticos e incluso a dependencias sintácticas. Estas reglas sintácticas permiten identificar palabra a palabra si cumplen con lo especificado en la regla. Por ejemplo la siguiente regla: «`{'POS': 'DET'}, {'POS': 'NOUN'}`» tiene como matcheo y devuelve la posición de aquellas palabras donde se tenga un determinante como «el», «la», «los», etc. y un sustantivo. Y de esta manera también se pueden realizar reglas más explicitas como buscar cuando un usuario realiza una acción, ejemplo: «`{'POS': 'DET'}, {'LEMMA': 'usuario'}` , `{'POS': 'VERB'}`». La palabra clave LEMMA permite buscar la raíz de una palabra, entonces para usuarios, usuarias, usuario, el LEMMA de cada uno de ellas será «usuario».

Por lo tanto, con esta herramienta es posible reestructurar las sentencias ingresadas por un usuario a un formato más y sencillo de procesos. Para ello se planteo 5 modelos con múltiples reglas heurísticas, donde se planteó una estructura jerarquizada como se ve en la figura 4.7 donde la clase padre de todos módulos unifica criterios a la hora de identificar sustantivos, verbos, conjunciones, etc y provee métodos que

son comunes a todos, esto permite un desarrollo escalable ya que a la hora de agregar un nuevo módulo o cambiar algún criterio de identificación de algún elemento sintáctico ya que estos cambios se realizarán de manera global en todos los módulos. De esta super clase se desprenden los módulos de: sentencias orientadas a sistemas, sentencias en voz pasiva, sentencias con conjunciones de sustantivos y verbos y por último el módulo que identifica identificación de sub-símbolos o jerarquía de símbolos. A continuación se explicará cada uno de estos módulos y en los siguientes párrafos la utilización en la herramienta.

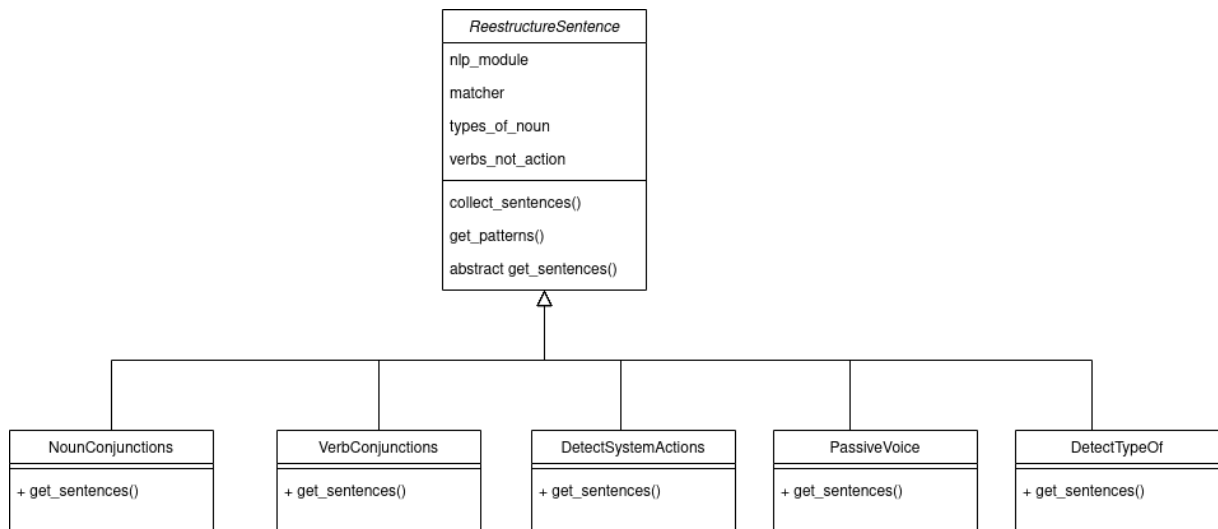


Figura 4.7: Jerarquía de clases de reestructuración

Sentencias orientadas a un sistema o aplicación

En este módulo se intenta identificar aquellas sentencias donde se define una funcionalidad de un sistema. Por ejemplo: «El sistema permitirá a los usuarios no registrados buscar productos en la página». Entonces la idea principal es que identifique si existe un usuario relacionado con la funcionalidad descrita y que reestructure la sentencia con foco al usuario. Para realizar estas reglas se analizaron las siguientes cuestiones sintácticas y gramaticales:

- Comienzan con un determinante. Ejemplo «*El* sistema»
- El que permite la acción es un sustantivo.
- Puede tener verbos auxiliares y contiene acciones que avalen una acción como: «permitir», «garantizar», «brindar».

- Puede afectar a un sujeto o varios sujetos.
- La acción puede ser representada a través de un verbo. Ejemplo: «El sistema permite a los usuarios comprar ...»
- La acción puede ser representada a través de un sustantivo. Ejemplo: «El sistema permite a los usuarios la compra de ..»

Sentencia: «*El sistema permite a los usuarios registrados comprar productos en la página principal*»

Sentencia post-reestructuración: «*Usuarios registrados compran productos en la página principal*»

Sentencias en voz pasiva

El módulo de sentencias en voz pasiva es un módulo importante para identificar correctamente cual es el sujeto de una acción. Por ejemplo: «Los productos son comprados por un usuario», en un procesamiento simple de esta sentencia, el sujeto indicaría que es el producto, por lo tanto lo que realiza este conjunto de reglas es inferir cual es el objeto asociado, cual es la acción y quien es el verdadero sujeto, esto basándose en la estructura básica de voz pasiva. Por lo tanto se tiene en cuenta lo siguiente:

- Existe un sujeto que es un sustantivo
- Existe el verbo lemma ser. Por ejemplo: «Los productos pueden *ser* comprados por usuarios» o «Los productos *son* comprados ..»
- Puede tener verbos auxiliares. Ejemplo: «Puede».
- Existe una post-posición de la palabra «por».
- Y por último existe un sustantivo luego de la palabra «por»

Sentencia: «*Los productos son comprados por usuarios*»

Sentencia post-reestructuración «*Los usuarios compran productos*»

Conjunciones de sustantivos

El módulo de conjunciones de sustantivo como de verbos surgen de la dificultad de representar múltiples sujetos, objetos y verbos que pueden estar implicados en una oración, tanto para procesar la información sino para otros módulos utilizados en esta jerarquía. Este módulo en particular busca atacar aquellas sentencias en voz activa que tengan múltiples objetos y múltiples sujetos. Por ejemplo: «Los usuarios registrados, no registrado y administradores pueden reservar pasajes de avión, hoteles, y autos» . En un análisis normal esto tal vez contaría como una sola sentencia, pero en realidad estamos hablando de por lo menos 9 sentencias individuales, donde cada sujeto se corresponde con una acción y con un objeto en particular. Además de dar visibilidad a nuevos elementos esto contribuye a identificar de manera más precisa cuales objetos y sujetos pueden llegar a ser importantes. Para lograr esta reestructuración se tuvieron en cuenta los siguientes criterios:

- La oración esta en voz activa
- Puede haber conjunciones simples o compuestas. Es decir, a través de comas o conectores como «y,e,o,u»
- Los sustantivos pueden ser simples o compuestos. Ejemplo: «Los pasajeros frecuentes y los administradores pueden ...»
- Los sustantivos pueden compartir un núcleo. Ejemplo: «Los usuarios registrados y no registrados pueden ...». Por lo tanto la parte «no registrados» hereda el núcleo «usuario» como sujeto.
- Tiene que haber un verbo que defina el sujeto y predicado de la sentencia.
- Puede haber conjunciones tanto en el sujeto como en el predicado.

Sentencia: «*Los usuarios registrados y no registrados pueden reservar hoteles, autos y pasajes de avión*»

Sentencias post-reestructuración: [«*Los usuarios registrados pueden reservar hoteles*» «*Los usuarios registrados pueden reservar autos*» «*Los usuarios registrados pueden reservar pasajes de avión*» «*Los usuarios no registrados pueden reservar hoteles*» «*Los usuarios no registrados pueden reservar autos*» «*Los usuarios no registrados pueden reservar pasajes de avión*»]

Conjunciones de verbos

El módulo de conjunciones de verbos al igual que el de sustantivos busca dar visibilidad aquellas conjunciones de verbos que con un simple análisis de PLN es difícil encontrar. Por lo tanto, este conjunto de reglas busca aquellas conjunciones simples o compuestas de verbos realizados en voz pasiva o voz activa. Para realizarlo se tuvieron en cuenta los siguientes criterios:

- La oración puede estar en voz activa o pasiva.
- Puede haber conjunciones simples o compuestas. Es decir, a través de comas o conectores como «y,e,o,u»
- Una vez que identifica verbos replica cada uno de ellos con el sujeto y el predicado.
- Se contempla errores de POS Tagging para verbos escritos en voz pasiva, como «son comprados», puede ser taggeado como adjetivo.

Sentencia: «*Los usuarios registrados pueden comprar, reservar o alquilar hoteles en marzo*»

Sentencias post-reestructuración: [«*Los usuarios registrados pueden comprar hoteles en marzo*» «*Los usuarios registrados pueden reservar hoteles en marzo*» «*Los usuarios registrados pueden alquilar hoteles en marzo*»]

Identificación de sub-símbolos

La identificación de sub símbolos identifica de aquellos símbolos que son parte de otro símbolo en si mismo. Análogamente como las clases y superclases. Por ejemplo, si en un texto, clasifica a un símbolo en distintas variantes esto quiere decir que estas variantes también serán símbolos del documento LEL. Para poder obtener este tipo de clasificación se tomaron los siguientes criterios.

- El símbolo padre aparece primero en la sentencia
- La clasificación esta dada por verbos especiales como: «clasifican», «dividen», «separan», «ofrecen»
- También puede ser identificado a través de la expresión: «pueden ser».
- Se tienen en cuenta los posibles dos puntos (:).

- Se tiene en cuenta que previamente paso por otros módulos que simplifican el procesamiento.

Sentencia: «Los usuarios pueden ser divididos en usuarios registrados»

Resultado de la herramienta «Usuarios registrados es un tipo de usuario»

4.6.3. Uso de módulos de reglas heurísticas y reestructuración de sentencias

Luego de que el usuario ingrese la información a procesar, el primer paso a realizar es reestructurar la sentencias a través de los módulos explicados en el paso anterior, en este paso no se normaliza todavía la entrada, ya que la falta de elementos sintácticos como puntuaciones, o stopwords puede modificar los TAGS asignados por las herramientas, por lo tanto para no perder semántica en las oraciones y que esto influya sintácticamente, este paso se realiza directamente sobre los datos que ingrese el usuario.

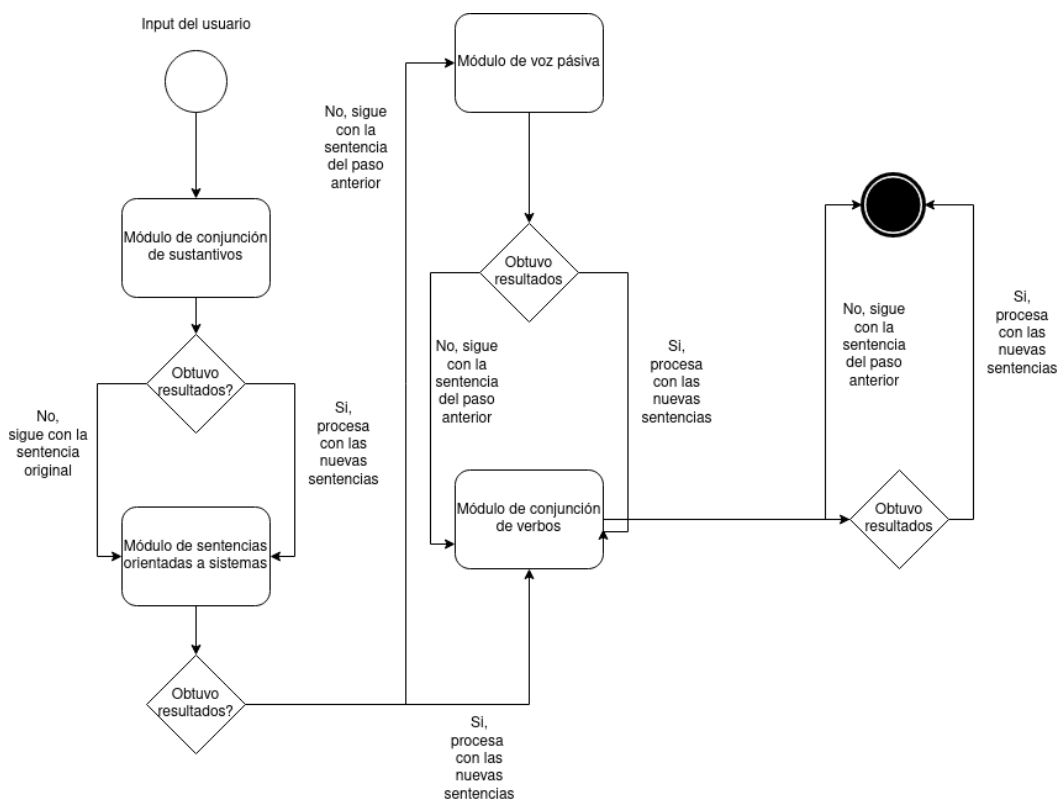


Figura 4.8: Flujo de módulos de reestructuración

Tal como se ve la figura 4.8 la herramienta propone dos flujos principales, el primero es procesar sentencias en caso de que sean sentencias orientadas a una funcionalidad de un sistema o una aplicación. Y el otro flujo es el caso de que exista una sentencia que este en voz pasiva con respecto a la funcionalidad de un usuario, luego ambas pasan por el mismo procesamiento de verificación de conjunciones de sustantivos y verbos. A continuación se deja ejemplificado el caso del primer flujo en caso de que pase por todos los módulos:

Ejemplo flujo 1

Sentencia: *«El sistema debe permitir a los usuarios registrados, no registrados alquilar y reservar hoteles y autos.»*

Resultado del flujo luego del módulo de sustantivos: [[*«El sistema debe permitir a los usuarios registrados alquilar y reservar hoteles.»*],[*«El sistema debe permitir a los usuarios registrados alquilar y reservar autos.»*],[*«El sistema debe permitir a los usuarios no registrados alquilar y reservar hoteles.»*],[*«El sistema debe permitir a los usuarios no registrados alquilar y reservar autos.»*]]

Resultado del flujo luego del módulo de detección de sistemas:[[*«Los usuarios registrados alquilan y reservan hoteles.»*],[*«Los usuarios registrados alquilan y reservan autos.»*],[*«Los usuarios no registrados alquilan y reservan hoteles.»*],[*«Los usuarios no registrados alquilan y reservan autos.»*]]

Resultado final luego del módulo de detección de conjunciones de verbos, sentencias encontradas:

- Los usuarios registrados alquilan hoteles.
- Los usuarios registrados reservan hoteles.
- Los usuarios no registrados alquilan hoteles.
- Los usuarios no registrados reservan hoteles.
- Los usuarios registrados alquilan autos.
- Los usuarios registrados reservan autos.
- Los usuarios no registrados alquilan autos.
- Los usuarios no registrados reservan autos.

4.6.4. Normalización de la entrada

Luego de la reestructuración de las sentencias a un formato prácticamente único a nivel sintáctico y estructural, la herramienta tiene como segunda etapa la normalización convencional que se tiene en la mayoría de los desarrollos de procesamiento de lenguaje natural. Este proceso de normalización se realiza a nivel token o a nivel palabra de una sentencia, por lo general se realiza este tipo de proceso para unificar el formato de la información y adicionalmente evita en gran medida los falsos positivos. En esta sección se explicará brevemente las distintas metodologías aplicadas en el primer ingreso de la información por parte del usuario.

El primero proceso de normalización que se tuvo en cuenta fue eliminar las palabras denominadas «stopwords». Estas palabras son aquellas que no agregan una importancia o no cambian la semántica de una sentencia, estos elementos pueden ser pronombre, preposiciones, artículos, etc. Este proceso se consigue través de distintas técnicas de procesamiento lenguaje natural, y es recomendable para evitar casos de falsos positivos a la hora de hallar información relevante en un texto ya que pueden generar ruido en búsquedas estadísticas de información. Para resolver este tipo de problemas por lo general se recopilan ontologías previamente verificadas por organizaciones o empresas que proveen las librerías. Estos glosarios se utilizaron para realizar un filtro de palabras, y de esta manera eliminar de una sentencia toda aquella palabra que este en esta ontología. Como segunda acción de normalización, se transformo la entrada de tipo texto a un formato unificado, en este caso, se paso a minúscula todas las palabras de la entrada. Esto permite que no haya diferencias entre palabras que solamente varían por la manera en las que están escritas.

Para ejemplificar, se encuentra el siguiente pseudo-código donde se utiliza la librería nltk que nos provee un diccionario de stopwords que nos permite eliminar cada una de esas palabras de una sentencia determinada.

```
1
2 def clean_stopwords(sentence):
3     // Tokeniza la sentencia
4     words = word_tokenize(sentence)
5     // Obtengo conjunto de stopwords del idioma español
6     spanish_stops = set(stopwords.words('spanish'))
7     // Se elimina toda aquella palabra que se encuentre
8     // en la ontologia y este en la sentencia.
9     return " ".join([word for word in words \
10                     if word not in spanish_stops])
```

4.6.5. Estructuración de texto: Obtener Sujetos y Objetos Principales

En este párrafo se explica el proceso que realiza la herramienta para conseguir los sujetos y objetos determinantes de un texto. Para realizar cualquier tipo de procesamiento de texto no estructurado dentro de un programa es necesario poder estructurar lógicamente la información, para ello una vez que el texto ya este normalizado, el objetivo es estructurar el texto de una manera que sea un objeto iterable, es decir poder alcanzar una división lógica de elementos independientes dentro de un texto, en este caso se hará esa división a través de las sentencias, por lo tanto se divide la información por cuantas sentencias u oraciones tenga la entrada de datos, donde cada una de estas unidades tiene su semántica y cuenta con sus objetos independientes. Para conseguir dicha estructura se utiliza un método de la librería NLTK [3], donde este método identifica todos los signos de puntuación convencionales o saltos de líneas para definir el fin de una sentencia. Aunque la independencia dentro del idioma español rara vez sucede, es importante aclarar que el alcance de esta tesina esta ligada a sentencias independientes entre ellas, de esta manera se puede procesar de manera completa y obtener la máxima cantidad de información de cada una de ellas. Por lo tanto la herramienta no será capaz de identificar relaciones semánticas entre las sentencias tales como anáforas.

Una vez obtenido esta división, el objetivo es poder estructurar aún más la información para identificar los roles que cumplen los elementos sintácticos en una sentencia determinada. En este caso fue de utilidad dividir cada sentencia realizando el análisis sintáctico básico de la lengua española, es decir, dividir en sujeto y predicado. Esto se utilizó para tener una referencia de cuales son los sustantivos que se utilizan como sujetos y como objetos. Para lograr esto, lo único que es necesario teniendo en cuenta una sentencia con sujeto y predicado explicito, es encontrar el verbo que divide estas dos estructuras. Para conseguir estos elementos se utiliza la técnica de POS Tagging, la cual permite etiquetar a las palabras con la función sintáctica que cumple en una sentencia. Se analizó varias herramientas para realizar este proceso, tal como NLTK o Stanford NLP , pero por la abstracción y complejidad de los resultados retornados por las librerías estas herramientas fueron descartadas. Por otro lado, se consiguió resultados muchos más simples utilizando la librería Spacy, ya que en el idioma español engloba de manera general las tags de cada palabra, permitiendo utilizar la información de una manera simple, adicionalmente se alineaba perfectamente con la finalidad de la estructura que se tenia como objetivo. Aunque la simplicidad de esta herramienta fue fundamental en la decisión, el porcentaje de precisión de los modelos utilizados también fue un factor importante, ya que cuenta con un 96 % de exactitud en la asignación de POS Tagging [51]. La herramienta

provee el análisis sintáctico entre palabras a través de grafos que representan las interrelaciones entre palabras de una sentencia. Adicionalmente cuenta con una interfaz gráfica web para poder representar esas relaciones, lo cual mejoró absolutamente la comprensión de como utilizar la información y de esta manera permitió observar las problemáticas generales que pueden suceder cuando una oración no está conformada de una manera convencional o tiene una dependencia semántica con otra sentencia. Estos grafos pueden ser utilizados para poder encontrar relaciones entre los símbolos, como adjetivos de sustantivos o acciones por parte un sujeto.

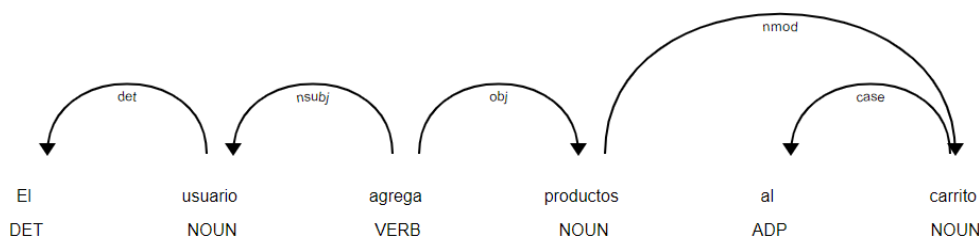


Figura 4.9: Análisis sintáctico de Spacy

El idioma español se destaca por tener un grado de orden sintáctico muy bajo[52]. Por esta razón, fue necesario las primeras etapas de la herramienta de reestructuración de las oraciones y acercarse a una estructura sintáctica básica como: Sujeto - Verbo - Objetos. Por lo tanto, en todo este capítulo se asumirá esta estructura para tratar la derivación de símbolos a partir de una sentencia.

A partir de estas aclaraciones, para cada sentencia se realizó el proceso de Tokenización, el resultado es un arreglo de palabras o «tokens» donde para cada uno de estos «tokens» se puede obtener la función sintáctica que tiene dentro del contexto de la oración como así las relaciones de dependencia que tiene con otras palabras.

Por cada sentencia, se iteró por cada una de sus palabras hasta identificar el primer verbo que es el objeto determinante para conseguir el sujeto y el predicado. Esta funcionalidad es un procesamiento simple, donde se analiza cada token hasta encontrar un token que tenga el TAG de verbo (VERB). Luego de identificar la posición de este verbo se puede dividir estructuralmente la sentencia en dos. Por un lado se acumularán todos los sujetos y por otro los predicados. A continuación se muestra un pequeño ejemplo de este procesamiento donde se va avanzando por cada parte de la estructura de la sentencia hasta recuperar todos los elementos del predicado:

```
1
2 def sujeto_y_predicado(doc):
3     sent ={}
4     // Empiezo agregando elementos a la estructura del sujeto
5     key="SUJETO"
6     sent["SUJETO"] = []
7     sent["VERBO"] = []
8     sent["PREDICADO"] = []
9     for token in doc:
10        // Si todavía no encontré el verbo entro
11        if(VERB == token.pos and not sent["VERBO"]):
12            key = "VERBO"
13        // Si encontré el verbo cambio la key a predicado
14        // ya que el verbo es un solo elemento
15        elif(key == "VERBO"):
16            key = "PREDICADO"
17        // Agrego el token a la estructura correspondiente
18        sent[key].append(token)
19    // Retorno la sentencia dividida por sujeto y predicado
20    return sent
```

Con la información estructurada de tal manera, fue posible identificar cuales son los sujetos que interactúan con el sistema y contra cuales objetos actúan, identificando los sustantivos que actúan en cada parte de estas estructuras. Sin embargo encontrar sujetos y objetos dentro de las oraciones no significa que todos esos elementos son símbolos dentro de un LEL, ya que se pueden encontrar muchos sujetos y objetos que en realidad no tengan una importancia real en el texto.

Para conseguir este descarte de símbolos se utilizó una técnica de análisis estadístico de texto donde se buscan los sustantivos más frecuentes como parte de los sujetos y de los predicados. Para este procesamiento, se obtienen los 10 sustantivos más frecuentes de cada parte (sujeto y predicado) y se realiza una comparativa entre el sustantivo más frecuente y los demás, en este caso la herramienta filtra todos aquellos sustantivos frecuentes que tienen un 30% de ocurrencias menos que el sustantivo más frecuente. Para este análisis se tuvo en cuenta la eliminación de las palabras stopwords [3], ya que sin realizar este filtro la salida hubiese tenido una cantidad muy grande de falsos positivos. Luego de este proceso se consiguen los sustantivos más frecuentes de los sujetos y los sustantivos más frecuentes del predicado, que la herramienta los cataloga como símbolos de sujetos y objetos respectivamente.

Una vez conseguido los dos grupos símbolos del texto, el sistema devuelve a la

pantalla principal los resultados obtenidos. Donde se encuentran los siguientes datos: Las palabras más importantes a través de un word-cloud (nubes de palabras), un resumen de todos los objetos, sujetos y verbos encontrados y por último muestra los sujetos y objetos encontrados durante el procesamiento. Para realizar el wordcloud se utilizó la librería de Python llamada «wordcloud» que recibe como parámetro las palabras del texto ingresado y la ocurrencia de las mismas. De esta manera sencilla el usuario tiene una noción de cuales palabras fueron relativamente importante para el análisis de símbolos y de la calidad del input de entrada.

4.6.6. Interacción del usuario con los símbolos

Una vez que el usuario tiene a su disposición la nueva vista con la información, el sistema provee una interfaz para que el usuario tenga la posibilidad de poder agregar o eliminar cualquiera de los símbolos encontrados, esto se realiza para tener una retro-alimentación del usuario y además para que el mismo pueda influir subjetivamente en las próximas etapas de procesamiento. Este tipo de feedback es importante para todo sistema que realiza un proceso automatizado, ya que siempre se espera un margen de error y este tipo de interacción permite poder mejorar y corregir detalles del proceso.

Esta etapa del proceso de la herramienta se considera como la más importante, ya que una vez definido los «sujetos» y «objetos», el sistema utiliza estos elementos para buscar relaciones, impactos y más símbolos dentro de la entrada de datos. Por lo tanto, sin la interacción correspondiente por parte del usuario o con símbolos erróneos por parte de la herramienta, el sistema podría tener sujetos o verbos que no sean de real importancia dentro del sitio, y en consecuencia el sistema buscaría información que no es relevante para el usuario. Esto podría ocurrir por varias situaciones, una de ellas puede ser por que la entrada de datos no es de gran calidad o los sujetos esperados a encontrar no son un factor importante en la conformación de las oraciones. Sin embargo, la madurez del desarrollo es baja y se tiene en cuenta un margen de error considerable.

Estas interacciones se realizan se través de la interfaz web donde el usuario puede corregir, eliminar o agregar nuevos símbolos que considere fundamentales. El sistema posee una entrada de datos por cada tipo de sustantivo (sujetos,objetos) donde puede escribir algún nuevo sustantivo y agregar cuantos el usuario crea conveniente. Así mismo existe la acción de eliminar un sustantivo que no se considere necesario. Todo estos cambios son guardados y tenidos en cuenta para los próximos pasos de procesamiento.

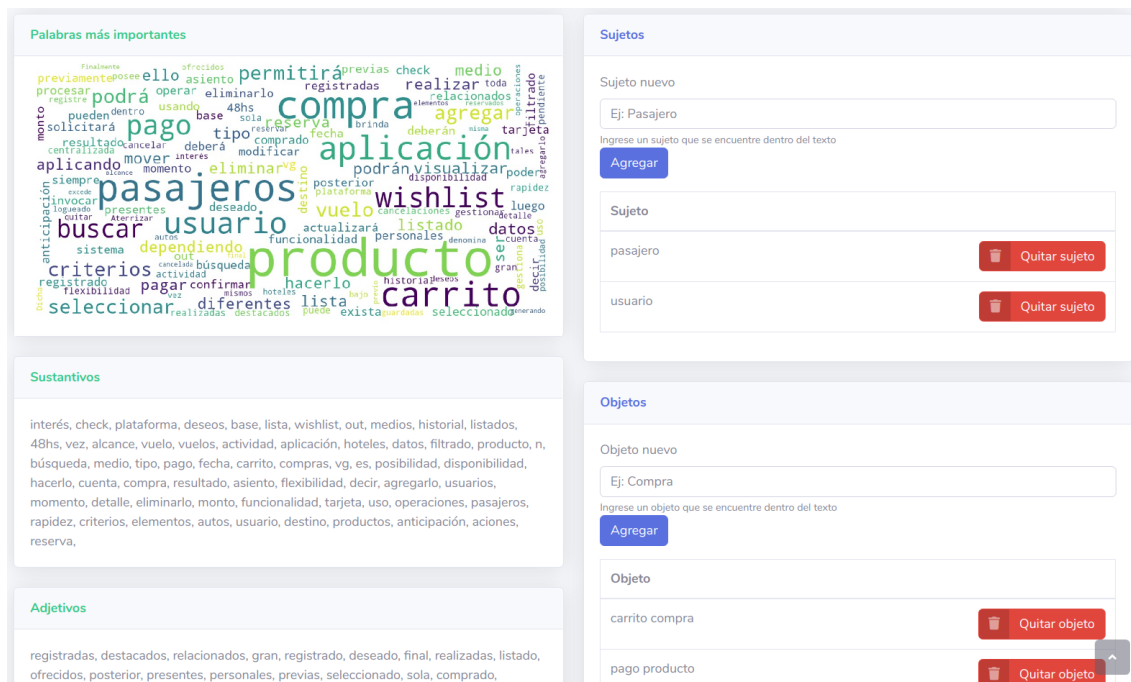


Figura 4.10: Vista de objetos y sujetos encontrados

Durante el desarrollo se analizó en utilizar modelos de aprendizaje automático para tener una manera de predecir que símbolos el usuario iba agregar o eliminar. Pero al tratarse de una especificación de dominio, tanto los sujetos como los objetos son dependientes del mismo, no existe una característica general para poder general un modelo preciso, por lo menos con la cantidad de corpus que se tenía. Sin embargo, para futuros trabajos se espera poder realizar una herramienta que se pueda amoldar a distintos dominios. De esta manera poder entrenar varios modelos de aprendizaje automático y que se tenga distintos modelos para cada tipo de dominio. Así mismo contar con distintas ontologías para identificar objetos o sujetos comunes ligados a un dominio específico.

Por último cuando el usuario termina de interactuar con los distintos símbolos, el sistema guarda la información modificada para la próxima etapa de la herramienta.

4.6.7. Búsqueda de relaciones

Una vez que el sistema tiene a su disposición conjuntos de símbolos de sujetos y objetos a partir de un texto, el mismo es capaz de inferir otros tipos símbolos relacionados, tales como verbos y estados. Antes de entrar en detalle con la búsqueda de relaciones entre los símbolos definidos, es importante remarcar que la herramienta

se centra en hallar los impactos para los sujetos encontrados. En otras palabras encontrar cuales son las acciones que realizan los sujetos y de esta manera identificar contra que objetos actúan y que cambios se realizan sobre los mismos. Para realizar este proceso el sistema encuentra aquellas sentencias donde se encuentre alguno de los sujetos definidos durante la etapa anterior, luego busca los verbos que estén asociados a estos sujetos e identifica sobre que objetos actúan y y cuales son los cambios que estas acciones provocan en los objeto también denominados estados. Para que estas sentencias puedan convertirse en un impacto, deberán contar con al menos un sujeto y un objeto del conjunto de símbolos definidos entre el usuario y la herramienta.

Para conseguir estas relaciones el sistema repite el proceso de separar lógicamente al texto ingresado por sentencias, en el cual se transforma el texto ingresado en un arreglo de sentencias través de la librería NLTK y lograr procesar cada una de ellas independientemente. Lo primero que realiza la herramienta es filtrar aquellas sentencias que cumplan la condición de tener por lo menos algún sujeto y objeto encontrado anteriormente. En caso de que cumpla esta condición, la sentencia pasa a ser una sentencia candidata para generar un impacto, pero primero busca si existen otros tipos de símbolos dentro de la oración. El primer símbolo a buscar es el verbo, es decir la acción que realiza el sujeto, para ello se utilizó nuevamente Spacy realizar el proceso de Pos-Tagging, el cual permite identificar los distintos tipos de elementos sintácticos de la oración. Por lo tanto, se itera por cada una de las palabras de la sentencia en cuestión buscando la palabra que corresponda con la categoría verbo. Así mismo es importante que esta acción o verbo sea realizada por el sujeto y que no sea un verbo aislado. Para identificar este requerimiento, la aplicación busca que exista una relación de dependencia, en este caso la dependencia nsubj entre el verbo y el sujeto, esta relación significa una relación entre un una palabra (verbo,adjetivo,sustantivo) y un sujeto o sustantivo, en caso de que sea un verbo esta relación define que el sujeto es el que realiza la acción. Por ejemplo si es un adjetivo, define que ese adjetivo corresponde al sujeto en cuestión. Una vez encontrado al verbo se lo agrega como otro símbolo de la oración.

Identificado el verbo del impacto candidato, la herramienta procede a identificar sobre que objeto actúa y adicionalmente sobre que otros objetos puede estar trabajando. Por ejemplo, dado una sentencia como: «Un usuario agrega vuelos a su carro de compra». Los símbolos que se pueden identificar son: usuario como sujeto, agrega/agregar como verbo y vuelos como objeto principal, sin embargo el carro de compra es un objeto secundario ya que también es parte de la sentencia y también forma parte de la semántica de la oración. Para ello la herramienta busca todas las frases nominales de la sentencia luego del verbo entonces, se puede inferir que la primera frase nominal es el afectado por la acción del sujeto en cuestión. Adicionalmente la herramienta toma al restos de los sustantivos como objetos secundarios, esto permite

dos situaciones muy importantes; la primera es poder identificar relaciones entre objetos de un LEL y como interrelacionan entre si, y adicionalmente permite garantizar un contexto en el cual se esta trabajando. Estos objetos secundarios pasan por un procesamiento adicional, donde se busca encontrar objetos compuestos o complejos, como por ejemplo: «carro de compras», ya que el mismo esta conformado por dos sustantivos.

Luego de encontrar una sentencia que haya pasado por los distintos filtros, la herramienta comienza el proceso de encontrar el cuarto símbolo para completar el impacto. Este tipo de procesamiento puede ser problemático ya que los cambios de estado de un objeto no suelen estar explícitamente en el contenido de un texto. La mayoría de las veces uno infiere estos cambios de estados a partir de acciones que lo indican, como lo pueden ser: comprado, reservado, alquilado, etc. Por lo tanto, se utilizaron ontologías que nos permita inferir el estado final de un objeto a través del verbo en cuestión. En este caso se utilizó arreglos de verbos que determinan cuando se cambia el estado a un objeto, por ejemplo: «El usuario agrega productos al carro de compra»; el estado final del producto una vez que el usuario realiza la acción es «agregado», y esta inferencia se guarda como el cuarto símbolo del impacto perteneciente a la sentencia.

Una característica deseada fue poder encontrar las distintas interacciones de acuerdo a los distintos tipos de sujetos, es decir, un sujeto puede tener distintos perfiles, como por ejemplo, un Usuario Operador no hace lo mismo que un Usuario Administrador. Entonces al presentarse esta problemática, se encontró la manera de encontrar la relación entre un sujeto/sustantivo y un adjetivo a través de relaciones de dependencias, tal como se hizo con los verbos y los objetos. Por lo tanto, una vez identificado cual es el sujeto, la herramienta es capaz de obtener cuales son los adjetivos que modifican a un sustantivo a través del grafo de dependencias, y de esta manera conseguir distintos estados o perfiles que puede llegar a tener un sujeto.

Antes de presentar al usuario los distintos impactos, la herramienta realiza dos procesamientos adicionales, el primero es identificar si el verbo obtenido es la acción real de un sujeto, por ejemplo: «El usuario realiza la compra de un producto», si lo analizamos sintácticamente «realiza» sería el verbo asociado al sujeto, pero la verdadera acción sería «compra», por lo tanto con la ayuda de una ontología que identifica esta clase de «falsos» verbos se realiza la derivación para que quede como verbo principal de la sentencia la verdadera acción, en este caso «compra», que da como resultado: «El usuario compra un producto». Por último se utilizó un clasificador de texto para determinar si un impacto puede ser válido o no. Para ello se investigó modelos de aprendizaje automático que funcionen de manera óptima para clasificar entrada de tipo texto, de los cuales se encontraron modelos utilizados para análisis de

sentimiento y análisis de significados de opinión [53]. Estos modelos se retro-alimentan a partir de entradas de texto y opiniones de cada una de ellas, el procesamiento que realizan estos modelos es preponderar las palabras utilizadas en cada una de las sentencias de texto con respecto a la opinión final. Es decir, que identifican que palabras influyen directamente sobre un resultado y utilizan estas preponderaciones aprendidas para poder predecir nuevas sentencias a futuro. La cantidad y calidad de información con la que aprenden estos modelos es fundamental para conseguir mejores resultados y obtener preponderaciones más certeras. Por lo tanto, una vez que un modelo está cargado, el mismo es capaz de determinar que tipo de resultado u opinión es una entrada de texto dado la preponderación que tenga las palabras utilizadas.

El modelo utilizado por esta aplicación fue el Modelo de Bayes, ya que se destaca por tener muy buenos resultados con entrada de datos de tipo texto [53], adicionalmente es un modelo simple, ya que utiliza como base el Teorema estadístico de Bayes de probabilidad condicional [54]. Para el aprendizaje de este modelo se utilizó 5 documentos de especificaciones de requerimientos que se tenían a disposición, con la misma herramienta aún sin el módulo de aprendizaje automático se recuperaron los impactos relacionados a cada documento y para cada una de estas observaciones se le dio una opinión (1=positiva, 0=negativa), esto se realiza para representar que un impacto es correcto o no respectivamente. Se capturaron 500 observaciones aproximadamente, de las cuales la mayoría fueron recuperadas de los documentos y otras fueron creadas para tener un dataset más amplio. El modelo se lo entrenó utilizando el 80 % de las observaciones y testeando con el 20 %. En el próximo capítulo se explicará en detalle las consideraciones tenidas en cuenta y los resultados de este aprendizaje. Una vez constituido el modelo, la herramienta lo utiliza para evaluar que impactos obtenidos en esta etapa considera correctos y cuales no antes de enviarlos a la vista final, es decir que aquellos impactos que el modelo prediga que es negativo no se mostrará como resultado.

A continuación se deja un extracto de como la herramienta predice como un impacto es correcto o incorrecto a partir del modelo precargado de bayes:

```
1
2 # Ejemplo de clasificador
3
4 def predict(sentences):
5     # Estructuras para ponderar a las nuevas sentencias
6     encVzer = pickle.load(
7         open("vector_clasificador_impacto.pkl", "rb")
8     )
9     encTfmer = pickle.load(
```

```
10         open("tfidf_clasificador_impacto.pkl", "rb" )
11     )
12     # Transformar sentencias de analizar
13     notas_new_counts = encVzer.transform(sentences)
14     notas_new_tfidf = encTfmer.transform(notas_new_counts)
15     # Abro clasificador
16     f = open('clasificador.pkl', 'rb')
17     clf = pickle.load(f)
18     f.close()
19
20     # Predict test data
21     pred = clf.predict(notas_new_tfidf)
22     predictions=[]
23     # Categorías
24     # 1 - Es un impacto
25     # 2 - No es un impacto
26     for review, category in zip(sentences, pred):
27         predictions.append(category)
28     # Retorno las predicciones a la herramienta
29     return predictions
```

Con respecto a la salida de la información, se tuvo énfasis en distintos aspectos para obtener una vista más simple para el usuario. Primero se evitó la duplicación de sustantivos normalizando todos los sustantivos a singular, de esta manera representar de manera única a un tipo de palabra. A su vez, se utilizó una técnica llamada lematizador[3] que se utiliza para inferir la forma base de una palabra, como por ejemplo: Si contamos con el siguiente vector de palabras [«guardando», «guardado», «guardaba»] todos los elementos comparten el «lemma» de «guardar». Esto permitió poder unificar distintas expresiones de un mismo verbo y no tener repeticiones de símbolos que ya han sido encontrados. Para evitar que no existan impactos repetidos se utilizó estructuras de datos tipo conjunto, que solo permite tener elementos únicos en el.

4.6.8. Búsqueda de nociones y retroalimentación

Luego de que el sistema recopile los distintos símbolos del texto, la herramienta busca las distintas definiciones que puede haber de cada palabra. Esto puede ser confundido o relacionado con las nociones, pero como se explicó la noción siempre dependerá del dominio o el contexto en el cual la palabra este siendo utilizada. Sin

embargo, como un avance existe la búsqueda de estas definiciones a la base de datos de Multilengual Central Repository para mostrar en pantalla.

En la vista final de resultados, la plataforma dispone los distintos símbolos encontrados con sus definiciones y los impactos encontrados del LEL con sus respectivos sujetos, verbo, objetos y estado final. Para retro-alimentar el clasificador de impactos, la herramienta provee un sistema de opiniones para que el usuario pueda expresar entre positivo y negativo a un impacto determinado. Esto permite que luego se puedan guardar estas elecciones de los resultados en una base de datos para luego crear un nuevo clasificador con esta información, esto permite un crecimiento continuo de la información que utiliza la herramienta impactando directamente en la calidad de los resultados. La vista permite identificar cuales sentencias el usuario considera descartables, implementando una coloración de las filas, asignando un color rojizo para determinar si es una sentencia descartada. Luego que el usuario define cuales impactos identificó como correctos, pasa a la vista final de la herramienta donde tiene la posibilidad descargar documentos de texto de formato CSV los símbolos e impactos encontrados.

Queda como a trabajo a futuro exportaciones complejas o generar nueva información a partir de estos símbolos, ya sea buscar grafos de conocimiento o exportar tipos de documentos que sirvan para la ingeniería de software como por ejemplo: modelos de clases, casos de uso, etc.

Impactos encontrados

Opinion	Sujeto	Acción	Objeto principal	Objeto relacionados
Opinion ▾	Pasajero	Buscar	Producto	
Opinion ▾	Pasajero	Aplicar	Criterio	
Opinion ▾	Pasajero	Depender	Tipo	Producto
Opinion ▾	Usuario No Registrado	Buscar	Producto	
Opinion ▾	Usuario No Registrado	Aplicar	Criterio	
Opinion ▾	Usuario No Registrado	Depender	Tipo	Producto
Opinion ▾	Usuario	Seleccionar	Producto	
Opinion ▾	Pasajero	Usar	Base	Filtrado Compra Previa
Opinion ▾	Pasajero	Visualizar	Producto	Relacionado Deseado
Opinion ▾	Pasajero	Agregar	Producto	Carrito Compra Pago
Opinion ▾	Pasajero	Eliminar	Producto	Carrito Compra Eliminarlo L
Opinion ▾	Pasajero	Agregar	Producto	Seleccionado Vg
Opinion ▾	Pasajero	Eliminar	Producto	Wishlist

Sujetos

Sujeto

- Pasajero**
Glosario externo de "Pasajero": Multiwordnet
- Usuario**
Glosario externo de "Usuario": Multiwordnet
- Usuario No Registrado**
Glosario externo de "No": Multiwordnet
Glosario externo de "Registrado": Multiwordnet

Objetos

Objeto

- Carrito Compra**
Glosario externo de "Carrito": Multiwordnet
Glosario externo de "Compra": Multiwordnet
- Pago Producto**
Glosario externo de "Pago": Multiwordnet

Figura 4.11: Impactos encontrados

Capítulo 5

Experimentación

Este capítulo abordará todas las pruebas realizadas para validar los resultados generados por la herramienta. Estas evaluaciones se dividen en dos núcleos importantes, por un lado se realiza un análisis detallado de la construcción del módulo del algoritmo de aprendizaje automático, donde se realiza una conclusión de las características del clasificador utilizado. Por el otro lado, se realiza una comparativa detallada entre los resultados esperados a partir de documentos realizados por especialistas con los devueltos por la herramienta, analizando el porque de las fallas y los aciertos de la misma.

5.1 Construcción del clasificador de Naive Bayes

5.1.1. Introducción

El objetivo principal del módulo de aprendizaje automático dentro de este trabajo fue detectar impactos que se considerarán válidos, para ellos se tuvo que analizar en una primera instancia que elementos son importantes para que un impacto sea válido. Un impacto en un LEL está compuesto por 4 elementos, un sujeto, una acción, un objeto principal, y un estado, por lo tanto se tuvo que analizar cada una de las partes para saber cuales de estos símbolos son claves para determinar la validez de un impacto. El sujeto, quien realiza la acción, suele estar ligado a un dominio en particular, y tampoco es un indicador importante para medir el nivel de validez. Por el lado de las acciones, la cuestión toma otro enfoque debido a que hay muchos verbos que son transversales a distintos dominios de aplicación tales como «agregar» «modificar» «eliminar», por lo tanto tener un conocimiento de las distintas acciones que se realizan sobre objetos es importante. Analizando los objetos, existe una dicotomía entre la relación de los objetos con el dominio, y algunos objetos que pueden

ser transversales a distintos dominios. Por ejemplo, si un sitio es de tipo e-commerce, el término «producto» se repetirá reiteradas veces sin importar el dominio de las ventas, por lo tanto puede agregar un valor. Por otro lado, también existen otros objetos transversales a distintos dominios tales como «fecha», «permisos», «sesión». Por último existen los estados, que al estar directamente relacionados con las acciones del impacto, carece de sentido su incorporación. Sin embargo, el criterio principal que se utiliza para diferenciar entre impactos válidos y no válidos, es que se toma como válido aquellos impactos que describan una funcionalidad, similar al concepto de requerimientos funcionales y no funcionales, por ejemplo, un impacto no válido sería «mostrar pantalla» y uno válido sería «comprar productos».

5.1.2. Muestras de aprendizaje

Para la construcción de cualquier modelo de aprendizaje automático supervisado es necesario tener un conjunto de datos para su aprendizaje, adicionalmente estos datos tienen asociadas etiquetas de clasificación. Dichas etiquetas engloban características de cada una de las muestras, generando una relación entre cada componente de una muestra y la etiqueta final. La asignación de etiquetas es un trabajo manual realizado por un especialista o un conjunto de especialistas. Para este trabajo final, se necesitaba buscar un conjunto de muestras que representen que sentencias son consideradas impactos y cuales no.

Por lo tanto, para crear el conjunto de muestras se analizó la estructura por lo cual están contruidos los impactos. La estructura sintáctica de un impacto es que existe un «sujeto», un «verbo», y un «objeto directo» al cual el sujeto realiza una acción, también se encuentra un estado pero no se encuentra de manera explícita en el texto, si no que se termina infiriendo a través de acciones. Por lo tanto a partir de los cinco (5) documentos disponibles de especificaciones de requerimientos se realizó un proceso manual donde se identificaron 558 sentencias que coincidían con esta composición. Debido a la poca cantidad de documentos para realizar las pruebas, no se esperaba una gran cantidad de sentencias para el muestreo, sin embargo era pre-condición que era de conocimiento.

Para el etiquetamiento de la información se realizó un trabajo manual junto a los directores para evaluar si una sentencia contenía las características suficientes para ser un impacto o no. Luego del proceso de etiquetamiento se identificaron 111 impactos, y 447 sentencias que no tenían características de impactos. Lo cual derivó en una nueva problemática al tener una cantidad de muestras tan distintas entre las muestras de «impactos» y «no impactos» que se verá en las siguientes secciones. A continuación se ejemplifica los impactos tomados como válidos y no válidos para realizar el aprendizaje.

```
1 Impactos válidos :
2
3 "usuario guarda productos"
4 "usuario compra productos"
5
6 Impacto no válidos :
7
8 "pantalla muestra mensaje"
9 "servidor envía evento"
```

5.1.3. Preparación de la información para el clasificador de Bayes

Para que la información, en este caso las sentencias etiquetadas (válidos, no válidos), puedan utilizarse como entrada del clasificador de Bayes, siendo este un clasificador probabilístico, es necesario procesar y normalizar la entrada de la información de tal manera que el clasificador pueda realizar todos los cálculos necesarios para predecir una etiqueta a partir de los elementos de la sentencia (palabras). Para conseguir esto, el clasificador necesita que la entrada de cada sentencia o muestra sea una bolsa de elementos donde cada uno de estos elementos estén asociados a un valor específico o ponderación, este proceso es conocido como ponderación de la información. Como se vio en la sección [Ponderación de las palabras](#), la complejidad de realizar de esta tarea varía de acuerdo a los resultados esperados. Para ejemplificar el proceso de transformación, a continuación se verá en detalle el proceso de ponderar una muestra desde un proceso simple, que tiene el nombre «Ponderación de Frecuencia de Término» hasta realizar un proceso más complejo como la «Ponderación de Frecuencia de Término \times Frecuencia de Documento Inverso (TF \times IDF)» que es lo utilizado en esta tesina.

Aunque el proceso de realizar esta transformación puede ser compleja, dado los avances tecnológicos y de herramienta este proceso resulta fácil de elaborar por distintas librerías, en este caso se trabajó con la librería «sklearn». Por lo tanto dada un conjunto de sentencias como el siguiente:

```

root@463a45d71016:/app/tesislel/classifier# python3 sentimentanalysis.py
['caso mostrar mensaje',
 'resultado búsqueda permitirá usuario',
 'sistema expone listado',
 'usuario ingresa datos',
 'caso usuario recuerde correo',
 'productos pagando monto',
 'aplicación tener listado',
 'hoteles ingresando datos',
 'sistema verifica pago',
 'usuario coloque nombre',
 'aplicación gestiona cancelaciones',
 'datos ser tarjeta',
 'lista desplegada ordenar precio',
 'sistema permitirá pasajeros',
 'viajes librando usuarios',
 'paquetes ser preparados administrador',
 'usuario registrado consultar compras',
 'productos buscados utilizando aplicaciones',
 'botón Agregar carrito',
 'requerimientos agregar funcionalidades',
 'cliente accede beneficio',
 'sistema habilita cuenta usuario',
 'tarjeta utilizar ingresa datos',
 'aplicación devolverá resultados',
 'pago momento efectuar checkout',
 'usuario agrega producto',
 'aplicación tener lista',
 'compras previas ver detalle',
 'cliente selecciona opción',
 'sección introduce documento',
 'capacidad realizar funciones',
 'servicio utilizando buscador',
 'sistema permitirá usuarios',
 'aplicación permitirá buscar producto',
 'personas datos obligatorios realizar búsqueda',

```

Figura 5.1: Sentencias a ponderar

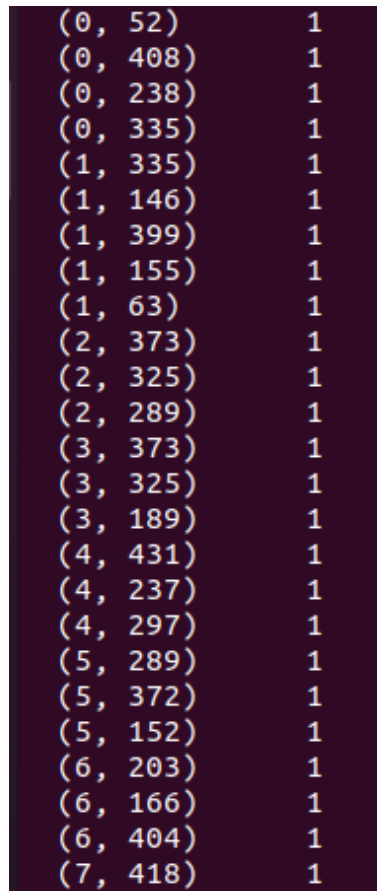
Sklearn lo que realiza es asignar un id a cada sentencia de aprendizaje, y luego para cada palabra asignarle un id, y hacer una referencia de palabra \leftrightarrow id. Por lo tanto si se vectoriza cada sentencia realizando una «Ponderación de Frecuencia» a través del siguiente código, donde el objeto «encVzer» guarda la relación entre el id creado y la palabra y devuelve como resultado la ponderación del conjunto de muestras:

```

1 # Use CountVectorizer
2 encVzer = CountVectorizer(tokenizer=nlk.word_tokenize)
3
4 docs_train_counts = encVzer.fit_transform(docs_train)
5 print(docs_train_counts)

```

Se llega a un resultado como la figura 5.2, donde los ids de la parte izquierda es la referencia al número de muestra, y el id consiguiente es id de la palabra y por último la frecuencia de esa palabra en esa sentencia. Entonces por ejemplo si se tiene una tupla (4,50) 5, quiere decir que en la sentencia 4 la palabra con id 50 tiene una repetición 5 veces. Esta sería la ponderación por palabras más simple ya que la palabra que tenga el valor más alto es el que tomará protagonismo a la hora de relacionarlo con la etiqueta final:



(0, 52)	1
(0, 408)	1
(0, 238)	1
(0, 335)	1
(1, 335)	1
(1, 146)	1
(1, 399)	1
(1, 155)	1
(1, 63)	1
(2, 373)	1
(2, 325)	1
(2, 289)	1
(3, 373)	1
(3, 325)	1
(3, 189)	1
(4, 431)	1
(4, 237)	1
(4, 297)	1
(5, 289)	1
(5, 372)	1
(5, 152)	1
(6, 203)	1
(6, 166)	1
(6, 404)	1
(7, 418)	1

Figura 5.2: Sentencias ponderadas por frecuencia

Si bien, esto sería considerado una ponderación de palabras, este tipo de solución no sería la más indicada, ya que le da muchísima importancia a la frecuencia de las palabras sin tener en cuenta el contexto en el que se encuentra lo cual disminuye en gran medida la posterior precisión de un clasificador. Por lo tanto se procedió a utilizar la ponderación «(TF×IDF)», la cual se lleva a cabo de la siguiente manera utilizando el CountVectorizer del paso anterior:

```
7 encTfmer = TfidfTransformer()  
8 docs_train_tfidf = encTfmer.fit_transform(docs_train_counts)  
9 print(docs_train_tfidf)
```

Teniendo como resultado la siguiente información donde el valor de cada elemento dependerá de la cantidad de elementos de la sentencia y de la frecuencia en la que se encuentra en el total de muestras:

(0, 408)	0.435610084468831
(0, 335)	0.3540276738755991
(0, 238)	0.6448091625960317
(0, 52)	0.5187768347483286
(1, 399)	0.43599636049087936
(1, 335)	0.2975359865436059
(1, 155)	0.5419178908391828
(1, 146)	0.5419178908391828
(1, 63)	0.3661004090779648
(2, 373)	0.598663812307257
(2, 325)	0.5759242454702242
(2, 289)	0.5566982156548503
(3, 373)	0.5634678328992379
(3, 325)	0.5420651454754755
(3, 189)	0.6234335404423541
(4, 431)	0.28826160655681543
(4, 297)	0.5819657345061715
(4, 237)	0.7604085283885078
(5, 372)	0.607131812438836
(5, 289)	0.4740742383745963
(5, 152)	0.6376868971793922
(6, 404)	0.48044280075245344
(6, 203)	0.6545326116459613
(6, 166)	0.5837480411076813
(7, 418)	0.6245064469444188

Figura 5.3: Ponderación por «(TF×IDF)»

A partir de esta información de muestras con bolsas de elementos ponderados y una etiqueta final asociada, el conjunto de datos está preparado para ser utilizado por el clasificador de Bayes.

5.1.4. Desbalanceo de datos y submuestreo (downsampling)

En los problemas de clasificación suele ocurrir que las muestras de información a través de las distintas clasificaciones estén desbalanceadas. Es decir, que se tiene muchas muestras de un tipo de dato, y pocos de otro/s. Esto provoca que a la hora de construir un modelo, los resultados del mismo con respecto a precisión, accuracy y otras medidas no sean tan reales como parecen.

Esto suena bastante razonable: si a una red neuronal le enseñamos 990 sentencias que no son impacto y sólo 10 de impactos, no se puede pretender que la misma logre diferenciar una clase de otra. Lo más probable que la red se limite a responder que una sentencia nunca es un impacto, ya que en los resultados de entrenamiento se tuvo como resultado un 99 % de precisión. Por lo tanto, si se construye un modelo con un desbalanceo presente, el mismo dará una falsa representación de que tan bien reconoce un tipo de dato de otro.

Para resolver el desbalanceo de datos, existen varios enfoques estudiados, pero en este trabajo de fin de grado se utilizó el downsampling como mecanismo de resolución. Este tipo de enfoque consiste en capturar de manera aleatoria muestras de la clase mayoritaria una cantidad igual a la cantidad de muestras que haya de la muestra minoritaria. En este caso se trata de capturar 111 sentencias de tipo "no impacto" de manera aleatoria y de esta manera construir el clasificador con la misma cantidad de elementos de cada parte. Sin embargo como se estaría capturando de manera

aleatoria una muestra de la clase mayoritaria es necesario realizar varias pruebas para poder sacar un número «real» acerca de los resultados finales del clasificador. Por lo general, se utiliza la proporción que haya de diferencia entre las clases para realizar un «n» número de downsampling para obtener resultados y por último promediarlos para calcular accuracy, precision, recall y f1 que se espera tener de cualquiera de los downsampling. De todas maneras, ante la poca cantidad de información y para obtener un resultado más robusto se realizó un k-fold cross validation combinado con la técnica de downsampling, que se verá en la próxima sección asociando lo visto hasta ahora.

5.1.5. K-Fold Cross Validation

La validación cruzada es un procedimiento de re-muestreo que se utiliza para evaluar modelos de aprendizaje automático en una muestra de datos limitada. El procedimiento tiene un solo parámetro llamado k que se refiere al número de grupos en los que se dividirá una muestra de datos determinada. Como tal, el procedimiento a menudo se denomina validación cruzada de k veces. Cuando se elige un valor específico para k, se puede usar en lugar de k en la referencia al modelo, por ejemplo, k = 10 se convierte en una validación cruzada de 10 veces.

La validación cruzada se utiliza principalmente en el aprendizaje automático aplicado para estimar la habilidad de un modelo de aprendizaje automático en datos invisibles. Es decir, utilizar una muestra limitada para estimar cómo se espera que funcione el modelo en general cuando se utiliza para hacer predicciones sobre datos no utilizados durante el entrenamiento del modelo.

Es un método popular porque es fácil de entender y porque generalmente da como resultado una estimación menos sesgada o menos optimista de la habilidad del modelo que otros métodos, como una simple división de entrenamiento / prueba. El procedimiento general es el siguiente:

1. Mezcle el conjunto de datos de forma aleatoria.
2. Divida el conjunto de datos en k grupos
3. Para cada grupo único:
 - a) Tome el grupo como un conjunto de datos de prueba o de reserva
 - b) Tome los grupos restantes como un conjunto de datos de entrenamiento
 - c) Coloque un modelo en el conjunto de entrenamiento y evalúelo en el conjunto de prueba
 - d) Conserve la puntuación de la evaluación y descarte el modelo

4. Resumir la habilidad del modelo usando la muestra de puntajes de evaluación del modelo

Es importante destacar que cada observación en la muestra de datos se asigna a un grupo individual y permanece en ese grupo durante la duración del procedimiento. Esto significa que a cada muestra se le da la oportunidad de ser usada en el set 1 de reserva y usada para entrenar el modelo $k-1$ veces.

Aunque este es el procedimiento normal de un K-Fold Cross Validation, que sucede cuando se encuentra un downsampling ya que se tiene una muestra reducida de la clase mayoritaria? Para solucionar este inconveniente se realiza un K-Fold Cross Validation por la diferencia que haya entre la clase mayoritaria y la minoritaria. Es decir, en este caso se tiene 111 sentencias de impactos validos y 447 sentencias que no son impactos, por lo tanto la cantidad de veces que se debería realizar el K-Fold Cross Validation serían 4, donde en cada una de esas validaciones se realiza un downsampleo distinto de la clase mayoritaria de manera aleatoria, de esta manera, nos aseguramos que cubramos la mayor cantidad de muestras durante la experimentación. Por lo tanto, en este caso se utilizo un $K=5$ lo que indica 5 grupos de aproximadamente 22 muestras cada uno, y se realizaron 4 downsamples, lo que indica que se realizaron un total de 20 experimentos para calcular los resultados del clasificador. A continuación se encuentra el código para realizar los experimentos y en la sección final se analizarán los resultados de estos experimentos.

```

1 recall=[]
2 pre=[]
3 acc=[]
4 f1=[]
5 cant_downsampling=4
6 k=5
7 max_array = [-99,-99,-99,-99]
8 for i in range(cant_downsampling):
9     # Downsampling of class 0
10    w_class0_downsampled = np.random.choice(
11        w_class0 , size=n_class1, replace=False)
12    # X, y from sampling
13    y_sampling=np.hstack((y[w_class1], y[w_class0_downsampled]))
14    X_sampling=np.hstack((X[w_class1], X[w_class0_downsampled]))
15
16    # Shuffle X data and labels
17    X_sampling, y_sampling = shuffle(X_sampling, y_sampling)
18    # K-Fold of 5
19    kfold = KFold(n_splits=k, shuffle=True)

```

```
20 print("=====")
21 print("Downsampling N: ", i+1)
22 for train, test in kfold.split(X_sampling):
23     docs_train, docs_test =
24         X_sampling[train], X_sampling[test]
25     y_train, y_test =
26         y_sampling[train], y_sampling[test]
27     # Use CountVectorizer
28     encVzer = CountVectorizer(
29         min_df=1, tokenizer=nlk.word_tokenize)
30     docs_train_counts = encVzer.fit_transform(docs_train)
31     # Use TfidfTransformer
32     encTfmer = TfidfTransformer()
33     docs_train_tfidf = encTfmer.fit_transform(docs_train_counts)
34     # Transform documents
35     docs_test_counts = encVzer.transform(docs_test)
36     docs_test_tfidf = encTfmer.transform(docs_test_counts)
37     # Classifier Multinomial
38     clf = MultinomialNB()
39     clf.fit(docs_train_tfidf, y_train)
40     y_pred = clf.predict(docs_test_tfidf)
41     # Calculate accuracy, precision, recall, f1
42     acc.append(accuracy_score(y_test, y_pred))
43     pre.append(precision_score(y_test, y_pred))
44     recall.append(recall_score(y_test, y_pred))
45     f1.append(f1_score(y_test, y_pred))
46     # Calculate best classifier
47     new_a=[acc[-1:][0], pre[-1:][0], recall[-1:][0], f1[-1:][0]]
48     if( new_a >= max_array):
49         max_array = new_a
50         print("Change max", max_array )
51         # save the classifier
52         with open('classifier.pkl', 'wb') as fid:
53             pickle.dump(clf, fid)
54
55         # save the tfidf
56         with open("tfidf_result.pkl", 'wb') as handle:
57             pickle.dump(encTfmer, handle)
58
59         # save the tfidf
```

```

60         with open("encvzer_result.pkl", 'wb') as handle:
61             pickle.dump(encVzer, handle)
62
63         print("Accuracy: " , accuracy_score(y_test,y_pred))
64         print("Recall: " , recall_score(y_test,y_pred))
65         print("Precision: " , precision_score(y_test,y_pred))
66         print("F1: " , f1_score(y_test,y_pred))
67
68
69     print("accuracy","precision", "recall", "f1", max_array)
70     print("Cantidad de experimentos: ", cant_downsampling*k)
71     print("Accuracy average:" , np.average(accuracy))
72     print("Precision average:" , np.average(precision))
73     print("Recall average:" , np.average(recall))
74     print("F1 averages:" , np.average(f1))
75     print("Accuracy standard desviation:" , np.std(accuracy))
76     print("Precision standard desviation:" , np.std(precision))
77     print("Recall standard desviation:" , np.std(recall))
78     print("F1 standard desviation:" , np.std(f1))

```

5.1.6. Resultados obtenidos

Una vez realizados los downsampling con sus respectivos K-Fold Cross Validation, se calcularon los accuracy, recall, precisión, y f1 promedio junto a su desviación estándar. El clasificador elegido como ejemplo para el uso de la herramienta fue el clasificador con mejores resultado de las 20 pruebas realizadas en la sección previa.

Medida	Accuracy	Precision	Recall	F1
Promedio	0.8054	0.7531	0.9163	0.8215
Desviación standard	0.0616	0.1023	0.0581	0.0652

Tabla 5.1: Resultado de las pruebas del clasificador

5.2 Resultados de la herramienta

5.2.1. Introducción

En la última sección de este capítulo, se analizarán los resultados obtenidos por parte de la herramienta, donde se verán los documentos de tipo texto utilizados como

entrada y la comparativa con los resultados esperados. Por último, el análisis final estará fuertemente ligado con el procesamiento determinista de la herramienta ya que previamente se ha evaluado el módulo de aprendizaje automático implementado.

5.2.2. Resultados de la herramienta comparados con los documentos LEL

Los documentos a continuación son trabajos de alumnos de la Maestría en Ingeniería de Software los cuales consistieron en definir documentos de especificaciones de requerimientos(SRS), un documento LEL y casos de usos de una aplicación de venta de vuelos que adicionalmente tiene paquetes promocionales de hoteles, vehículos, etc. Para realizar las pruebas se tomó como entrada de tipo de texto los SRSs y se comparó los resultados de la herramienta con los documentos LEL definidos por cada grupo de alumnos. Sin embargo es importante volver a remarcar que los resultados de la herramienta son orientados a los impactos de los sujetos del sistema, por lo tanto se espera la ausencia de ciertos elementos pertenecientes a impactos no contemplados.

Dado por sentado la importancia de ciertos elementos en los hallazgos de la herramienta a continuación se mostrarán los resultados comparativos entre los símbolos definidos en los documentos y los encontrados por la herramienta, teniendo en cuenta y analizando el porqué de los resultados.

Documento 1

- Los sujetos no encontrados se debió a que el input se centró en funcionalidades a partir de un usuario, y dentro de las especificaciones no se menciona los distintos sujetos.
- Los objetos no encontrados por la herramienta se ven directamente relacionados con requerimientos no funcionales o componentes de la aplicación (sección de confirmación, actividad sospechosa, uptime).
- Las acciones no encontradas se vieron relacionadas con requerimientos funcionales de la aplicación y no de un usuario en particular.
- Los estados no encontrados se dan porqué se necesitaría un análisis más complejo que solo derivar el estado a partir de un verbo. Ejemplo: cuando un producto esta disponible.

Símbolo	Documento	Herramienta	Porcentaje de acierto
Sujeto	4	1	25 %

Objetos	40	20	50 %
Acciones	12	9	75 %
Estados	6	3	50 %

Tabla 5.2: Resultados del primer documento

Documento 2

- De los sujetos definidos por el documento LEL, el 50 % no aparece como parte de la entrada de texto. Ya que la entrada se compone de funcionalidades vista desde una categoría «usuario» sin especificar los distintos usuarios que podría haber (conductor, pasajero). Existe un sujeto que no se vio encontrado por la herramienta por la poca frecuencia que aparece en la entrada.
- De las acciones no encontradas se dividen en 3 problemas distintos:
 - Tres impactos no fueron encontrados porque no se identificó un sujeto necesario.
 - Existió conflicto al hallar un impacto porque el verbo era una palabra en inglés (check-out)
 - No se identificó correctamente un impacto que debería haber estado en la salida.
- Los objetos faltantes se da por los errores en la identificación de sujetos, que luego influye en el resto del proceso.
- Un estado se vio reflejado ante la falla en el procesamiento de identificar impactos. Otro estado no se vió identificado por la falta de complejidad en el reconocimientos de estados.

Símbolo	Documento	Herramienta	Porcentaje de acierto
Sujeto	6	2	33 %
Objetos	16	5	31 %
Acciones	19	14	73 %
Estados	8	6	75 %

Tabla 5.3: Resultados del segundo documento

Documento 3

- De los objetos no encontrados, la herramienta no fue capaz de encontrar tres elementos pertenecientes a una super clase (productos: hoteles, vuelo, auto). Luego un elemento fue parte una oración demasiado compleja de analizar para la herramienta y por último un elemento no se encontraba en el input ingresado.
- Dos acciones no encontradas estaban descritas de una manera compleja para ser analizadas por la herramienta. Y dos acciones más no se encontraban dentro del input ingresado.
- El estado no encontrado fue por una oración muy compleja de analizar.

Símbolo	Documento	Herramienta	Porcentaje de acierto
Sujeto	2	2	100 %
Objetos	11	6	54 %
Acciones	10	6	60 %
Estados	1	0	0 %

Tabla 5.4: Resultados del tercer documento

Documento 4

- Los sujetos encontrados fueron los únicos descritos por el input.
- El 70 % de los objetos que no se encontraron no estaban dentro del input de la herramienta. El 30 % restante fueron errores de la herramienta, ya que falló en la búsqueda de impactos debido a que no pudo identificar el sujeto «persona» (No formaba parte del conjunto de símbolos definidos). Por otro lado hubo objetos que no se tuvieron en cuenta por formar parte de funcionalidades de la aplicación.
- Las acciones que no se encontraron fueron producto de que no se encontraban dentro del input y por la falla en el reconocimiento de un sujeto (Que no formaba parte del conjunto de símbolos definidos).
- El estado no encontrado fue producto de ser un estado complejo (producto disponible)

Símbolo	Documento	Herramienta	Porcentaje de acierto
Sujeto	6	2	33 %

Objetos	27	6	22 %
Acciones	22	15	68 %
Estados	4	3	75 %

Tabla 5.5: Resultados del cuarto documento

Documento 5

Había un quinto documento para analizar pero debido a que la estructura del texto no cumplía con ciertos criterios sintácticos la herramienta no fue capaz de procesarla ya que eran funcionalidades enumeradas. En este caso, el input solo describía funcionalidades de un sistema sin hacer referencia a un usuario en particular. Ejemplo: - Agregar productos al carrito.

5.3 Conclusión de resultados

Dado los resultados del clasificador propuesto para la tesis y los resultados deterministas de la herramienta, se puede concluir que los mismos fueron satisfactorios. En primer lugar, como se utiliza un algoritmo de aprendizaje automático, el uso de la misma y el uso de los resultados obtenidos como retroalimentación generará que el output de la herramienta vaya mejorando. Por otro lado se pudo buscar una causa raíz a todas las ausencias de símbolos e impactos en la comparativa entre los documentos LEL y los resultados de la herramienta, que a partir de un proceso de correcciones, los resultados pueden ir mejorando a medida que se van agregando nuevas reglas o nuevos procesos para detectar más símbolos e impactos.

Capítulo 6

Conclusiones

Culminando con la escritura de esta tesina se irán detallando los puntos destacados de este trabajo de fin de grado realizando un pequeño resumen de todos los capítulos vistos hasta ahora y por último realizando una conclusión que ha dejado este proyecto.

Tal como se explicó en las primeras secciones, los analistas deben consumir y analizar una gran cantidad de información sobre el dominio y requerimientos de un proyecto para construir los diferentes documentos de ingeniería de software que se utilizan a lo largo del ciclo de vida de desarrollo. Como se sabe, la calidad de estos documentos son de vital importancia para el éxito de un desarrollo e implica un gran esfuerzo por parte de los analistas. Por esta razón, se elaboró una herramienta que sea capaz de tamizar y sintetizar toda la información de un dominio de manera tal que facilite o asista a los analistas.

En este trabajo, se implementó una solución semi-automatizada para crear documentos LEL a partir de texto ingresado por el usuario, donde la información ingresada debe contener detalles de requerimientos o de dominio de una aplicación. Estos documentos LEL son glosarios que especifican de manera detallada cada elemento de un dominio (símbolos) y sus posibles relaciones entre sí (impactos). Donde estos documentos suelen utilizarse para tener un nexo comunicativo entre los analistas y los clientes.

Por lo tanto para elaborar este tipo de documentos de manera automática o semiautomática se investigaron distintas derivaciones desde entrada de tipo texto hacia modelos que se utilicen actualmente para el proceso de ingeniería de software. En esta investigación realizada se encontraron derivaciones desde texto a modelos de clases (UML), casos de uso, modelos entidad relación e incluso a glosarios, por supuesto con enfoques distintos a los documentos LEL. De estas investigaciones, se concluyó que el procesamiento de lenguaje natural y las reglas heurísticas eran el núcleo central de cualquiera de estos proyectos. Sin embargo, dichos procesos deterministas fueron quedando insuficientes a medida que el proceso de recolección de información se iba

transformando en un proceso más complejo. Por lo cual surgieron nuevas alternativas para mejorar los resultados intrínsecos de este tipo de procesamiento, que desembocó, en la mayoría de los casos, en la inclusión de algoritmos de aprendizaje automático supervisados, los cuales permiten predecir o clasificar nueva información a través de un conocimiento previo.

Con estas tecnologías se diseñó una herramienta que sea capaz de obtener información de tipo texto por parte del usuario, y que a través un módulo principal de procesamiento de lenguaje natural y reglas heurísticas el usuario final reciba símbolos e impactos (elementos de un documento LEL) que se encontraron en la entrada de datos. Sin embargo, ningún proceso determinista es perfecto y nunca se podrá abarcar todas las combinaciones sintácticas posibles que se pueden dar en un idioma particular. Para la mitigación y reducción de errores en el proceso determinista de la herramienta se incluyó un módulo de aprendizaje automático supervisado que tenga la capacidad discernir que sentencia de texto podría llegar a ser un impacto y cual no. Si bien no solucionará todos los posibles errores de salida por parte de la herramienta ya que el mismo módulo cuenta con sus propios falsos positivos, el algoritmo podrá seguir aprendiendo desde la interacción del usuario para ir mejorando gradualmente los resultados. Para tener una mejor experiencia de uso por parte del usuario, se implementó sobre un sistema web, el cual permite realizar las funcionalidades descritas de una manera amigable para el usuario y que adicionalmente sea multiplataforma. De esta manera, un usuario con solo tener una entrada de texto puede tener rápidamente una vista general de como puede estar compuesto el documento LEL que tiene que conformar, incluso utilizar el resultado de la herramienta como base para desarrollar el documento final.

Dentro de las principales fortalezas que tiene esta herramienta, es que hasta el día de hoy no se encuentran herramientas en idioma español que provea las funcionalidades que provee este trabajo. Particularmente, en el área de los documentos LELs esta tesina sería el primer aporte para la generación automática a partir de una entrada de texto. Por otro lado, dado el análisis de los resultados en el capítulo de [Experimentación](#), en casi todos los casos de falsos negativos, se pudo encontrar la razón de estos negativos, e incluso la mayoría se dieron por falta de features encadenados que todavía la herramienta no posee. Por último, el uso de algoritmos de aprendizaje automático permite la mejora continúa de la salida de la información a medida que sea utilizada por expertos.

Sin embargo esta herramienta deja un gran marco de investigación a continuar, como se ha visto durante los análisis de los resultados, el procesamiento estadístico en la detección de símbolos puede fallar en caso que tengan pocas ocurrencias, por lo cual es un gran punto a analizar debido a que por ejemplo, los sujetos del sistema son los símbolos más importantes para reconocer al resto de los símbolos de un LEL, y sin ellos se puede perder una gran cantidad de información. Por otro lado, por la mecánica

y el objetivo de esta tesina, la herramienta no presenta módulos de integración para utilizarse en procesos más complejos o ser parte de otra herramienta. Por último, otra arista de investigación que debe profundizarse es la capacidad de detectar símbolos e impactos cuando el texto posee estructuras complejas y poco convencionales, y que puede derivar en el uso de nuevas tecnologías en la detección de los símbolos y la búsqueda de relaciones entre los mismos.

Un aporte significativo que se le puede incluir a la herramienta desarrollada es la capacidad de que el modelo predictor aprenda con el correr del tiempo. En estos momentos, un usuario de la herramienta haría uso de ella con un modelo predictor pre-entrenado. Sin embargo, es sabido que distintos dominios tienen sus propias particularidades, lo cual podría hacer que un modelo entrenado en un dominio no resulte efectivo en otro, en este caso el modelo predictor está entrenado en base a un dominio e-commerce de una aerolínea. Es por esto que contar con una funcionalidad que permita entrenar un modelo nuevo, o bien ir retro-alimentando al modelo con el propio uso de la herramienta, indicando que impacto está bien detectado y cual no, es de gran utilidad.

Para concluir este trabajo final, esta herramienta deja un puntapié inicial para que cualquier alumno o investigador pueda utilizarla como ayuda para la elaboración de documentos LELs, para integrarla a nuevos flujos de procesamiento, o directamente usar componentes de la herramienta para crear nuevas implementaciones. Es importante volver remarcar que esta herramienta es el primer aporte sobre como derivar texto a documentos LEL de manera semi-automática. Donde se presenta el punto de inflexión a partir de la publicación de este trabajo, en el cual los analistas podrán utilizar una herramienta como asistencia en la construcción de futuros documentos LEL.

Bibliografía

- [1] L. Antonelli, G. Rossi, J. C. S. do Prado Leite, y A. Oliveros, “Language extended lexicon points: Estimating the size of an application using its language,” 2014.
- [2] C. C. Aggarwal y C. X. Zhai, *Mining Text Data*, 2012.
- [3] J. Perkins, *Python 3 Text Processing with NLTK 3 Cookbook*, 2014.
- [4] E. K. Ikonomakis, S. Kotsiantis, y V. Tampakas, “Text classification using machine learning techniques,” *WSEAS TRANSACTIONS on COMPUTERS*, 2005.
- [5] B. W. Sorte, P. P. Joshi, y P. V. Jagtap, “Use of artificial intelligence in software development life cycle: A state of the art review,” *International Journal of Advanced Engineering and Global Technology Vol-03*, 2015.
- [6] J. Shabbir y T. Anwer, “Artificial intelligence and its role in near future,” *Journal of Latex class files - Vol 14*, 2015.
- [7] P. Yalla y N. Sharma, “Integrating natural language processing and software engineering,” *International Journal of Software Engineering and Its Applications Vol. 9*, 2015.
- [8] J. Lee, H. Davari, J. Singh, y V. Pandhare, “Industrial artificial intelligence for industry 4.0-based manufacturing systems,” *Manufacturing Letters*, 2018.
- [9] D. Zhang, “Machine learning and software engineering,” *Software Quality Journal*, 2003.
- [10] S. Macdonell, K. Min, y A. Connor, “Autonomous requirements specification processing using natural language processing,” 2014.
- [11] L. Antonelli, G. Rossi, J. C. S. do Prado Leite, y A. Oliveros, “Deriving requirements specifications from the application domain language captured by language extended lexicon. in proceedings of the workshop in requirements engineering,” *Buenos Aires, Argentina*, 2012.

-
- [12] L. Antonelli, G. Rossi, A. Oliveros, y J. C. S. do Prado Leite, “Buenas prácticas en la especificación del dominio de una aplicación,” 2013.
- [13] H. Ammar, W. Abdelmoez, y M. S. Hamdi, “Software engineering using artificial intelligence techniques: Current state and open problems,” *ICCIT*, 2012.
- [14] L. Antonelli, G. Rossi, y A. Oliveros, “A collaborative approach to capture the domain language,” 2015.
- [15] B. Manrique-Losada, “Procesamiento de lenguaje natural para adquisición de conocimiento: Aproximaciones desde la ingeniería de requisitos,” 2015.
- [16] A. B. Clegg, “Computational-linguistic approaches to biological text mining,” 2008.
- [17] D. Jurafsky y J. H. Martin, *Speech and Language Processing (3rd ed. draft)*, 2000.
- [18] R. Feldman y J. Sanger, *The text mining handbook*, 2000.
- [19] U. D. Developers, “Universal dependencies documentation,” 2014.
- [20] V. Nasteski, “An overview of the supervised machine learning methods,” 2017.
- [21] A. Abraham, “Comparison of supervised and unsupervised learning algorithms for pattern classification,” 2013.
- [22] M. W. Berry y A. H. Mohamed., *Supervised and Unsupervised Learning for Data Science*, 2019.
- [23] V. Nasteski, “Supervised and unsupervised machine learning techniques for text document categorization,” 2013.
- [24] J. Boustedt, “Ways to understand class diagrams,” 2010.
- [25] G. A. Mala y G. Uma, “Automatic construction of object oriented design models [uml diagrams] from natural language requirements specification,” *Lecture Notes in Computer Science*, 2010.
- [26] M. I. R. Ahmad, “Class diagram extraction from textual requirements using natural language processing (nlp) techniques,” *Second International Conference on Computer Research and Development*, 2010.
- [27] K. J. Letsholo, L. Zhao, y E.-V. Chioasca, “Tram: A tool for transforming textual requirements into analysis models,” *Conference Paper*, 2013.

- [28] S. K. Rath y A. Tripathy, "Application of natural language processing in object oriented software development," *International Conference on Recent Trends in Information Technology*, 2014.
- [29] A. Tripathy, A. Agrawal, y S. K. Rath, "Requirement analysis using natural language processing," *Conference Paper*, 2014.
- [30] R. Sharma, P. K. Srivastava, y K. K. Biswas, "From natural language requirements to uml class diagrams," *AIRE 2015, Ottawa, ON, Canada*, 2015.
- [31] C. Narawita y K. Vidanage, "Uml generator – use case and class diagram generation from text requirements," *International Journal on Advances in ICT for Emerging Regions (ICTer)*, 2017.
- [32] J. Schamltz, "Reader software specification - capítulo 3 üse case diagrams", 2019.
- [33] D. K. Deeptimahanti y M. A. Babar, "An automated tool for generating uml models from natural language requirement," *IEEE/ACM International Conference on Automated Software Engineering*, 2009.
- [34] M. Z. Alksasbeh, B. A. Y. Alqaralleh, T. A. Alramadin, y K. A. Alemerien, "An automated use case diagrams generator from natural language requirements," *Journal of Theoretical and Applied Information Technology*, 2017.
- [35] S. Vemuri, S. Chala, y M. Fathi, "Automated use case diagram generation from textual user requirement documents," *IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2017.
- [36] M. S. Osman, N. Z. Alabwaini, T. B. Jaber, y T. Alrawashdeh, "Generate use case from the requirements written in a natural language using machine learning," *IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, 2019.
- [37] I.-Y. Song, "Entity-relationship modeling," *IEEE Potentials*, 1995.
- [38] E. S. Btoush y M. M. Hammad, "Generating er diagrams from requirement specifications based on natural language processing," *International Journal of Database Theory and Application*, 2015.
- [39] R. Dedhia, A. Jain, y P. K. Deulkar, "Techniques to automatically generate entity relationship diagram," *International Journal of Innovations & Advancement in Computer Science*, 2015.

- [40] P. G. T. H. Kashmira y S. Sumathipala, “Generating entity relationship diagram from requirement specification based on nlp,” *3rd International Conference on Information Technology Research (ICITR)*, 2018.
- [41] S. Ghosh, P. Mukherjee, y B. Chakraborty, “Automated generation of e-r diagram from a giventext in natural language,” *International Conference on Machine Learning and Data Engineering (iCMLDE)*, 2018.
- [42] A. Dwarakanath, R. R. Ramnani, y S. Sengupta, “Automatic extraction of glossary terms from natural language requirements,” *21st IEEE International Requirements Engineering Conference (RE)*, 2013.
- [43] C. Arora, M. Sabetzadeh, L. Briand, y F. Zimmer, “Automated extraction and clustering of requirements glossary terms,” *IEEE Transactions on Software Engineering*, 2017.
- [44] T. Gemkow, M. Conzelmann, K. Hartig, y A. Vogelsang, “Automatic glossary term extraction from large-scale requirements specifications,” *IEEE 26th International Requirements Engineering Conference*, 2018.
- [45] F. N. A. A. Omran y C. Treude, “Choosing an nlp library for analyzing software documentation: A systematic literature review and a series of experiments,” 2017.
- [46] Django, *Django Software Foundation Documentation*, 2019.
- [47] ADIMEN, “Multilingual central repository web, <http://adimen.si.ehu.es/web/mcr>,” 2016.
- [48] WordNet, “Wordnet web, <https://wordnet.princeton.edu/>,” 2016.
- [49] MultiWordNet, “Multiwordnet web, <http://multiwordnet.fbk.eu/english/home.php>,” 2016.
- [50] Google, “Google translate, <https://cloud.google.com/translate/docs/?hl=es>,” 2020.
- [51] Spacy, “Spacy documentation, <https://spacy.io/usage/linguistic-features>,” 2020.
- [52] L. A. H. Cuadrado, “El orden de palabras en español,” *REVISTA DE FILOLOGÍA*, 2005.
- [53] S. Padmaja y P. S. S. Fatima, “Opinion mining and sentiment analysis –an assessment of peoples’ belief: A survey,” 2013.

- [54] D. Berrar, “Bayes’ theorem and naive bayes classifier,” 2018.
- [55] C. U. Press y G. 1997, *Survey of the State of the Art in Human Language Technology*, 1997.
- [56] K. Li, R.G.Dewar, y R.J.Pooley, “Object-oriented analysis using natural language processing,” 2013.
- [57] P. G. T. H. Kashmira y S. Sumathipala, “Generating entity relationship diagram from requirement specification based on nl,” 2018.
- [58] C. D. Manning, M. Surdeanu, J. Bauer, y J. Finkel, “The stanford corenlp natural language processing toolkit,” 2014.

Apéndices

Apéndice A

Manual de uso

A.1 Instalación

A.1.1. Docker

Docker es una plataforma abierta para desarrollar, enviar y ejecutar aplicaciones que te permite empaquetar tu proyecto con todas sus dependencias únicamente necesarias en un simple binario, de forma totalmente aislada al resto de aplicaciones que puedan convivir en el mismo host.

A.1.2. Docker Compose

Es una herramienta para definir y ejecutar aplicaciones Docker multicontenedor que permite simplificar el uso de Docker a partir de archivos YAML, de esta forma es más sencillo crear contenedores que se relacionen entre sí, conectarlos, habilitar puertos, volúmenes, etc. Nos permite lanzar un solo comando para crear e iniciar todos los servicios desde su configuración (YAML), esto significa que puedes crear diferentes contenedores y al mismo tiempo diferentes servicios en cada contenedor, integrarlos a un volumen común.

A.1.3. Instalación de imagen

Para facilitar la instalación de dependencias que tiene este trabajo, se realizó una imagen de docker asociada con docker-compose para permitir que cualquier investigador pueda levantar un entorno sin preocuparse por la plataforma o las tecnologías que se utilizan. Esta imagen de docker permite instalar los servicios que la plataforma necesita, tanto el servidor web Django como la base de datos MySQL, a su vez crea las bases de datos con opiniones pre-cargadas para el módulo de Machine Learning

y el glosario de Repositorio Central Multilinguaje. Por lo tanto, para la persona que necesita utilizar esta herramienta solo es necesario que ejecute el comando «*docker-compose up*», esperar el tiempo necesario para que se instale la plataforma y ya podrá utilizarla sin problemas. El archivo `docker-compose.yml` que permite la instalación se conforma de la siguiente manera:

```
1
2
3 # Archivo docker-compose.yml
4 version: '3'
5 services:
6   db:
7     container_name: db
8     image: mysql:8
9     command: --default-authentication-plugin=mysql_native_password
10    ports:
11      - "3306:3306"
12    environment:
13      - MYSQL_USER=tesislel
14      - MYSQL_PASSWORD=password
15      - MYSQL_ROOT_PASSWORD=password
16    volumes:
17      - ./init:/docker-entrypoint-initdb.d
18  web:
19    container_name: django_web
20    build: .
21    command: bash -c "python manage.py runserver 0.0.0.0:8000"
22    ports:
23      - "8000:8000"
24    volumes:
25      - ./projectesis:/app
26    environment:
27      - MYSQL_USER=tesislel
28      - MYSQL_PASSWORD=password
29      - MYSQL_ROOT_PASSWORD=password
30      - MYSQL_HOST=db
31    depends_on:
32      - db
```

A.2 Uso de la herramienta

Una vez realizada la instalación, el usuario puede proceder a utilizar la herramienta que por defecto se ejecuta en el puerto 8000 del sistema. Para acceder, se debe abrir un navegador e ingresar a *http://localhost:8000*, donde se podrá ver la vista en la figura A.1 donde se dispone de una input HTML para que el usuario ingrese un texto a procesar.

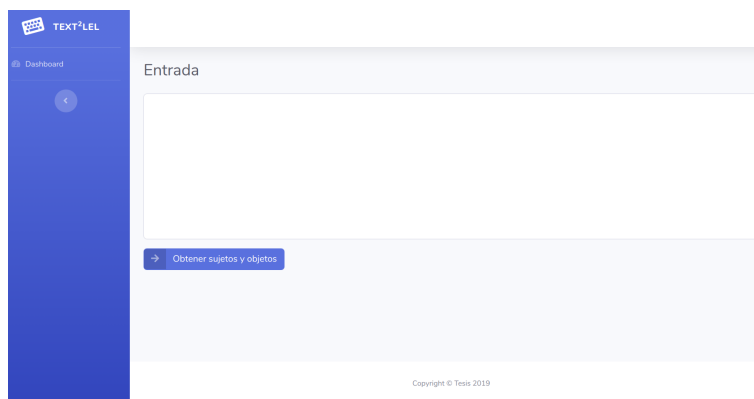


Figura A.1: Entrada de información del sistema

Luego de que el usuario confirme el texto ingresado clickeando en el botón «Obtener sujetos y objetos», el sistema procederá a mostrar una vista A.2 donde la herramienta es capaz de obtener los símbolos de «sujetos» y «objetos» recuperados en el texto. Adicionalmente dispone un wordcloud de palabras, donde la herramienta expresa las palabras más importantes encontrados en el text. Así mismo, el sistema permite modificar los símbolos hallados, agregando nuevos o eliminando falsos positivos A.3, esto permite que para el próximo uso de la herramienta la subjetividad del usuario se vea influenciado en el proceso. Una vez que el usuario está seguro de estos símbolos debe confirmar realizando un click en «obtener relaciones» para obtener los impactos asociados a estos símbolos. Llegando al final del uso de la herramienta, la misma dispone todos los impactos asociados a los elementos encontrados en el paso anterior, donde adicionalmente encuentra otros objetos asociados a los impactos, de esta manera ampliando aún más la cantidad de elementos de este tipo de símbolos. En esta misma vista el usuario, tiene la posibilidad de filtrar aquellos impactos que no le parecen adecuados o que son incorrectos, como se puede apreciar en la imagen A.5, de todas maneras el sistema permite que el usuario se pueda arrepentir o corregir en caso de un error. Luego el usuario puede confirmar estos impactos clickeando en «Guardar opiniones y seguir a la vista final», donde el sistema guardará en la base de datos los impactos que el usuario considero como correctos e incorrectos para mejorar



Figura A.2: Vista de los objetos y sujetos encontrados

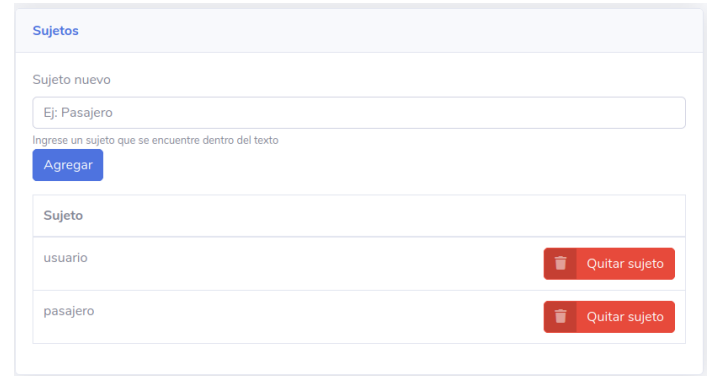


Figura A.3: Acciones sobre los sujetos

la herramienta.

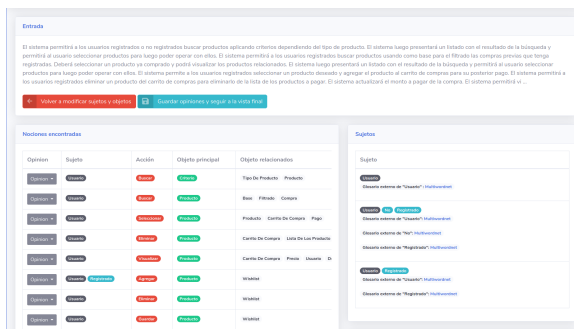


Figura A.4: Vista de los impactos encontrados



Figura A.5: Opción de filtrar un impacto

Para finalizar el proceso, se dispone de dos opciones de descarga de los impactos y símbolos encontrados en formato CSV, de esta manera el usuario puede importarlos en una plantilla de cálculo, o utilizarlo como entrada de otro sistema.

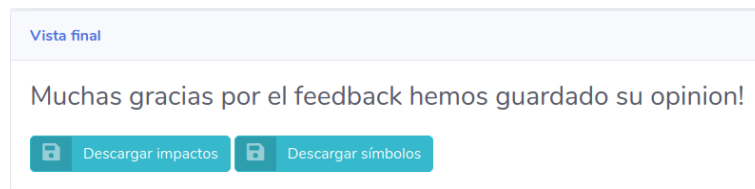


Figura A.6: Vista final y descarga