UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

CLASIFICACIÓN AUTOMÁTICA DE REQUISITOS NO FUNCIONALES UTILIZANDO REDES NEURONALES CONVOLUCIONALES

TESIS

PARA OBTENER EL GRADO DE
MAESTRA EN TECNOLOGÍAS DE CÓMPUTO APLICADO

PRESENTA:
ING. SANDRA ESTEFANÍA MARTÍNEZ GARCÍA

DIRECTOR DE TESIS:
DR. CARLOS ALBERTO FERNÁNDEZ Y FERNÁNDEZ

CO-DIRECTOR DE TESIS:
M.T.C.A. ERIK GERMÁN RAMOS PÉREZ

HUAJUAPAN DE LEÓN, OAXACA, MÉXICO. NOVIEMBRE DE 2020

Dedicatoria

A mi Dios, mi familia, amigos y maestros.

Agradecimientos

En primer lugar, agradezco infinitamente a mi Dios por permitirme culminar una de mis metas, darme la capacidad y la fortaleza de enfrentarme a cada reto que se me ha presentado.Y por regalarme la existencia de mi familia y amigos hasta el día de hoy.

A mi Juanita y a mi Vicky, mis madres hermosas, que sin su amor, paciencia y cariño esto no habría sido posible. Gracias por dejarme volar.

A mis padres Héctor e Ismael, porque siempre están ahí para mi, de la forma en la que necesite. Sé que sin su respaldo, su amor y guía mi camino por la vida sería muy difícil.

A Reyna, Oscar y Polo, que más que tíos han sido mis amigos y cómplices en diferentes situaciones, me han enseñado con el ejemplo que la superación depende de uno. Gracias por su amor.

A Lalis, Libris y Yul, gracias por demostrarme que no necesitamos tener la misma sangre para tenernos tanto cariño. Sus historias de vida me han inspirado a seguir luchando.

A mis primos hermanos: Fer, Duni, Sheri, Karly, Edgar, Maye y Kari; gracias por motivarme y ser la pila que necesito para lograr mis objetivos. Gracias por esos momentos inolvidables. Gracias por crecer a mi lado.

A mi grillito y mi morenita por su amor de hermanas, por ser grandes ejemplos de vida, por so sombeo y todas las aventuras que hemos vivido juntas, gracias por entender mi ausencia y guardar ese amor para cuando nos reunimos. Gracias.

A mi Bachito gracias por aceptar mi ausencia y recibirme siempre con tanto amor, eres mi alegría entera. A mi Ringo por ser ese alivio durante mis días de estrés, con tus ocurrencias todo se hacía nada.

A mis amigos por entender mi ausencia en algunos momentos invaluables. A mis nuevos amigos que me cobijaron, me brindaron su amor y paciencia durante mi estancia en la Maestría.

Al Mto. Erik Ramos, Mto. Moises Ramirez, la Dra. Carla Pacheco y el Dr. Manuel Hernandez que siempre tuvieron paciencia y tiempo para enseñarme, explicarme y ayudarme. A mi director de tesis y mis sinodales por guiarme.

A los profesores de mi alma mater que me apoyaron y estuvieron al pendiente de mis últimos trámites burocráticos, mil gracias por no dejarme sola.

A Razi por ser mi compañero durante esta aventura.

A todas esas personas que se vieron involucradas directa o indirectamente, gracias.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Durante las fases iniciales del ciclo de vida del desarrollo del software, independientemente del modelo que se pretenda seguir, la fase de requisitos se declara como una pieza clave para conseguir un desarrollo exitoso [2, 5, 91]. Si los requisitos no se descubren y se definen correctamente en esta fase temprana surgen fallas durante el desarrollo, lo cual promueve que la entrega final sea la de un software incompleto, es decir que éste no haga lo que debía hacer, sumando a eso que los tiempos establecidos no se lleguen a cumplir y se extiendan, lo cual ocasionará que los costos previamente estimados se eleven [5, 75]. Por ello, dicha fase se considera vital ya que la correcta ejecución de las actividades evitará que se produzca el fracaso del desarrollo del software [5, 10, 83, 88].

La importancia de la fase de requisitos introduce a la Ingeniería de Requisitos (RE, por sus siglas en inglés) [8, 55, 74]. De acuerdo con [2, 15, 55, 74, 91, 96], la RE está formada de actividades como son elicitación, análisis, especificación, validación/verificación y gestión de requisitos, por lo que el objetivo de la RE es que cada una de estas actividades busque producir requisitos de calidad que vayan directamente ligados a los objetivos y metas de la empresa, es decir, que sólo contengan información necesaria que permita alcanzar el éxito del desarrollo del software. Los requisitos para un software son las descripciones de lo que debe hacer el sistema, los servicios que proporcionará y las restricciones de su funcionamiento [49, 96]. Estos requisitos reflejan las necesidades de los clientes para un sistema que cumple un determinado propósito; de ahí que se busca identificar la funcionalidad y las características no funcionales del software [91, 96].

Las actividades que forman la RE tienen una constante relación con las partes interesadas o *stakeholders*, por lo que la presencia del Lenguaje Natural (NL, por sus siglas en inglés) se encuentra presente durante casi toda la fase [15, 48, 91]. Dada la existencia innata del lenguaje durante las actividades de la RE se producen problemas constantes [28, 85], por lo que se ha buscado implementar técnicas y métodos que permitan el tratamiento de dichos problemas. Por citar algunos ejemplos para la selección de técnicas de elicitación en [29] proponen un enfoque de tres componentes: primero, se realiza una revisión de la

literatura para identificar los atributos que afectan la selección de las técnicas comunes de elicitación; segundo, se construye un modelo de regresión múltiple para analizar estos atributos con el fin de identificar los atributos críticos que influyen en la selección de las técnicas; y tercero, se establece un modelo basado en una Red Neuronal Artificial (ANN, por sus siglas en inglés) para seleccionar las técnicas de elicitación adecuadas para un proyecto dado, el propósito del modelo ANN es ayudar a reducir la participación humana en este proceso.

Además, en [76] se propuso el algoritmo TTC (*Two-Tired Clustering*) que indexa y agrupa las especificaciones de requisitos; primero, identifica los verbos y las palabras clave de cada requisito, luego agrupa estos requisitos por funcionalidad, y a continuación cada grupo se subdivide utilizando la similitud de coseno entre grupos. Por otro lado, en [93] se propone un enfoque de transformación sistemática que extrae automáticamente diversos elementos de casos de uso de especificaciones de problemas textuales, su enfoque utiliza el analizador de NL para identificar etiquetas de partes del habla (POS, por sus siglas en inglés), dependencias de tipo (TD, por sus siglas en inglés) y roles semánticos de la especificación de texto de entrada para completar los elementos de casos de uso.

Considerando lo anterior, se puede observar que la RE realiza actividades con constante interacción humana por lo que el lenguaje natural llega a ser un problema común durante la ejecución de éstas. Algunos de los ejemplos mencionados en los párrafos anteriores tienen como propósito automatizar y evitar la intervención humana, por ello aplicaron técnicas y métodos de aprendizaje automático con el fin de minimizar dichos problemas.

En consecuencia con lo anterior, se encontró la motivación del tema de tesis para utilizar técnicas de aprendizaje automático y aplicarlas en la RE en la actividad de Especificación de Requisitos (RS, por sus siglas en inglés), en particular sobre la clasificación de requisitos. Dicha actividad consiste en identificar la clase de requisito a la que pertenece o simplemente diferenciar entre un requisito e información [48,49]. En particular, se enfocará a los Requisitos No Funcionales (NFR, por sus siglas en inglés), ya que frecuentemente son discriminados porque los consideran de poca o nula importancia para el desarrollo de software, así como la falta de conocimiento para identificarlos [18, 22, 40, 54].

La actividad de clasificación que se llevará acabo durante el desarrollo de esta tesis consiste en una etiquetación automática de NFR; el enfoque a utilizar serán las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés). Éstas se eligieron debido a las características de clasificación que poseen, por ejemplo, realizan el procesamiento de datos de gran dimensionalidad y, gracias a las múltiples neuronas que pueden ser implementadas, brindan mayor confianza con respecto a la precisión de las predicciones. En la sección de Justificación y Marco teórico, de esta tesis, se detalla más sobre este tema.

1.1. Planteamiento del problema

Durante la fase de Especificación de Requisitos se encuentra la actividad de anotar los requisitos de las partes interesadas en un documento, los cuales deben ser completos, consistentes y deben carecer de ambigüedad [48,49,91]. Este documento debe describir los requisitos de los *stakeholders* en lenguaje natural, de ahí que no se debe usar la jerga de software, las notaciones estructuradas o las notaciones formales. La importancia de este documento es que puede usarse como parte del contrato del desarrollo del proyecto [15,49,91].

Para llevar a cabo la RE es necesario tomar en cuenta cual es la utilidad final del sistema, para ello es necesario determinarlo en dos clases de requisitos: por funcionalidad y por características no funcionales [22]. La primera clase hace referencia a los Requisitos Funcionales (FR, por sus siglas en inglés), que se definen en [91] como: *"enunciados acerca de servicios que el sistema debe proveer, de cómo debería reaccionar el sistema a entradas particulares y de cómo debería comportarse el sistema en situaciones específicas"*, y la segunda clase que son los Requisitos No Funcionales representan limitaciones propias sobre servicios o funciones del sistema e incluyen restricciones que actúan como agentes fundamentales durante la búsqueda de la solución de alguna característica del sistema [40,89,91].

La actividad de anotar los requisitos conlleva a realizar una tarea difícil de clasificación mediante la identificación de las clases de requisitos [48]. Debido a la inherente ambigüedad que existe en el lenguaje natural, dicha clasificación está propensa a errores; además, consume tiempo, es costosa porque se necesitan anotadores expertos para tener resultados correctos y podría ser abrumador si son pocos anotadores, ya que no tendrían más puntos de vista con respecto a la clase de requisito en cuestión [48, 60].

Un ejemplo de dicho problema es la investigación de [82], ya que en su propuesta de clasificación de NFR, realizaron un *corpus*. Éste fue elaborado por cuatro anotadores, quienes realizaron un proceso de anotación manual analizando 3, 064 oraciones, obtenidas de distintos documentos de especificación de requisitos de software para posteriormente realizar el etiquetado de los requisitos. Los autores mencionan que cada anotador etiquetaba las 3, 064 oraciones y posteriormente se reunían los cuatro anotadores para determinar si cada una de las oraciones pertenecía, efectivamente, a la clase de requisitos que individualmente identificaron. Este es un claro ejemplo de que el esfuerzo y el tiempo que envuelve el realizar esta actividad representa un problema para los analistas, además de que si no se identifica bien a qué clase de requisitos pertenece cada oración podrían presentarse fallas en etapas posteriores [44, 65]. Por ejemplo, en [44] se menciona que muchas de las fallas durante las pruebas del proyecto se producen porque en el documento de especificación se incluyen requisitos incorrectos, requisitos modificados y requisitos faltantes. También existen informes de estadísticas [41, 51] que muestran que al menos el 70 % de los fracasos de desarrollo de software provienen de requisitos y especificaciones incompletas, cambios

frecuentes en los requisitos y especificaciones, expectativas no realistas, y a objetivos poco claros. Aunque el peso de la especificación de requisitos no solo radica en la actividad de la clasificación, llega a ser una pieza importante para obtener una especificación de calidad [49].

Una prueba más de que la especificación de requisitos no se realiza de forma adecuada es que frecuentemente se pasan por alto o existe falta de atención hacia los requisitos no funcionales. Esto puede deberse a que muchos analistas tienen un vago entendimiento de qué son realmente este tipo de requisitos y por lo tanto consideran que éstos no son importantes durante el desarrollo del proyecto [18, 24, 40, 47, 54]. Sin embargo, ante la discrepancia que existe entre los expertos del tema, [25], [32] y [57] recalcan que los NFR son tan importantes como los FR, ya que describen restricciones importantes sobre el desarrollo y el comportamiento un sistema de software. Por ello, dichos expertos mencionan que los NFR son parte fundamental del diseño arquitectónico y por lo tanto deben considerarse y especificarse de manera oportuna. De no ser identificados en esta etapa temprana los costos de producción del proyecto podrían elevarse, la fecha estimada de entrega sería aplazada y la inconformidad del cliente sería notoria [52, 96].

La existencia de estos problemas e inconvenientes antes mencionados refleja el rol tan significativo que representa el lenguaje natural debido a que el contenido de las especificaciones son descripciones textuales de las necesidades concretas para realizar el desarrollo de un proyecto de software. Por ello se han realizado varios intentos para automatizar el proceso de identificación y clasificación de requisitos, funcionales y no funcionales, de documentos de especificaciones utilizando distintas técnicas de aprendizaje automático, entre ellas las CNN, mediante el procesamiento del lenguaje natural [18, 53, 62, 68, 82, 90] utilizando diferentes conjuntos de datos, entre ellos, Promise [87]. Los intentos mencionados anteriormente y los que se encuentran plasmados en el estado del arte, muestran que el rendimiento de clasificación que han obtenido no ha rebasado el 81 %, por lo que se podrían implementar otros métodos de aprendizaje automático que ayudarían a mejorar el rendimiento de clasificación de los NFR, lo cual le facilitaría a los analistas esta actividad de identificación y clasificación de requisitos.

1.2. Justificación

La RE busca mejorar la calidad en el de desarrollo del software, reducir el riesgo de exceder el presupuesto, retrasos y fallos en el proyecto, por lo que representa una base fundamental para el desarrollo de software, [4, 39, 56, 69]. Dicha mejora se realiza mediante actividades como la elicitación, análisis, especificación, validación/verificación y gestión de requisitos [2, 15, 55, 74, 91, 96], las cuales, si son ejecutadas de forma correcta, promoverán al éxito del desarrollo [4, 10].

Durante la realización de dichas actividades se presenta un problema recurrente: el lenguaje natural, por ello la comunidad de investigadores de la RE han incrementado su interés por atacar dichos problemas con la automatización mediante la Inteligencia Artificial (AI, por sus siglas en inglés) [28]. Algunos ejemplos de la aplicación de AI en actividades de la RE son los siguientes: para la selección de técnicas de elicitación de acuerdo a un proyecto dado se estableció un modelo basado en una ANN, su propósito fue reducir la participación humana [29]; para el análisis de requisitos [31] realizó un generador de diagramas UML a partir de requisitos en lenguaje natural, esto fue mediante el procesamiento del lenguaje natural; y en [76] se propuso un algoritmo TTC para la identificación y agrupación de los tipos de especificaciones de requisitos. Los ejemplos mostrados anteriormente proponen el procesamiento del lenguaje natural en base a las diversas técnicas que ofrece la AI para solucionar ciertas actividades de la RE. Por ello, para la realización de está tesis se propone atacar el problema de clasificación de requisitos, que es una de las actividades la Especificación de Requisitos, la cual busca identificar las clases de requisitos que existen en un documento de requisitos con texto en NL. Los documentos de requisitos concentran descripciones textuales que regularmente son conjuntos de datos con grandes cantidades de información [50, 91, 97].

Actualmente se han hecho diferentes propuestas para realizar dicha actividad entre ellas se encuentra la de [18], donde evaluaron un enfoque semi supervisado para la clasificación de los NFR mediante la Maximización de Expectativa con Clasificadores Bayesianos *naïve*. Esto fue para evaluar la efectividad de los clasificadores y calcular la precisión de categorizar la recopilación de requisitos utilizando diferentes tamaños del conjunto de entrenamiento Promise. Posteriormente, se elaboró una lista de requisitos sobre clases reales, donde un simulador analista hacia comentarios de clasificación sobre dicha lista y a continuación un analista humano se encargaba de expresar su acuerdo o desacuerdo con la elección del simulador. El intercambio de información sobre las clases asignadas en las listas clasificadas entre el analista y el simulador permitía converger la clasificación real dado un número de iteraciones, de ahí que obtuvieron una precisión en la clasificación dentro de los primeros 5 elementos de 90.11 % y considerando 10 elementos se obtuvieron 93.66 %.

Por otro lado, en [82] se utilizaron 6 tipos de documentos de requisitos para elaborar el conjunto llamado Concordia, donde obtuvieron 3, 140 oraciones sobre las cuales 6 analistas realizaron el trabajo de la clasificación de forma manual. Además, en ese trabajo implementó un clasificador basado en el modelo de una Máquina de Soporte Vectorial (SVM, por sus siglas en inglés) donde obtuvieron una medida de clasificación del 84.96 %.

En [68] se realizó la identificación de NFR mediante la similitud semántica de palabras con el análisis semántico latente (LSA, por sus siglas en inglés) y la agrupación de palabras requeridas (*required word clustering*). Para determinar el rendimiento de la clasificación se

realizó un análisis con tres métodos: Similitud semántica de texto (TSS, por sus siglas en inglés), Modelo de espacio vectorial (VSM, por sus siglas en inglés) e Indización semántica latente (LSI, por sus siglas en inglés). Estos métodos realizan mediciones en términos de precisión (*precision*) y recuperación (*recall*). Los investigadores pudieron notar que el mejor rendimiento fue obtenido por VSM, la principal desventaja entre TSS y LSI residió en su proceso de calibración, el cuál fue computacionalmente costoso. Los tres métodos lograron una medida en umbral del 70 %, esta baja precisión se debe a las impurezas de los conjuntos de datos que manejaron, en particular por el proceso de agrupamiento, ya que algunos clústeres de NFR contenían ruido, o palabras no relacionadas. Otro aspecto que llevó a la baja precisión es que los conjuntos de datos no eran robustos. El conjunto de datos que utilizaron fue SmartTrip, BlueWallet y SafeDrink.

En [89] se propuso un enfoque basado en reglas para la clasificación automatizada de NFRs de Especificaciones de Requisitos. Durante el análisis de requisitos utilizaron la herramienta GATE, donde cada oración fue anotada por roles temáticos usando reglas JAPE (complemento de GATE). Los investigadores reportaron la comparativa entre los conjuntos de datos de Promise con un 97.21 % y Concordia obtuvo un 94 % de medida, cabe mencionar que dicha medida fue en base a subclases de NFRs.

En [53] se planteó la implementación de una CNN donde evaluaron la clasificación de requisitos funcionales y no funcionales. El procesamiento de los datos se realizó con el modelo *Skip-gram*, el cual predice las palabras de contexto, posteriormente el resultado se implementó en la CNN para la clasificación y por último para medir qué tan precisa fue evaluada la clasificación, se implementó un entrenamiento con la validación cruzada *K-Fold*, con $k = 10$; una precisión del 81 %. El conjunto de datos que utilizaron fue Promise.

Las propuestas mencionadas en los párrafos anteriores, aunque son solo unos cuantos ejemplos, dejan ver las técnicas y modelos de aprendizaje automático que se han aplicado para atacar la actividad de clasificación. En base a ello y a la revisión que se hizo en el estado del arte, se plantea implementar una CNN. Aunque este modelo ha sido utilizado para detección de objetos, visión por computadora, procesamiento de imágenes y voz, recientemente ha generado nuevas ideas con respecto al procesamiento del lenguaje natural en texto [12, 63, 71–73]. La razón más importante para elegir dicho modelo es que ofrece una mayor precisión y mejora el rendimiento del sistema debido a características únicas como pesos compartidos e interconectividad local, además que pueden procesar datos de gran dimensionalidad y la implementación de múltiples neuronas dentro de esta red, lo cual puede brindar confianza en las predicciones de clasificación [12].

La entrada para el entrenamiento de la CNN será el conjunto de datos Promise que contiene 625 oraciones de requisitos preclasificadas, de las cuales 255 son FR y 370 son NFR. Sólo se abordarán los requisitos no funcionales, como se mencionó en el planteamien-

to del problema, ya que frecuentemente no son tomados en cuenta debido a que existen problemas para identificarlos de forma adecuada durante la actividad de clasificación, e inclusive se piensa que no son relevantes para el desarrollo del software [9,34,94]. La selección del conjunto de datos Promise, se debió a su frecuente uso en investigaciones previas, lo cual permitirá ofrecer una comparativa de rendimiento.

1.3. Hipótesis

Si se aplican redes neuronales convolucionales para el procesamiento de lenguaje natural se mejorará el rendimiento en la clasificación automatizada de NFRs a partir de texto en el idioma natural inglés.

1.4. Objetivos

1.4.1. Objetivo general

Clasificar requisitos no funcionales mediante el procesamiento de lenguaje natural a partir de texto en el idioma inglés utilizando redes neuronales convolucionales para mejorar el rendimiento de la clasificación.

1.4.2. Objetivos particulares

1. Seleccionar la arquitectura profunda más adecuada para el procesamiento del lenguaje natural.
2. Implementar la arquitectura seleccionada con las herramientas Tensorflow y Keras.
3. Comparar los resultados obtenidos con la arquitecturas profundas y el trabajo relacionado.
4. Proponer una guía de software para la clasificación de NFRs usando la arquitectura profunda implementada.

1.5. Estructura de la tesis

En el Capítulo 2 se presenta el marco teórico y estado del arte del tema de tesis. Para el marco teórico se comienza resaltando la influencia del NL en la RE, para dirigir la atención, posteriormente, a las estrategias que han usado los expertos en el área para contrarrestar

dicha influencia. Después, se identifica en qué actividad de la RE se encuentra la clasificación de requisitos; además de mencionar qué clase requisitos se usarán en el desarrollo de la propuesta de la tesis, así como el conjunto de datos a implementar. A continuación, se establece la unión entre el NLP y el aprendizaje automático, por ello primero se definen términos teóricos propios de ambos conceptos, todo con el fin de que en la sección que concluye dicho capitulo se exponga el trabajo relacionado con la implementación de dicha unión para clasificar requisitos[1]. En el estado del arte se hace una distinción entre los trabajos relacionados que implementaron CNNs de los que implementaron otras técnicas del aprendizaje automático, así como distintos métodos de vectorización.

En el Capítulo 3, se describe la metodología para realizar la clasificación de NFR utilizando CNN's, detallando cómo se realiza el tratamiento de los datos para posteriormente implementarlos sobre una CNN, al final de dicho capítulo se muestran los resultados de forma general.

En el Capitulo 4, se realiza la experimentación que aborda el tema de tesis. Se analizan los resultados obtenidos con los clasificadores Naïve Bayes, Máquina de soporte Vectorial (SVM, por sus siglas en inglés) y la implementación de la CNN propuesta, junto a estrategias que permitan mejorar los resultados comparados con el estado del arte [37, 53].

El Capítulo 5 revela las conclusiones del trabajo de tesis relacionadas con el proceso de investigación y el punto de vista personal. Finalmente, en este capítulo se incluye una propuesta de trabajo a futuro.

Por último en la sección de Anexos, se presenta información adicional de relevancia que se haya obtenido durante la investigación. Por ejemplo, se muestra un concentrado de resultados bajos para el modelo de CNN base como referencia de la mejora del modelo propuesto. Además, incluye una guía para el uso de la aplicación web que auxilia al analista a clasificar los requisitos no funcionales.

[1]La recopilación de ese trabajo relacionado se encuentra fuera del rango temporal establecido para el estado del arte, por ello se incluye en una sub sección del marco teórico.

Capítulo 2

Antecedentes y Fundamentos

2.1. Marco teórico

Para comprender el contexto en el que se encuentra el tema de tesis y el desarrollo del mismo, primero se establece el fundamento teórico necesario para poder comprender tecnicismos de AI y RE. El punto de partida se inicia señalando el impacto que tiene el NL en la RE y cómo los especialistas en el área han optado por utilizar NLP para automatizar algunas actividades de RE. Posteriormente, se hace una breve descripción de las actividades de la RE para identificar en qué actividad interviene la clasificación de requisitos y en qué consiste, también se dirige la atención hacia los NFR y al conjunto de datos que se usará en la experimentación de la tesis. Por último, se muestran conceptos teóricos básicos: aprendizaje automático y sus tipos, el aprendizaje de representaciones, también se habla de algunos métodos de vectorización y arquitecturas profundas, con el fin de presentar el trabajo de algunos investigadores que han atacado problemas de clasificación de requisitos, demostrando que la unión del NLP y del aprendizaje automático es un buen aliado para ese tipo de problemas.

2.1.1. El lenguaje natural en la Ingeniería de Requisitos

La RE es un enfoque disciplinado y sistemático para elicitar, analizar, especificar, validar/verificar y gestionar los requisitos [7, 38, 91]. Conforme se va avanzando durante la realización de cada una de las actividades de la RE, la actividad anterior resulta tan importante como la que precede, la relevancia radica en que la RE es el pilar para el desarrollo del software [92]. Por lo que si en la RE, alguna actividad o tarea dentro de las actividades se llegaran a omitir, desencadenaría una avalancha de problemas posteriores durante el desarrollo del proyecto [20, 89, 97]. Como se mencionó anteriormente la RE es fundamental para el desarrollo del software, sin embargo, su correcta realización depende del factor humano y el NL, por lo que dichas actividades resultan con complicaciones e inconvenientes [36, 68, 89].

Pero, ¿por qué el factor humano y el NL complican la RE? Para entenderlo imagine la siguiente situación: un analista[1], consciente de que omitió actividades y tareas de la RE porque el creía que eran innecesarias, consideró que su trabajo en la etapa de requisitos había concluido, por lo que, según él, la especificación de requisitos estaba lista. Por ello, decidió entregar la especificación al encargado de diseño, el cual al ver que tenía mucho por leer, optó por archivar la especificación y abordar al analista durante el almuerzo para que en unas cuantas palabras le dijera qué es lo que el software debería hacer. Con eso en mente, el encargado de diseño efectúa las actividades propias de su etapa. Posteriormente, el resultado de dicha etapa se envía a codificación, y los programadores comienzan a codificar de acuerdo con las peticiones del área de diseño. Una vez que ésta concluyó, se procede a enviar el resultado de la etapa al área de prueba. Sin embargo, durante el testeo se dan cuenta que el software carga cualquier tipo de archivos, a lo que los *stakeholders* muestran inconformidad ya que alegan que una de las funciones principales del software era cargar sólo archivos en formato .PDF, con un rango de peso definido y con una longitud específica del nombre del archivo. Después de hablar con los *stakeholders* y mencionarles que ese error será corregido, el jefe del proyecto va con el área de codificación reclamando que cometieron errores a la hora de establecer restricciones y funcionalidades del software, a lo que estos mencionan que su entrega fue lo que el área de diseño pidió. Así es como el jefe de proyecto dialoga con el encargado de diseño y se da cuenta que el desarrollo fue en base a lo que él pensó que el software debería hacer, así que pide que realice su trabajo conforme a lo que dice la especificación. Y una vez más se hecha a andar el desarrollo del software, no obstante, cuando llegan al área de test se encuentran nuevamente con un error, por lo que buscan el requisito dentro de la especificación y se dan cuenta que la interpretación de éste fue errónea.

Aunque parezca un cuento de nunca acabar, las situaciones del ejemplo anterior son comunes dentro del desarrollo de los proyectos [41, 51]. Las situaciones que se observaron en el párrafo anterior dejan ver cómo la negligencia del personal, la falta de información y su interpretación errónea de información desencadenan inconvenientes. Algunos inconvenientes son: entregas fuera de tiempo, costos elevados con respecto a lo que se estableció en un inicio y clientes inconformes con la entrega [18, 53, 82, 97].

Para contrarrestar las aptitudes negligentes del factor humano, algunas empresas han optado por sustituir al personal que muestre dichas aptitudes, hacerles llamadas de atención para que mejoren su desempeño o bien utilizar algún software que realice su trabajo [86]. Sin embargo, para la RE, en la actualidad, la consideración de la última opción puede aplicarse parcialmente durante la realización de sus actividades, ya que de acuerdo al estudio que realizaron [38] y [30] se puede observar que la mayoría de software dirigido a la RE involucra sólo cuestiones de gestión en todas sus actividades, es decir, la

[1]Es la persona que se encarga de identificar y definir las necesidades del software a desarrollar en base a la información que proporcionan los *stakeholders*.

modificación y eliminación de información. Al analizar los estudios que se mencionaron anteriormente, se identificó sólo una herramienta que sirve de apoyo para la elicitación de requisitos. Dicha herramienta ofrece al analista una versión automatizada de la técnica de entrevistas, esto es mediante el chat que hace preguntas dependiendo de las respuestas del *stakeholder*. El chat automatizado contiene un conjunto de cuestionarios con tópicos comunes de empresas que permiten la interacción con el *stakeholder*, además permite ingresar cuestionarios propios. Ahora bien, ¿qué estrategia se utiliza para contrarrestar la presencia de las interpretaciones erróneas de la información durante las actividades de la RE, es decir, el lenguaje natural?.

2.1.2. Ingeniería de Requisitos y el Procesamiento de Lenguaje Natural

Cada una de las actividades de la RE, tiene como común denominador la interacción entre el analista y los *stakeholders*, por lo que en consecuencia se presenta el innherente NL [22,38,62,90]. Por ello, se dice que es la notación predominante a la hora de documentar y especificar software y requisitos del sistema [92]. Todos esos requisitos escritos en NL se extraen de la información que proporcionan los *stakeholders*, por lo que el analista debe establecer la distinción entre deseos y necesidades que éstos expresaron [5] [75]. Sin embargo, el NL puede presentar ambigüedades, lo que llevaría a una definición de requisitos incorrecta [17]. Dada esa situación, se comprende que los requisitos deben ser una conceptualización abstracta de lo que realmente se necesita del sistema, debido a ello el proceso de definición de requisitos puede llegar a estar inherentemente abierto a distintas interpretaciones [36].

2.1.2.1. Procesamiento de Lenguaje Natural

Algunos especialistas en el área de RE se han inclinado por utilizar el NLP y utilizar las ventajas que éste ofrece [29, 48, 60, 76, 93]. A continuación se detalla más sobre el NLP y sus aplicaciones.

El NLP es un área de investigación de la AI que se encarga de explorar cómo se pueden usar las computadoras para comprender y manipular texto y habla en el lenguaje natural con el objetivo de aplicarse en distintos campos donde pueda ser de utilidad [21, 36]. El objetivo de los investigadores de esta área es reunir conocimientos sobre cómo los seres humanos entienden y usan el lenguaje para comunicarse y relacionarse, por ello se busca desarrollar herramientas y técnicas apropiadas para que los sistemas informáticos entiendan y manipulen el lenguaje natural y puedan desempeñar diversas tareas que normalmente consumen recursos significativos para dichas tareas [21].

Las disciplinas en las que se ha implementado el uso del NLP son Ingeniería de Software, Informática, Lingüística, AI, Robótica, Psicología, entre otras. Tiene aplicaciones en distintos campos de estudio por ejemplo, por mencionar algunos, en la traducción automática, el procesamiento y resumen de texto en lenguaje natural, las interfaces de usuario, el reconocimiento de voz, y sistemas especialistas [21].

2.1.2.2. Herramientas para NLP

Existen diversas herramientas que son de gran utilidad para realizar tareas asociadas con el campo de aprendizaje automático y las redes neuronales profundas [18, 53, 62, 82]. El objetivo de éstas es construir y entrenar redes neuronales para detectar y descifrar patrones. Para el caso de esta tesis las que se consideran son las siguientes:

- Tensorflow. Se originó del trabajo de *Google Brain Team*, de un grupo de ingenieros e investigadores que formaban parte de la organización de investigación de aprendizaje automático y redes neuronales profundas. TensorFlow es una biblioteca de software libre que se utiliza para realizar cálculos numéricos mediante diagramas de flujo de datos. Los nodos de los diagramas representan operaciones matemáticas y las aristas muestran matrices de datos multidimensionales, llamados tensores, comunicadas entre ellas. Para utilizar esta herramienta se necesita una API para desplegar el sistema informático de una o varias CPU[2] o GPU[3].
- Keras. Keras es una API de redes neuronales de alto nivel, escrita en Python y capaz de ejecutarse sobre TensorFlow, CNTK o Theano. Fue desarrollada para permitir la experimentación rápida. Es decir, pasar de la idea al resultado con el menor retraso posible con el objetivo de una buena investigación.

2.1.3. ¿Qué es la clasificación de requisitos y en qué actividad de la Ingeniería de Requisitos se encuentra?

Como se mencionó en secciones anteriores, la RE cuenta con cinco actividades. Las cuales se muestran gráficamente en la Figura 2.1 y se describen brevemente a continuación. Durante el proceso de elicitación se capturan, sin restricciones y sin ninguna formalización, las necesidades de los *stakeholders* en NL [75, 91]. Posteriormente, el análisis de estas necesidades generalmente implica la manipulación de grandes conjuntos de texto que, al estar escritos en un lenguaje natural, frecuentemente contienen ambigüedades; por ello, es necesaria la intervención de un experto que los interprete sin que pierda de vista las metas y el objetivo de la empresa, para así obtener los requisitos y concentrarlos en un documento [78]. Dicho documento será manipulado posteriormente por un experto para diferenciar

[2]Central Processing Unit. Unidad de procesamiento central
[3]Graphics Processing Unit. Unidad de procesamiento gráfico

entre una clase y otra de requisitos [35], lo cual servirá para determinar las características funcionales y no funcionales; a este proceso se le conoce como especificación de requisitos. Finalmente, se realiza la verificación y validación del resultado de la actividad anterior, que consiste en corroborar que efectivamente cada requisito corresponde a las necesidades de los *stakeholders* [91]. Durante la ejecución de cada una de las actividades descritas anteriormente, se realiza, de forma casi simultánea, la gestión de requisitos, esta actividad se encarga de comprender y controlar los cambios que se vayan realizando durante cada una de las actividades anteriores a ésta [91]. El resultado del proceso sistemático de la RE es la Especificación de Requisitos del Software (SRS, por sus siglas en inglés), el propósito de este documento es definir los requisitos específicos de lo que el software debe de hacer y cómo se espera que funcione, por ello se deben encontrar alineados con su propósito y naturaleza [47, 89, 91]. El nivel de importancia de este documento es elevado ya que establece la base del proyecto, siendo así un contrato o documento oficial [91].

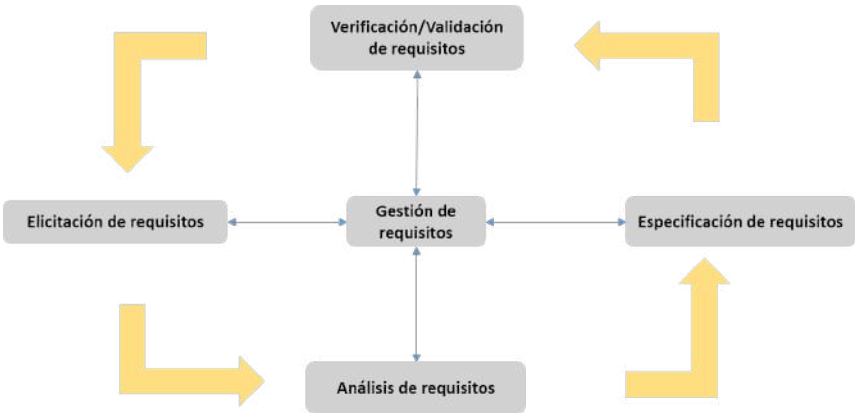


Figura 2.1: Actividades de la RE [75].

Como se ha resaltado, durante todo el documento de tesis, las actividades tienen el mismo nivel de importancia así como de dificultad. El objetivo de ello ha sido el no menospreciar una de la otra, ya que el tema de investigación se centra en una tarea en particular de una de estas actividades: la clasificación de requisitos. Si se dirige la atención una vez más a las actividades descritas anteriormente, la ubicación de la clasificación de requisitos se encuentra dentro de la Especificación de Requisitos (RS, por sus siglas en inglés) [35,91].

La clasificación de requisitos consiste en diferenciar entre los FR y los NFR[4], dicha actividad se realiza mediante el conocimiento que el analista tiene de lo que es un FR y un NFR para determinar a qué clase pertenece cada oración de la SRS y hacer una anotación manual de éstos [60,82]. En el Cuadro 2.1 se muestra un ejemplo de la anotación realizada de un extracto de un documento de requisitos. En los casos en los que se tiene un grupo de

[4]Se sabe que existen otros tipos de requisitos, pero para efectos de la tesis solo se hará mención de los FR y los NFRs.

analistas, se hacen sesiones constantes para comparar la clasificación que determinó cada analista y, en caso de no coincidir se abre el debate para determinar de forma conjunta a qué clase pertenece en realidad [48,82,89]. Al observar ambas situaciones, las dos implican tiempo y esfuerzo por parte de los analistas, sumándole que las SRS no son solo 5 oraciones, ya que dependiendo del tamaño del proyecto será el contenido de la SRS [53, 80, 82, 89]. Ahora bien, si sólo es un analista, podría interpretar de forma errónea debido a que existe la posibilidad de que no sepa diferenciar una clase de la otra e incluso las subcatergorias de NFR lo que implicará, posteriormente, problemas futuros durante las etapas siguientes al desarrollo [47, 61, 90].

Extracto de un documento de requisitos

El siguiente caso de uso describe la aprobación de un presupuesto. Primero, el usuario navega a la página de resumen del presupuesto. El sistema luego muestra el resumen del presupuesto con atributos de presupuesto editables. El sistema preestablece algunos de los atributos del presupuesto. El usuario edita los atributos del presupuesto y establece el estado como "Aprobado". Todos los atributos obligatorios no pueden estar vacíos y los importes del presupuesto no pueden ser negativos. El usuario finalmente guarda el presupuesto.

Requisito por oración	Etiqueta de anotación
El siguiente caso de uso describe la aprobación de un presupuesto.	Ruido
Primero, el usuario navega a la página de resumen del presupuesto.	Funcional
El sistema luego muestra el resumen del presupuesto con atributos de presupuesto editables.	Funcional
El sistema preestablece algunos de los atributos del presupuesto.	Funcional
El usuario edita los atributos del presupuesto y establece el estado como "Aprobado".	Funcional
Todos los atributos obligatorios no pueden estar vacíos y los importes del presupuesto no pueden ser negativos.	No Funcional
El usuario finalmente guarda el presupuesto.	Funcional

Cuadro 2.1: Ejemplo de anotación de requisitos del software (traducido de [48]).

Al igual que en todas las actividades de la RE, el NL se encuentra presente dentro de la RS ya que está contenido en la lista de requisitos ahí descritos. De acuerdo con [67], alrededor del 90 % de las RS se encuentran representados en NL. En dicha lista, comúnmente, se deben encontrar dos clases de requisitos, estos son los FR que describen las funciones que el software ejecutará y los NFR que actúan para restringir la solución, por lo cual se les conoce como restricciones o requisitos de calidad [89]. Sin embargo, la ambigüedad innata al lenguaje natural, así como la interpretación dada por los analistas, da lugar a que la clasificación, de las clases mencionadas anteriormente, resulte difícil de realizar [82]. Por ello los analistas se inclinan por los FR ya que, frecuentemente, no tienen suficiente conocimiento para identificar y clasificar un NFR, además de que consideran que si no los incluyen en la SRS no tendrá algún impacto significativo para el desarrollo del software [34, 40, 82, 90]. No obstante, varios investigadores han resaltado la importancia de los NFR durante y después el desarrollo del software, ya que si se les subestima pueden causar fracasos [5, 9, 23, 40, 80, 90]. Por ello, la clasificación que se propone en la tesis va dirigida a los NFR. A continuación se detalla un poco más de ellos, así como del conjunto de datos Promise que se utilizará en la experimentación.

2.1.3.1. Requisitos No Funcionales

En [40] se realizó una revisión sistemática de los NRFs, donde se les define atributos o restricciones en un sistema. Este investigador definió la taxonomía de requisitos mostrada en la Figura 2.2, donde hace especial referencia a los NRFs. Dicha taxonomía menciona que estos requisitos pueden ser identificados como atributos y restricciones, donde ambos se enfocan en definir las métricas para el desempeño y calidad de los NRFs.

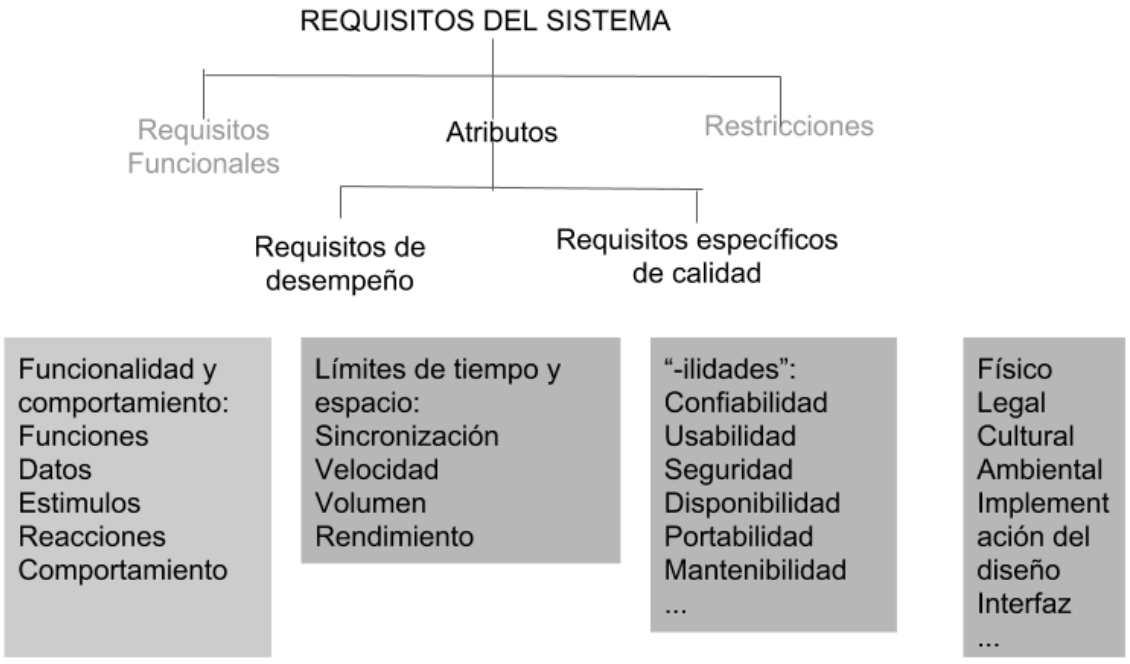


Figura 2.2: Taxonomía de Requisitos. Requisitos No Funcionales (traducido de [40]).

2.1.3.2. Conjunto de datos Promise

El repositorio PRedictOr Models In Software Engineering (Promise) se inició en diciembre de 2004 por Sayyad Shirabad y Tim Menzies para fomentar el desarrollo de modelos predictivos para la Ingeniería de Software [70]. La primera versión de este repositorio se creó a partir de datos de la NASA y se alojó en la Universidad de Ottawa (Canadá). Este repositorio es una colección de conjuntos de datos proporcionados al público de forma gratuita por la comunidad de ingenieros de software para servir a los investigadores y la industria del software.

Promise está contenido por 625 requisitos con 255 FR y 370 NFR. El conjunto de los NFR está etiquetado con las siguientes categorías: Disponibilidad = A, Legal = L, Look and feel = LF, Mantenimiento = MN, Operacional = O, Rendimiento = PE, Escalabilidad

= SC, Seguridad = SE, Usabilidad = US, Tolerancia a fallos = FT y Portabilidad = PO. En el Cuadro 2.2 se muestran ejemplos de cada clase de requisitos que se encuentran almacenados en Promise. En la Figura 2.3 se muestra la distribución de clases que contiene Promise.

Requisito	Clase
The system shall have a MDI form that allows for the viewing of the graph and the data table.	F
The product shall be installed by an untrained realtor without recourse to separately-printed instructions.	US
The system shall have basic data integrity checking to reduce the possibility of incorrect or invalid data being introduced.	SE
The product must work with most database management systems (DBMS) on the market whether the DBMS is colocated with the product on the same machine or is located on a different machine on the computer network.	O
The preferred repair facility ratings shall be saved within 5 seconds. The save shall occur within 5 seconds.	PE
The website shall be attractive to all audiences. The website shall appear to be fun and the colors should be bright and vibrant.	LF
The system shall be capable of processing 100% of nursing students and their classes for the next 10 years.	SC
The system shall be available for use between the hours of 8am and 6pm.	A
Promotional updates to the website should take a day to update.	MN
The System shall meet all applicable accounting standards. The final version of the System must successfully pass independent audit performed by a certified auditor.	L
The product shall operate in offline mode whenever internet connection is unavailable.	FT
The product is expected to run on Windows CE and Palm operating systems.	PO

Cuadro 2.2: Ejemplos del contenido del conjunto de datos Promise [70].

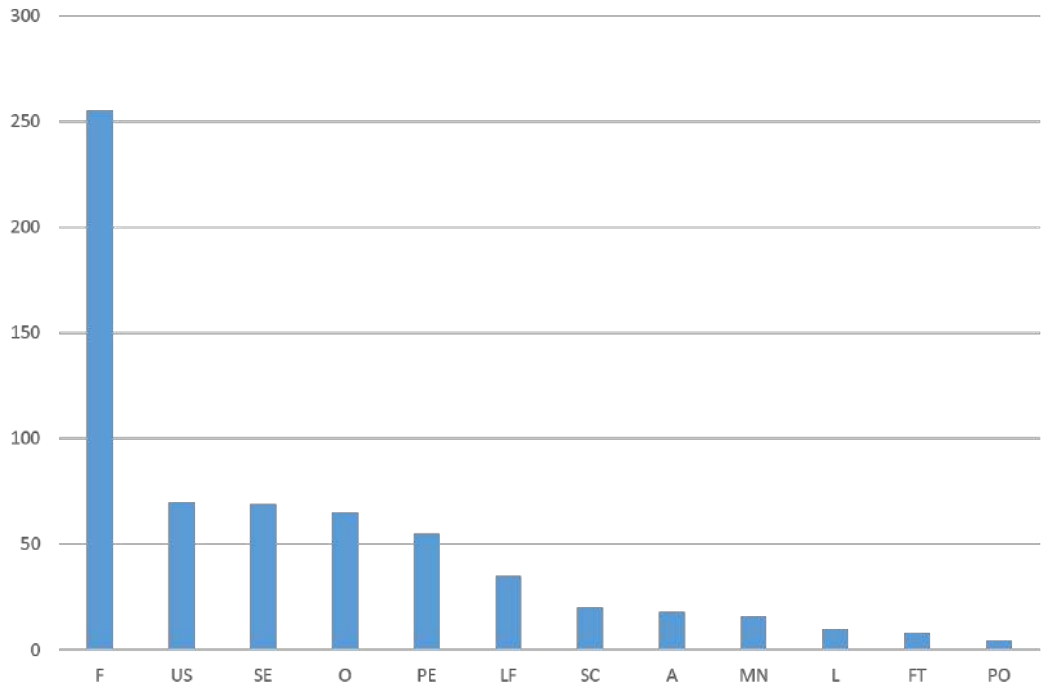


Figura 2.3: Distribución de clases de Promise (traducido de [53]).

2.1.4. Procesamiento del lenguaje natural y aprendizaje automático. La mancuerna ideal para la clasificación de requisitos

En secciones anteriores se analizó como el NL se encuentra presente en la clasificación de los requisitos, debido a la ambigüedad que presenta y la interpretación del analista, éste podría seleccionar como FR o NFR o incluso podría tener confusión entre las subcategorías de NFR [22, 40, 82, 91]. En esta sección se hará un breve acercamiento al aprendizaje automático, así como al aprendizaje de representaciones y a las arquitecturas profundas, que serán de ayuda para comprender términos técnicos. Todo ello servirá de parteaguas para que en la sección siguiente se muestre cómo especialistas en el área de RE han optado por utilizar el NLP junto con técnicas de aprendizaje profundo como ayuda para automatizar dicha actividad.

2.1.4.1. Aprendizaje automático

Es una rama de la AI que busca crear algoritmos capaces de generalizar comportamientos y reconocer patrones usando ejemplos de datos (datos de entrada o casos conocidos) o experiencia acumulada, para luego usar los patrones descubiertos para predecir datos futuros o para realizar otros tipos de toma de decisiones bajo incertidumbre [84]. El aprendizaje automático ha sido de vital importancia para el avance de la tecnología basada en el uso inteligente de la información [11].

2.1.4.2. Tipos de aprendizaje automático

De acuerdo con [81], algunos tipos de aprendizaje automático son:

- **Aprendizaje supervisado**. En este tipo de aprendizaje los datos se encuentran etiquetados (salidas), los cuales consisten en pares (x,y), donde x es la entrada obtenida de una distribución de probabilidad desconocida, y y es la salida que se asocia con la entrada x, como se muestra en la Figura 2.4. El proceso que se busca realizar con la información anterior es que después de la fase entrenamiento, el algoritmo de aprendizaje proporcione soluciones también llamadas salidas, que se muestren adecuadas ante entradas nuevas o desconocidas.

- **Aprendizaje no supervisado**. En la Figura 2.5 se observa, para este tipo de aprendizaje, que los datos de entrada no están etiquetados, por lo que se dice que sólo se conocen los datos de entrada x. El objetivo de este tipo de aprendizaje es encontrar propiedades o estructuras ocultas en los datos.

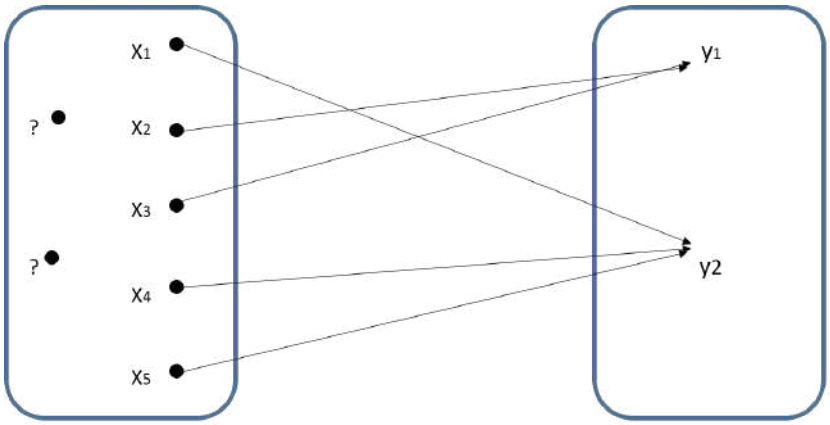


Figura 2.4: Aprendizaje supervisado [81].

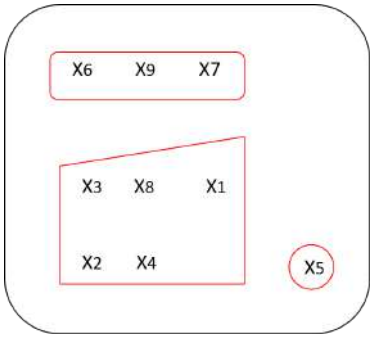


Figura 2.5: Aprendizaje no supervisado [81].

- **Aprendizaje semi-supervisado**. Para este tipo de aprendizaje se conocen muy pocas entradas. Al contrario del aprendizaje anterior, en éste los datos ya se encuentran etiquetados lo que conlleva a la existencia de un gran número de datos sin información de clase o etiquetas. Por lo que, debido a las características que presenta, éste ofrece la ventaja de realizar tareas de tipo supervisado y no supervisado. Su esquema de representación se puede observar en la Figura 2.6.

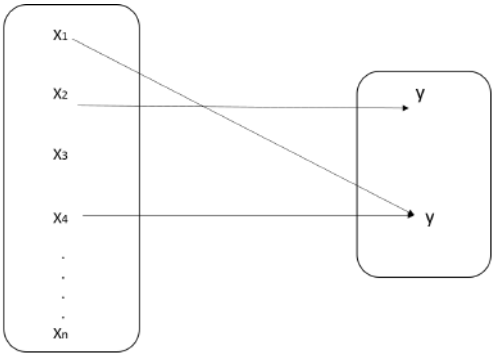


Figura 2.6: Aprendizaje semi-supervisado [81].

2.1.4.3. Aprendizaje de representaciones

Es un conjunto de técnicas de aprendizaje automático cuyo objetivo principal es aprender las características desde un bajo nivel hasta un nivel superior de los datos de entrada original, los cuales forman una representación que puede ser aprovechada de manera efectiva en una tarea de aprendizaje, por ejemplo la clasificación o el agrupamiento [11]. Cuando se emplea este tipo de aprendizaje se puede llegar a mejoras significativas con respecto a los modelos estándares supervisados en pruebas fuera de su dominio. Para realizar este proceso el aprendizaje de representaciones utiliza un enfoque no supervisado, el cual tiene la ventaja de no requerir información de clase [46].

2.1.4.4. Aplicaciones

De acuerdo con [81], algunas aplicaciones del aprendizaje de representaciones son las siguientes:

1. Reconocimiento de voz y procesamiento de señal. El reconocimiento de voz fue una de las primeras aplicaciones neuronales, el aprendizaje profundo y el aprendizaje de representaciones han tenido un fuerte impacto en esta área.

2. Reconocimiento de dígitos. En los inicios del aprendizaje profundo uno de los principales problemas que se abordaron fue la clasificación de imágenes de dígitos utilizando la base de datos publica MNIST.

3. Procesamiento de lenguaje natural. Las aplicaciones de procesamiento de lenguaje natural se basan en el aprendizaje de representaciones distribuidas de palabras llamadas *word embedding*. Por ejemplo, en el 2011 se aplicó esta idea junto con una arquitectura convolucional para desarrollar el sistema SENNA, el cual utilizó técnicas avanzadas de análisis sintáctico, etiquetamiento de categorías gramaticales, entre otros.

2.1.4.5. Algoritmos de aprendizaje

Los algoritmos de aprendizaje son utilizados en la implementación de las CNNs, estos se realizan mediante entrenamientos controlados por un agente externo, es decir, por un aprendizaje supervisado, el cual determina la respuesta que debería generar la red a partir de una entrada determinada [42].

- **El algoritmo de propagación hacia atrás (BP, por sus siglas en inglés)**. Es uno de los algoritmos de aprendizaje más populares en redes neuronales [6, 79]. Es como si el error se propagara desde la salida y de vuelta a las entradas, por ello se adaptó dicho nombramiento. De acuerdo con [66], este algoritmo realiza una aproximación a la minimización global lograda por el método de descenso más pronunciado. Como tal, el algoritmo puede verse como una generalización del algoritmo adaptativo

de mínimos cuadrados medios (LMS, por sus siglas en inglés). La principal diferencia entre BP y LMS es el modelo subyacente; a saber, LMS ocupa un combinador lineal adaptador, mientras que BP funciona en un perceptrón multicapa. El algoritmo de BP minimiza el costo funcional *E (n)* al alterar recursivamente el coeficiente basado en la técnica de búsqueda de gradiente.

- **Adam** [59]. Es un algoritmo para la optimización basada en gradientes de primer orden de funciones objetivas estocásticas, basado en estimaciones adaptativas de momentos de orden inferior. Este algoritmo es apropiado para objetivos no estacionarios y problemas con gradientes muy ruidosos y/o dispersos. Los hiperparámetros tienen interpretaciones intuitivas y generalmente requieren poco ajuste.

- **AdaGrad** [33]. AdaGrad ajusta de manera adaptativa la tasa de aprendizaje con respecto al gradiente cuadrado acumulado en cada iteración para cada dimensión. Este algoritmo se desempeña mejor para datos dispersos porque disminuye la velocidad de aprendizaje más rápido para parámetros frecuentes y más lento para parámetros poco frecuentes.

- **Descenso de gradiente estocastico (SGD, por sus siglas en inglés)** [16]. Es una simplificación drástica por lo que en lugar de calcular exactamente el gradiente de $E_n(f_w)$, cada iteración estima este gradiente sobre la base de un solo ejemplo elegido al azar. Dado que algoritmo es estocástico, no necesita recordar qué ejemplos se visitaron durante las iteraciones anteriores, puede procesar ejemplos sobre la marcha en un sistema implementado. En tal situación, el descenso del gradiente estocástico optimiza directamente el riesgo esperado, ya que los ejemplos se extraen aleatoriamente de la distribución de la verdad fundamental.

2.1.4.6. Métodos de análisis para la experimentación de aprendizaje automático

Los resultados que se obtienen durante la experimentación del aprendizaje automático corresponde a analizar los datos de manera que cualquier conclusión que se obtenga no sea subjetiva o se deba al azar. Por ello, se han desarrollado métodos que permiten analizar dichos resultados, a continuación se presentan algunos de ellos:

- **Validación cruzada** [6]. El siguiente ejemplo ilustrará la función de éste método. Para fines de replicación, la primera necesidad es obtener varios pares de conjuntos de entrenamiento y validación de un conjunto de datos X (después de haber omitido alguna parte como conjunto de prueba). Para obtenerlos, si la muestra X es lo suficientemente grande, se puede dividirla aleatoriamente en K partes, luego dividir aleatoriamente cada parte en dos y usar una mitad para el entrenamiento y la otra

mitad para la validación. Esto se realiza mediante validación cruzada, que es el uso repetido de la misma división de datos de manera diferente.

- **Estratificación** [6]. Dado un conjunto de datos X, se esperaría generar K pares de conjuntos de entrenamiento/validación, Ti, Vi K i = 1, a partir de dicho conjunto de datos. Por ello se espera que los conjuntos de entrenamiento y validación sean lo más grandes posible para que las estimaciones de error sean sólidas y, al mismo tiempo, se pueda mantener la superposición entre los diferentes conjuntos lo más pequeña posible. De ahí que es necesario el asegurarse que las clases estén representadas en las proporciones correctas cuando los subconjuntos de datos se mantengan en estratificación, para no perturbar las probabilidades previas de la clase. De ahí que la estratificación considera todas las dimensiones y, por lo tanto, pueda proporcionar un marco de muestreo que proporcione la representación de todas las características en cantidades proporcionales.

- **Validación cruzada de *K-fold*.** El conjunto de datos X se divide aleatoriamente en K partes de tamaño de validación de igual paso, X_i, $i = 1, ..., K$. Para generar cada par, Se mantiene una de las partes K como el conjunto de validación y se combinan las partes K - 1 restantes para formar el conjunto de entrenamiento. Haciendo este K veces, cada vez se deja fuera otra de las K partes, se obtiene K pares.

- ***Leave-one-out.*** Un caso extremo de validación cruzada de *K-fold* es dejar fuera uno donde dado un conjunto de datos de N instancias, solo se deja una instancia como el conjunto de validación (instancia) y la capacitación usa las instancias $N-1$. Luego obtenemos N pares separados dejando fuera una instancia diferente en cada iteración. Esta técnica se usa cuando la cantidad de ejemplos es pequeña.

2.1.4.7. Arquitecturas profundas

Las arquitecturas profundas permiten que los modelos computacionales compuestos de múltiples capas de procesamiento aprendan representaciones de datos con múltiples niveles de abstracción [81]. Cada nivel de arquitectura representa características en un nivel diferente de abstracción, definido como una composición de características de nivel inferior [64]. Estos métodos han ido mejorando rápidamente para desarrollar diversas herramientas concernientes, por ejemplo, al reconocimiento de voz, reconocimiento visual de objetos, detección de objetos e interpretación del lenguaje natural en base a texto [11].

2.1.4.8. Redes neuronales Artificiales

De acuerdo con [27], una ANN se define como un sistema de mapeo no lineal cuya estructura está basada en el comportamiento observado en los sistemas nerviosos de humanos y animales. Por lo que ésta, toma como ejemplos problemas previamente resueltos para construir un sistema capaz de tomar decisiones en base al entrenamiento aplicado y

mediante ello realizar clasificaciones. Las ANN constan de un número grande de procesadores simples, unidos por conexiones y pesos. Las unidades de procesamiento se denominan neuronas. Cada unidad recibe entradas de los nodos que la alimentan y genera una salida simple escalar que depende de la información local disponible.

2.1.4.9. Redes Neuronales Convolucionales

En las últimas décadas, se han considerado como una de las herramientas más poderosas ya que son capaces de manejar una gran cantidad de datos, esto es debido al rendimiento que se puede obtener mediante su método de reconocimiento de patrones, a estas herramientas se les conoce por el término de *Deep Learning* o *Deep Neural Network*, haciendo referencia a las ANNs con capas múltiples [3].

Las CNNs, toman este nombre por la operación lineal matemática entre matrices llamada convolución; de ahí que las CNN cuenten con capas múltiples, como por ejemplo la capa convolucional, la de no linealidad, de agrupación y la capa totalmente conectada [45]. Como se mencionó anteriormente, una CNN es una red neuronal profunda que está formada por una combinación en cascada de capas convolucionales, capas de *pooling* y capas totalmente conectadas [43]. Un ejemplo de una arquitectura para un modelo de una CNN para NLP se muestra en la Figura 2.7. En dicha figura se puede observar una matriz de entrada, donde cada fila representa una oración. A su vez cada oración pasa por un filtro realizando una operación de convolución, lo cual devuelve un mapa de características. Para esta arquitectura se cuenta con dos "canales" de vectores de palabras: uno que se mantiene estático durante todo el entrenamiento y otro no estático que se ajusta mediante la propagación hacia atrás. Por ello en esta arquitectura multicanal cada filtro se aplica a ambos canales y los resultados se agregan para calcular una característica por convolución mediante una determinada ventana de palabras. Posteriormente, se aplica la operación *max-over-time pooling* sobre el mapa de características, este esquema de *pooling* trata longitudes de oraciones variables. Hasta ese punto es el proceso para la extracción de una sola característica, por lo que ésta forma la penúltima capa y se pasan a una capa *softmax* completamente conectada cuya salida es la distribución de probabilidad sobre las etiquetas.

Debido a su arquitectura las CNN llegan a ser una excelente herramienta para el desempeño en problemas de aprendizaje automático [3, 58, 97]. Las CNN se han utilizado en clasificación de imágenes, visión por computadora y procesamiento de lenguaje [3].

2.1.4.10. Métodos de vectorización

Las características del aprendizaje automático son básicamente atributos numéricos desde los cuales cualquier persona puede realizar alguna operación matemática, como por ejemplo el producto punto. Por ello para convertir el texto en NL en algo que una máquina pueda entender, se realiza el proceso de conversión de texto a un vector numérico.

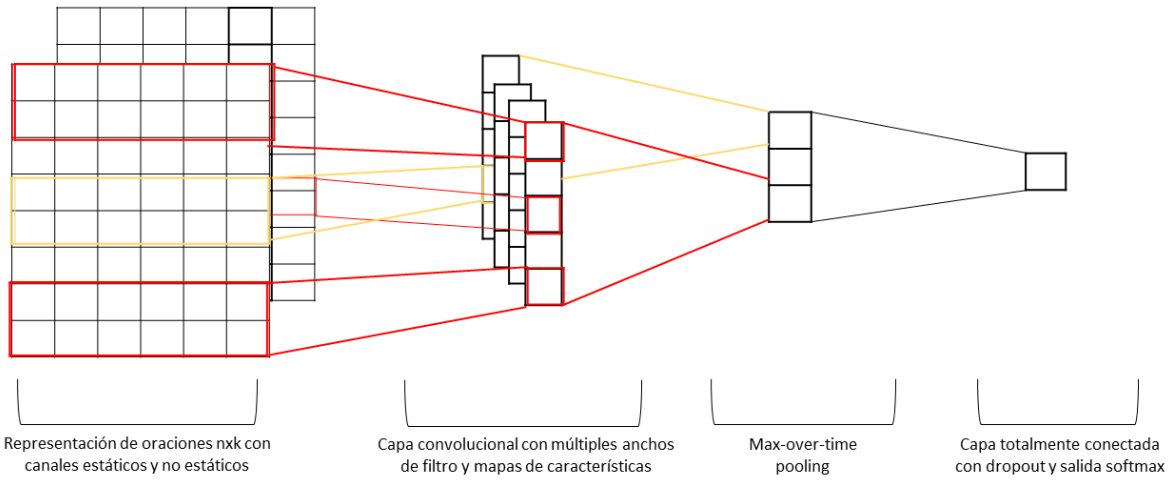
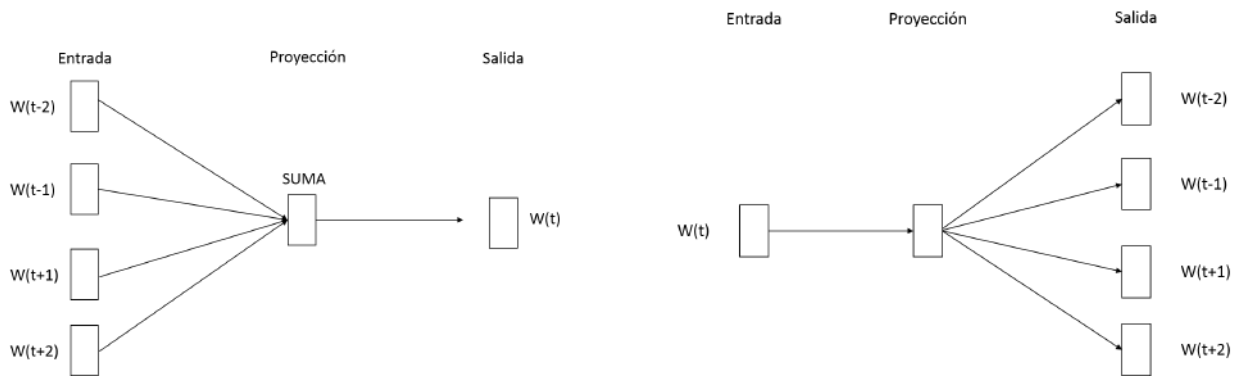


Figura 2.7: Modelo de arquitectura de CNN para NLP (traducido de [58]).

A dicho proceso se le denomina vectorización. Existen diversos métodos para llevar a cabo la vectorización, por lo que a continuación se hace una breve explicación con el fin de entender cómo funciona cada método.

- Modelo de Bolsa de Palabras Continuas (CBOW, por sus siglas en inglés). La idea de éste modelo es tratar un grupo de palabras con la finalidad de obtener una palabra que se relacione con su contexto [72]. Como se puede observar en la Figura 2.8a, el modelo recibe un vector de palabras (representando una oración con alguna palabra faltante), por lo que cada posición es una palabra, posteriormente mediante algoritmos de aprendizaje automático se realizan operaciones obteniendo una suma que da como salida la palabra que mejor encaja en el contexto de la entrada. Poniendo un ejemplo práctico, a la oración *"El", "gato", "se", "sobre", "el", "sillón"*, le falta una palabra. Teniendo como entrada al modelo de CBOW, dicha oración, el resultado del predictor sería la palabra *"sentó"*.

- El modelo Skip-Gram, también propuesto por [72], está basado en el modelo anterior, por lo que en lugar de predecir la palabra basada en el contexto se busca predecir el contexto basado en la palabra. En la Figura 2.8b se muestra gráficamente el proceso de clasificación para este modelo, en donde usa cada palabra actual como entrada del clasificador en la capa de proyección, y predice las palabras dentro de un cierto rango establecido por la palabra actual.



(a) Modelo CBOW (b) Modelo Skipgram

Figura 2.8: Modelos CBOW y Skipgram (traducido de [72]).

Este modelo está implementado en la herramienta Word2Vec, la cual realiza un pre-procesamiento de la entrada (conjunto de datos), es decir, una "limpieza" que busca eliminar caracteres no útiles (simbolos y números), para ello efectúa etiquetado gramatical (*part-of-speech tagging*). Después del pre-procesamiento, se crea un diccionario a partir de las palabras actuales y su frecuencia. A continuación, el diccionario se adapta para eliminar *stopwords*[5] e incluso ignorar las palabras cuya frecuencia sea menor de un límite dado. El diccionario, al que se denomina vocabulario, es la entrada para el entrenamiento de la red neuronal en donde se usarán pares de palabras encontradas en los datos de entrenamiento, estos pares dependerán del *windows size*[6]. La salida de éste es, mediante métodos de aprendizaje automático, una matriz de probabilidades (pesos). Esta matriz representa la cercanía que existe entre la palabra y el contexto determinado por el modelo Skipgram. En la Figura 2.9 se puede apreciar de forma práctica cómo es que se lleva acabo el entrenamiento en base a una oración.

[5]Palabras vacías son palabras en el idioma inglés consideradas como inútiles, por ejemplo, "the", "a", "an", "in".

[6]Windows size o ventana, es el parámetro que determina el número de palabras de cada contexto.

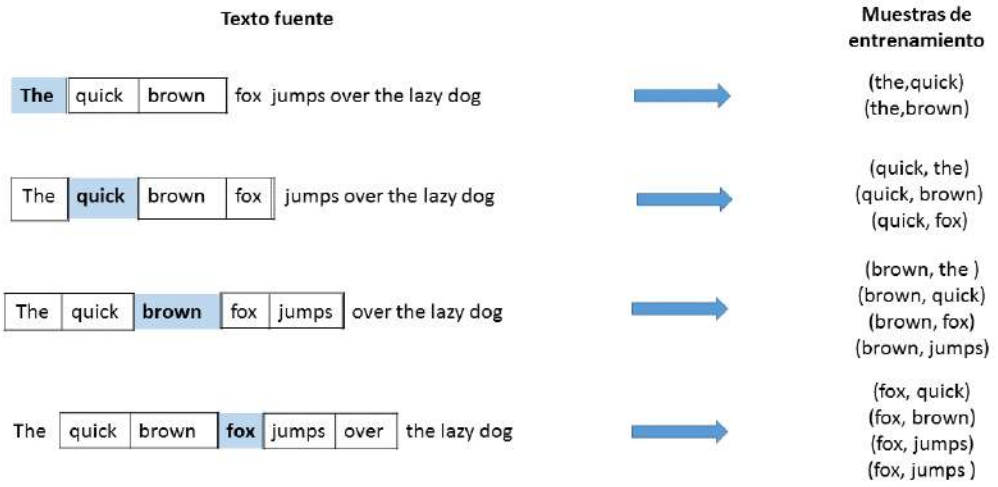


Figura 2.9: Modelo de Skipgram (traducido de [19]).

- Frecuencia de Términos (TF, por sus siglas en inglés). Cuenta el número de ocurrencia de palabras en un documento. Cada dimensión de un vector corresponde a una palabra que se encuentra al menos una vez en cualquier documento. El número de ocurrencia de una palabra que no se encuentra en un documento es cero.

- Frecuencia de términos – frecuencia inversa de documento (Tf-idf, por sus siglas en inglés). Representa la importancia o frecuencia de una palabra en un documento dentro de un conjunto de documentos, ignora las relaciones entre las palabras en un documento, y los n-gramas de palabras a veces se agregan como nuevas palabras antes de la vectorización.

- Word2Vec. Es un método para aprender incrustaciones de calidad para palabras. Las incrustaciones prácticamente significan representaciones numéricas dimensionales de palabras.

- Doc2vec. Es un método para aprender las presentaciones numéricas dimensionales de un documento. Este método también puede reducir el tamaño de la dimensión dramáticamente. Doc2Vec proporciona dos enfoques como Word2Vec: bolsa de palabras distribuida (DBOW, por sus siglas en inglés) y memoria de distribución (DM, por sus siglas en inglés).

2.1.5. El uso del aprendizaje automático para el procesamiento del lenguaje natural en la clasificación de requisitos

En esta sección se presentan ejemplos de investigaciones que emplearon NLP y aprendizaje automático para la clasificación de requisitos de software, cabe mencionar que las aportaciones aquí mostradas no fueron incluidas en el estado del arte ya que no utilizaron CNNs como parte del desarrollo de la propuesta. Sin embargo, sus contribuciones dejan ver que el uso de algoritmos de aprendizaje, métodos de vectorización y métodos de análisis para la experimentación contribuyen a que la precisión de la clasificación sea más óptima.

El aprendizaje automático se ha visto reflejado en estudios de extracción de la información en diferentes áreas que han utilizado técnicas de aprendizaje automático, tales como Márkov, modelos de máxima entropía, maquinas de soporte vectorial, entre otras [13, 14, 77]. Por ejemplo, en [26] se obtuvieron resultados favorables en el campo del NLP, ya que demostraron que una arquitectura de red neuronal unificada, entrenada simultáneamente con un conjunto de tareas relacionadas, ofrece etiquetados más precisos. Dicho estudio propuso una red neuronal profunda, la cual aprendió a discriminar entre oraciones genuinas y oraciones generadas sintéticamente. También la propuesta de [79], implica el uso de una arquitectura profunda para la extracción de información basada en caracteres, en particular del idioma chino. Esta propuesta consiste en un sistema unificado de extremo a extremo que, dado una cadena de caracteres, proporciona varias capas de extracción de características y predice las etiquetas de cada carácter, las características relevantes para el objetivo de extracción de la información son aprendidas automáticamente por la *backpropagation* en las capas más profundas del modelo de la red.

En el párrafo anterior se mencionaron investigaciones con relación a la extracción y clasificación de información las cuales señalan que los resultados que se han obtenido han sido favorables para la comunidad de AI, ya que han conseguido facilitar la extracción de información mediante el procesamiento de lenguaje natural. Por ello, para la comunidad de Ingeniería de Software, la integración que se puede crear entre la AI y la RE resulta de gran interés. En base a eso, a continuación se analiza en especifico cómo especialistas en el área han buscado optimizar el desempeño de la precisión en cuanto a la clasificación de FR y NFR.

En [25] se presentó un enfoque para recuperar y clasificar los NFR de documentos estructurados y no estructurados. El conjunto de datos que utilizaron fue Promise. Los autores primero utilizaron los términos del indicador ponderado que luego utilizaron para clasificar los requisitos adicionales. Para la usabilidad, seguridad, funcionamiento y rendimiento de los NFR, lograron una recuperación de entre el 20 % y el 90 % y una precisión de entre el 13 % y el 73 %, respectivamente. Para evaluar la efectividad del clasificador de NFR se aplicó una técnica de validación cruzada sin interrupciones (*leave-one-out cross*

validation) y para el análisis de los resultados de la clasificación se utilizó una matriz de confusión que determina la representación de positivos verdaderos y falsos, así como negativos verdaderos y falsos.

La investigación presentada por [47] tuvo por objetivo automatizar el proceso de detección de oraciones de NFRs mediante el uso de un clasificador de texto equipado con un etiquetador llamado POS (*part-ofspeech*), el cual consiste en marcar una palabra en un texto como correspondiente a una parte particular del habla, basada tanto en su definición como en su contexto, es decir, su relación con palabras adyacentes y relacionadas en una frase, oración o párrafo. La clasificación de características o palabras claves, fue considerada por dos medidas de probabilidad diferentes: 1) Medida de probabilidad no suavizada y 2) Medida de probabilidad suavizada. Utilizaron el conjunto de datos Promise. Los resultados reportados fue de una precisión del 98.56 % utilizando validación cruzada de 10 iteraciones. Como mencionan estos investigadores su propósito fue ayudar a los analistas de software a resaltar los NFR en los documentos de especificaciones de software, lo cual buscó evitar una mayor supervisión.

En [18] evaluaron un enfoque semi supervisado para la clasificación de los NFR mediante la maximización de expectativa con clasificadores Bayesianos *naïve*. Esto fue para evaluar la efectividad de los clasificadores y calcular la precisión de categorizar la recopilación de requisitos utilizando diferentes tamaños del conjunto de entrenamiento con datos de Promise. Se obtuvo una precisión en la clasificación dentro de un top 5 de requisitos de 90.11 % y considerando un top 10 de 93.66 %.

En [98] realizaron un estudio empírico para clasificar automáticamente los NFR utilizando expresiones multipalabra (MWE, por sus siglas en inglés) y una máquina de soporte vectorial con *kernel* lineal como clasificador. Se utilizó el conjunto de datos Promise. Los resultados mostraron que la clasificación automática de los NFR es mejor cuando el conjunto de datos es grande, ya que cuantas más muestras haya en una categoría en el conjunto de datos mejor será el rendimiento.

En [90] desarrollaron un enfoque que examina documentos no estructurados utilizando el procesamiento automatizado del lenguaje natural. Se analizó qué tipos de documentos contienen NFR, los cuales fueron asignados a categorías de NFR (por ejemplo, capacidad, confiabilidad y seguridad) y se midió la eficacia con la que se puede identificar y clasificar las oraciones de NFR dentro de estos documentos. Para ello, se utilizaron SVM y el algoritmo de Bayesiano *naïve*.

En [82] realizaron dos contribuciones para la clasificación de NFRs en la Especificación de Software, la primera es una ontología llamada Concordia, la cual tiene anotaciones en un manual que contiene documentos provenientes de diferentes documentos de requisitos,

estos documentos fueron procesados con Arquitectura General para Ingeniería de Texto (GATE, por sus siglas en inglés), en particular con su componente llamado ANNIE; y la segunda es un clasificador basado en Máquinas de Soporte Vectorial, el cual tiene un enfoque para las tareas de clasificación de requisitos donde obtuvieron un 84.96 % de medida.

En [80] proponen un sistema automático de identificación de NFR a partir de algoritmos de clasificación de oraciones de similitud difusa de vecinos más cercanos, (FSKNN, por sus siglas en inglés), con la adición de factores semánticos como el desarrollo del término de *hipernim* y la medición de la relación semántica entre cada término y cada categoría de aspectos de calidad. Dicho estudio utilizó el conjunto de datos Promise. El resultado de esta investigación es que el método FSKNN semántico puede reducir el valor de la tasa de error en un 21.9 %, y también elevar el valor de la precisión en un 43.7 %. Sin embargo, se produce la disminución en el rendimiento de este método ya que el proceso de información de patrones de entrenamiento tiene un proceso adicional de búsqueda de relación semántica entre cada categoría de NFR.

En [68] realizaron la identificación de NFR mediante similitud semántica de palabras con el análisis semántico latente (LSA, por sus siglas en inglés) y agrupación de palabras requeridas (*Required word clustering*). Para determinar el rendimiento de la clasificación se realizó un análisis de 3 métodos que fueron Similitud semántica de texto (TSS, por sus siglas en inglés), Modelo de espacio Vectorial (VSM, por sus siglas en inglés) e Indización semántica latente (LSI, por sus siglas en inglés), estos métodos miden en términos de precisión (*precision*) y recuperación (*recall*). El mejor rendimiento fue obtenido por VSM, la principal desventaja entre TSS y LSI reside en su proceso de calibración computacionalmente costoso. Los tres métodos lograron una medida en umbral del 70 %, esta baja precisión se debe a las impurezas de los conjuntos de datos que manejan, en particular por el proceso de agrupamiento, ya que algunos clústeres de NFR contienen ruido, o palabras no relacionadas. Otro aspecto que llevó a la baja precisión es que los conjuntos de datos no eran grandes.

Por otro lado, en [89] se propuso un enfoque basado en reglas para la clasificación automatizada de NFRs de SRS, considerando el análisis de requistos mediante GATE, donde cada oración fue anotada por roles temáticos usando reglas JAPE (complemento de GATE). Dicho estudio reportó la comparativa entre el corpus de Promise con un 97.21 % y Concordia obtuvo un 94 % de medida, cabe mencionar que dicha medida fue en base a subclases de NFRs.

En el caso de [62] se estudió la precisión con la que se puede clasificar automáticamente los FR y NFR del conjunto de datos Promise con aprendizaje supervisado. También se evaluó la precisión con la que se pueden identificar las clases de NFR, en particular: usabilidad, seguridad, funcionamiento y rendimiento. Esta investigación se basó en un en-

foque de aprendizaje automático supervisado utilizando metadatos, características, léxicas y sintácticas. Se emplearon también estrategias de *under* y *over sampling*, SVMs y validación cruzada. Para la clasificación e identificación de NFR de las clases mencionadas anteriormente, se obtuvo un rendimiento arriba del 70 %.

Finalmente, la contribución presentada en [1] es la revisión de cómo funcionan varios enfoques de aprendizaje automático para el contexto de los NFRs y FR. Dicha investigación también utilizó el conjunto de datos de Promise. Se encontraron diferencias significativas en el rendimiento del enfoque de LDA, modelado de temas término, y el clasificador Bayesiano *naïve* para la subclasificación de NFRs.

2.2. Estado del arte

La sección anterior representa las bases teóricas que permiten tener un mejor entendimiento del desarrollo de las propuestas de la revisión de literatura relacionada con el tema de tesis, las cuales se presentan a continuación.

2.2.1. Clasificación automática de requisitos basados en redes neuronales convolucionales

2.2.1.1. Propuesta

La investigación realizada en [97] pretendió clasificar automáticamente los elementos de contenido de una especificación de requisitos de lenguaje natural como requisito o información utilizando CNNs.

2.2.1.2. Desarrollo de la propuesta

Para construir un clasificador que sea capaz de distinguir información de los requisitos en oraciones en lenguaje natural, se utilizó el proceso de Descubrimiento de Conocimientos en Bases de Datos (KDD, por sus siglas en inglés). Este proceso describe los pasos necesarios y los riesgos a tener en cuenta al crear conocimiento a partir de datos sin procesar utilizando técnicas de extracción de datos. El proceso contiene los 9 pasos que se muestran a continuación. En cada paso se identificó la forma en la que los investigadores aplicaron el KDD para su propuesta:

1. Entender el dominio de aplicación. Se realizó una distinción de lo que es una especificación de requisitos, por lo que caracterizaron una especificación de requisitos como un conjunto de elementos de contenido. Los elementos de contenido fueron identificados como partes atómicas de una especificación de requisitos que generalmente contenían una sola oración o una figura. De ahí que los elementos de contenido fueran asociados a diferentes atributos. Por ello, para comprender mejor los diferentes tipos

de contenido, almacenados en una especificación de requisitos, los investigadores se dieron a la tarea de examinar un conjunto de especificaciones de requisitos de un fabricante automotriz. De esos documentos solo se enfocaron en el texto que se encontraba estructurado, por lo que definieron atributos como requisitos e información.

2. Crear un conjunto de datos. El conjunto de datos que crearon fue de la base de datos de documentos DOORS, mencionada en el punto anterior, por lo que completaron los siguientes criterios de selección para 89 documentos: 1) El documento debería ser una especificación que describiera un solo componente electrónico del vehículo. 2) Todos los documentos debían estar escritos en el mismo idioma. 3) Para entrenar al clasificador, los elementos de contenido de los documentos seleccionados debían clasificarse como información o requisito. 4) La clasificación de los elementos del contenido dentro del documento debería ser confiable.

3. Preprocesar y limpiar los datos. Todos los elementos de contenido fueron preprocesados mediante pasos de preprocesamiento estándar, como convertir texto en minúsculas y eliminar palabras de parada (*noise words*).

4. Transformar los datos utilizando la técnica de palabras incrustadas *Word2Vec*. Esto creó un diccionario de palabras frecuentes. Es decir, con un mapeo de cada palabra frecuente se asignó a un vector, a su vez, con el objetivo de convertir oraciones en matrices.

5. Elegir la tarea apropiada para la minería de datos. Consistió en determinar el tipo elementos dado el contenido de un documento de clasificación. En este sentido, se determinó que el contenido se clasificaría como requisito o información.

6. Elegir el algoritmo. Se seleccionó a las CNNs debido al éxito que tienen en la resolución de los problemas comunes del lenguaje natural.

7. Emplear el algoritmo seleccionado. Los hiper parámetros del modelo, como el tamaño de incrustación, los tamaños de filtro y el recuento de filtro por tamaño, fueron elegidos mediante un proceso iterativo. Se comenzó con pocos filtros pequeños, posteriormente, se aumentó tanto el recuento como el tamaño de los filtros hasta que el rendimiento de la red ya no siguió aumentando. Se logró el mejor rendimiento utilizando el tamaño de incrustación 128, filtros de 1, 2 y 3 y 64 por tamaño (192 filtros en total). Para entrenar a la red el conjunto de datos se dividió al azar en 90 % de datos de entrenamiento y 10 % de datos de prueba. La red fue entrenada con los datos de entrenamiento utilizando el gradiente estocástico.

8. Evaluar el enfoque seleccionado. Se evaluó el rendimiento de la red entrenada utilizando medidas estándar. Esto incluyó la precisión de la red en el conjunto de entrenamiento y prueba, así como la precisión, recuperación y puntuación *f1* de ambas clases objetivo en el conjunto de prueba.

9. Usar el conocimiento adquirido. Para aplicar este enfoque en la industria se planteó integrar una CNN pre-entrenada en una herramienta.

2.2.1.3. Resultados

Los resultados que obtuvieron estos investigadores se resumen en el Cuadro 2.3. Como se puede observar la precisión de requisitos fue de un 0.733, para la clasificación de información se obtuvo un 0.896, y para la recuperación un 0.885 y un 0.754, de requisitos e información respectivamente.

Clase	Precisión	Recuperación	f1
Requisito	0.733	0.885	0.802
Información	0.896	0.754	0.819

Cuadro 2.3: Tabla de resultados (traducido de [97]).

Considerando lo anterior se concluyó que la precisión de la CNN propuesta puede mejorar al aumentar la cantidad de datos de entrenamiento. Es decir, al incluir documentos de otros tipos, como las especificaciones de sistemas de componentes múltiples. De manera similar, también se aumentó la calidad para filtrar elementos de contenido mal clasificados e incorrectos mediante la aplicación cuidadosa de técnicas semiautomáticas.

2.2.2. Hacia el soporte de la Ingeniería de Software utilizando Deep Learning: Un caso de clasificación de requisitos de software

2.2.2.1. Propuesta

El estudio realizado en [53], se propuso el uso de Deep Learning (DL, por sus siglas en inglés) para clasificar los requisitos de software (SR, por sus siglas en inglés) sin necesidad de hacerlo manualmente. El modelo propuesto se basa en una red neuronal convolucional (CNN), debido al comportamiento que tiene en las tareas relacionadas con el lenguaje natural, de acuerdo a su revisión del estado del arte. Para la evaluación del modelo se utilizó el conjunto de datos de Promise.

2.2.2.2. Desarrollo de la propuesta

El modelo creado se basa en una CNN. Para evaluar dicho modelo se utilizó el conjunto de datos PROMISE. Antes de implementar el modelo, se realizó un preprocesamiento que consistió en lo siguiente:

1. Generar vectores de incrustación de palabras (WEC, por sus siglas en inglés), utilizando *Skip-gram* previamente entrenado con Wikipedia. El modelo Skip-gram apunta a predecir palabras de contexto

$$wcontext = ((w_t)i, ..., (w_t)1, (w_t) + 1, ..., (w_t) + i)$$

donde

$$i \geq 1$$

recibe una palabra $w(t)$ como entrada . La proyección es una representación de baja dimensionalidad y espacio vectorial continuo de la palabra $w(t)$.

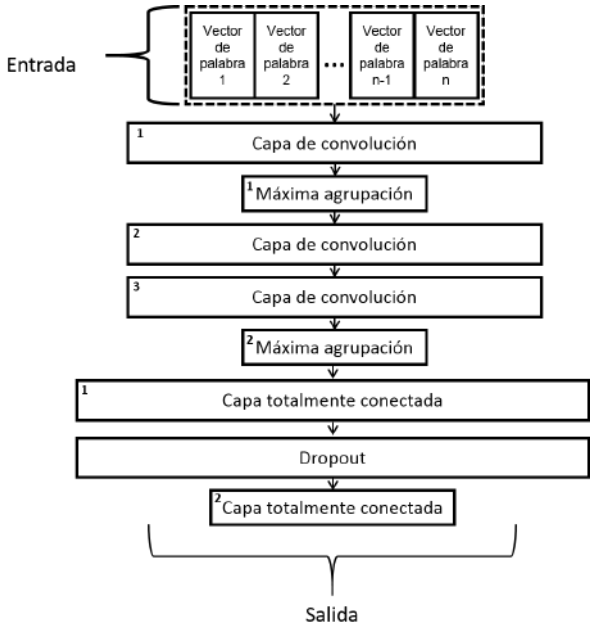
2. Generar *one-hot-encoding*. Todas las palabras en los requisitos que coincidan se cambian por un 1, que incluye la representación de esa palabra. Las palabras que no existen en el espacio de incrustación se reemplazan por vectores con 0s.

3. Utilizar el resultado de lo anterior para el modelo CNN. Esto es una concatenación de WEC que representan cada requisito de software.

La configuración propuesta por los investigadores se encuentra en la Figura 2.10. En la Figura 2.10a, se observa la arquitectura de la CNN que, como se mencionó anteriormente, recibe una entra de WEC que servirá para el entrenamiento de la CNN. Esta red cuenta con tres capas de convolución, dos de máxima agrupación y dos capas totalmente conectadas, por lo que ésta última obtiene la salida del entrenamiento. Y en la Figura 2.10b se describen las dimensiones que se establecieron por cada capa.

2.2.2.3. Resultados

Para evaluar el modelo propuesto, los investigadores utilizaron el promedio de las siguientes medidas aplicadas a cada pliegue (*fold*) a lo que obtuvieron: precisión= 80 %, recuperación= 78.5 % y medida de f = 77 %. Además, se menciona que el modelo propuesto presenta una estabilidad muy buena que se refleja en la baja desviación estándar (<0.035) en precisión, recuperación y medida de *f*. En ese sentido, los resultados obtenidos muestran que, aunque el conjunto de datos se procesó de manera deficiente para convertirse a propósito en información del modelo, con el fin de medir qué tan bien las técnicas DL pueden crear y transformar características útiles por sí mismas para clasificar SR.



(a) Arquitectura de CNN

Capa	Dimensión
Entrada	MAX x 400
Convolución 1	MAX x 400 x 32
Máxima agrupación 1	$\frac{MAX}{2}$ x 400 x 32
Convolución 2	$\frac{MAX}{2}$ x 200 x 64
Convolución 3	$\frac{MAX}{2}$ x 200 x 64
Máxima agrupación 2	$\frac{MAX}{4}$ x 100 x 64
Totalmente conectada 1	128
Dropout	0.5
Salida: Totalmente conectada 2	12

(b) Dimensiones de la CNN

Figura 2.10: Arquitectura y dimensiones de la CNN (traducido de [53]).

2.2.3. Clasificación de requisitos de software mediante incrustaciones de palabras y redes neuronales convolucionales

2.2.3.1. Propuesta

La investigación realizada en [37] exploró la aplicación de técnicas de aprendizaje profundo en la clasificación de requisitos de software, específicamente el uso de incrustaciones de palabras para la representación de documentos cuando se realiza un entrenamiento en una CNN.

2.2.3.2. Desarrollo de la propuesta

En primer lugar se realizó una limpieza en los datos, eliminando palabras de parada del *corpus* y lematizandolas y aplicando la estratificación al conjunto de datos. La Figura 2.11 muestra la arquitectura de la red que se empleó para la propuesta. En la capa de incrustación se utilizó el tensor de entrada para transfórmala en una matriz de incrustación, lo que significa que el texto de entrada sin procesar se conviertió en una pila de sus incrustaciones respectivas. Después de la preparación en la capa de incrustación, se definió una colección de capas convolucionales y de agrupación que actuaron en paralelo, cada capa de convolucional se agrupó al máximo y la salida de toda la agrupación se concatenó en un vector de características final para canalizar a la capa de *dropout*. Y por último, se implementó una capa de salida, es donde se aplicó una función de regularización de

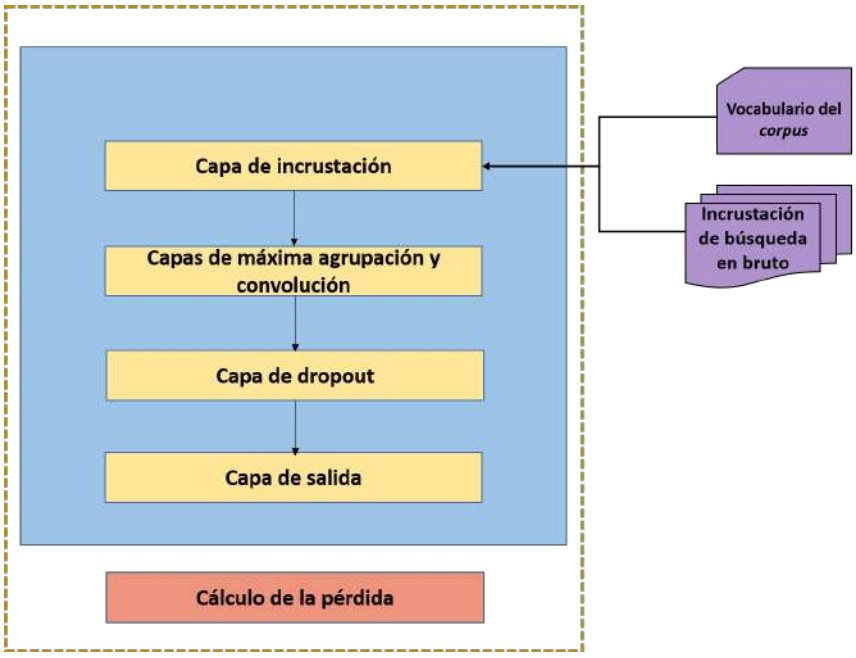


Figura 2.11: CNN de palabras incrustadas [37].

softmax para convertir los valores de salida de la capa de *dropout* en valores normalizados para la predicción de clase.

2.2.3.3. Resultados

Los resultados obtenidos utilizando distintos métodos de vectorización fueron los siguientes: para *random* se obtuvo un 73.1 % de precisión, usando *Word2Vec* se obtuvo un 80.4 % y para *fastText* arrojó un 79.2 %. De acuerdo con [37], el uso de una CNN para la clasificación de requisitos puede tener mayor precisión cuando se trata de multi-etiqueta, término que se empleo para referirse a cada clase de los NFR.

2.2.4. Análisis comparativo de propuestas

Las propuestas mencionadas anteriormente son comparadas en este apartado para identificar cuáles son las técnicas de aprendizaje automático que se utilizaron y cuáles fueron los resultados que se obtuvieron. Primero se establecerá la definición de los criterios, lo que permitirá entender la comparativa que se realiza en la sección posterior.

2.2.4.1. Definición de criterios

Los criterios a evaluar son las características comunes que se identificaron durante el análisis de dichas investigaciones. Estas características están presentes durante el desarrollo de las propuestas. La primera de ellas es el preprocesamiento, ya que la limpieza de los documentos permitió resultados más favorables que un conjunto de datos sin preprocesar

[37, 97], esto sucede debido al contenido no relevante que pueda tener el conjunto de datos, que va desde caracteres que no se encuentran en el alfabeto hasta dígitos o palabras de parada. La segunda es el método de vectorización empleado, ya que es necesaria la incrustación de palabras como entrada para la arquitectura a utilizar. La tercera es la arquitectura usada, todas las propuestas fueron aplicadas por medio de una CNN, también se muestra la configuración de cada modelo y el tipo de validación. Por último, se presentan los resultados obtenidos por cada propuesta. Cabe mencionar que se tomaron los resultados respectivos a la clasificación de NFR, a excepción de [53] ya que clasificaron FR y NFR.

2.2.4.2. Comparación sobre las propuestas

Se identificó que las tres propuestas ocuparon el mismo preprocesamiento, arquitectura y vectorización, y se obtuvieron diferentes resultados de precisión en cuanto a la clasificación (Ver Cuadro 2.4). Por lo que se determinó que existieron variables que influyeron en dichos resultados: la configuración de la arquitectura, el método de validación y el número de clases que se tomaron del conjunto de datos utilizado. Por ejemplo, en la propuesta de [97] no se muestra cual es la configuración que se utilizó, y no se menciona algún tipo de validación empleado, además los autores mencionan que debido a que el conjunto de datos que utilizaron contiene poca información, la clasificación de requisitos obtuvo una precisión del 73 %. Por otro lado, [53] obtuvo un 80 %, utilizando la validación cruzada con parámetros de k=10 y la aplicación del optimizador *AdamOptimizer*. Para el caso de [37], se utilizan los mismos parámetros de [53] para la validación cruzada y el mismo método de optimización, pero con resultados de precisión del 80.4 % con el método de vectorización *Word2Vec*.

Estudios	Desarrollo					Métrica de evaluación
	Pre procesamiento	Vectorización	Arquitectura	Algoritmo de aprendizaje	Validación	Precision
[97]	✓	Word2Vec	CNN	SGD	-	73%
[53]	✓	Word2Vec	CNN	Adam	Cruzada	80%
[37]	✓	Random, Word2Vec, fasttext	CNN	Adam	Cruzada	73.1% 80.4% 79.2%

Cuadro 2.4: Comparación sobre las propuestas.

2.2.4.3. Resultados de la comparativa

En la revisión de dichas propuestas se observó que han utilizado estrategias similares con distintos resultados de precisión. La diferencia radica en que una de ellas utiliza la estratificación del conjunto de datos [37]. En base al análisis de la clasificación de requisitos en la matriz de confusión de [53] se observa la notoria inclinación sobre una clase, que es la de FR, característica que resalta cuando no se utiliza la estratificación, de ahí la explicación al resultado de la precisión que obtuvieron.

2.2.4.4. Conclusiones

En base a lo analizado, se pudo observar que la estratificación permite obtener un resultado de precisión óptima, debido a que se realiza un balanceado de datos, lo cual evita la inclinación de una clase específica. Por ello se aplicará tanto en la etapa de entrenamiento como en la etapa de pruebas para esta tesis. También se pretende emplear distintos algoritmos de aprendizaje lo que podrá aumentar la precisión en cuanto a la clasificación, así como la implementación de otros métodos de vectorización. Además se planea hacer una búsqueda de malla, lo que permitirá determinar la mejor precisión de la CNN.

Capítulo 3

Metodología

Las propiedades únicas de los documentos de requisitos de software, presentados en el Capítulo 2 de esta tesis, plantean una serie de preocupaciones cuando se consideran las técnicas de aprendizaje profundo, ya que la naturaleza de los datos no se alinea con las condiciones convencionalmente adecuadas para las aplicaciones de aprendizaje profundo como las que se muestran a continuación:

- El documento de requisitos de software es corto, lo que da como resultado muestras de datos insuficientes. Los análisis de aprendizaje profundo generalmente requieren grandes volúmenes de datos, del orden de millones o miles de millones.

- El documento de requisitos de software cuenta con 12 clases, de las cuales existen algunas que cuentan con tres ejemplos por clase, lo que orilla a buscar una alternativa para poder realizar un balanceo de ejemplos entre clases.

- Los ejemplos por clase, a los que llamaremos oraciones, tienen en su contenido una serie de palabras de ruido o *stopwords* las cuales podrían interferir con el contexto relevante para determinar a que clase pertenece cada una de estas.

Debido a lo anterior se busca evaluar si el uso de representaciones de palabras en características proporcionadas a través de incrustaciones de palabras puede enriquecer las características de estos documentos y compensar su escasez. Especialmente porque los documentos de requisitos a menudo se caracterizan por la escritura formal y la jerga específica del dominio, las palabras que componen el texto del documento debe ser clave para identificar las características de los requisitos individuales. Dado que se afirma que las incorporaciones de palabras preservan las relaciones semánticas entre palabras que de otro modo se pierden en los métodos de vectorización tradicionales como TF-IDF, se determinó

investigar si estas ventajas pueden contrarrestar la calidad deficiente de las muestras de requisitos.

3.1. Especificación de hardware y software

Para analizar los atributos mencionados al inicio de este capítulo, se pudo identificar que se necesitaba orquestar la preparación de datos, la capacitación de clasificadores y la evaluación del desempeño necesarios para evaluar el impacto de las tecnologías mencionadas anteriormente en relación con las interrogantes con la condición del dominio. Por ello resultó indispensable el uso de una tarjeta gráfica que tenga la capacidad de afrontar con dicha condición, para la experimentación la máquina cuenta con dos tarjetas de la marca NVidia GeForce rtx 2080 TI, la cual por su naturaleza será de utilidad para procesar la arquitectura de la CNN.

El software necesario para poder desarrollar la aplicación de esta investigación se soporta mediante la ayuda de herramientas de aprendizaje automático como TensorFlow 2.3.0, desarrollada por Google, y Keras 2.4.0, Scikit-Learn, Gensim y NLTK, esto es para realizar el preprocesamiento, la vectorización de palabras y el uso de los clasificadores,así como la creación del modelo de la CNN.

3.2. Conjunto de datos

El conjunto de datos de atributos de calidad NFR, también conocido como el corpus PROMISE [87], es una compilación de especificaciones de requisitos para 15 proyectos de software desarrollados por estudiantes de la Universidad DePaul como un proyecto de término para un curso de Ingeniería de Requisitos, el idioma del contenido es en inglés. El conjunto de datos consta de 326 NFR y 358 requisitos FR. El conjunto de datos NFR se presta, para efectos de esta tesis, a la clasificación de etiquetas múltiples de varios tipos de requisitos NFR. Una muestra del contenido de este conjunto se muestra en el Cuadro 3.1.

Requisito	Clase
The system shall have a MDI form that allows for the viewing of the graph and the data table.	F
The product shall be installed by an untrained realtor without recourse to separately-printed instructions.	US
The system shall have basic data integrity checking to reduce the possibility of incorrect or invalid data being introduced.	SE
The product must work with most database management systems (DBMS) on the market whether the DBMS is colocated with the product on the same machine or is located on a different machine on the computer network.	O
The preferred repair facility ratings shall be saved within 5 seconds. The save shall occur within 5 seconds.	PE
The website shall be attractive to all audiences. The website shall appear to be fun and the colors should be bright and vibrant.	LF
The system shall be capable of processing 100% of nursing students and their classes for the next 10 years.	SC
The system shall be available for use between the hours of 8am and 6pm.	A
Promotional updates to the website should take a day to update.	MN
The System shall meet all applicable accounting standards. The final version of the System must successfully pass independent audit performed by a certified auditor.	L
The product shall operate in offline mode whenever internet connection is unavailable.	FT
The product is expected to run on Windows CE and Palm operating systems.	PO

Cuadro 3.1: Promise [70].

3.3. Estructura experimental

La propuesta para esta tesis es realizar la estructura experimental que se muestra en la Figura 3.1. Dicha figura muestra el proceso a seguir para realizar la clasificación iniciando con el preprocesamiento de datos, seguido de la vectorización de palabras, continuando con la implementación en el modelo seleccionado y por último con la evaluación del modelo, en las siguientes secciones se explicará lo que implica hacer cada uno de los procedimientos mencionados. Dicha estructura fue basada en el trabajo de [18, 37, 53, 62, 97], en donde se hace la unión de las técnicas y procedimientos comunes para realizar la clasificación de requisitos. A continuación se explica de forma breve la propuesta de esta tesis durante dicha estructura:

- **Preprocesamiento.** No solo limitarse a usar la biblioteca llamada *CountVectorizer* como [37], la cual sirve para eliminar *stopwords* y vectorizar; sino que antes de pasar a esa parte realizar una limpieza previa que elimine más *stopwords*

- **Vectorización de palabras**. Usar los vectorizadores que utilizaron [37, 53, 62].

- **Modelo ML y evaluación**. Si bien el objetivo de esta tesis es el de implementar la estructura de una CNN para lograr la clasificación, sin embargo, se mostrará la implementación en los clasificadores Naïve Bayes y SVM para demostrar la eficiencia de la CNN utilizando los mismos criterios de evaluación cómo son precisión, medida F1 y recuperación que utilizó [37]. Además se pretende utilizar la estratificación de

datos y una búsqueda de malla que permitirá encontrar los valores óptimos para obtener mejores criterios de evaluación.

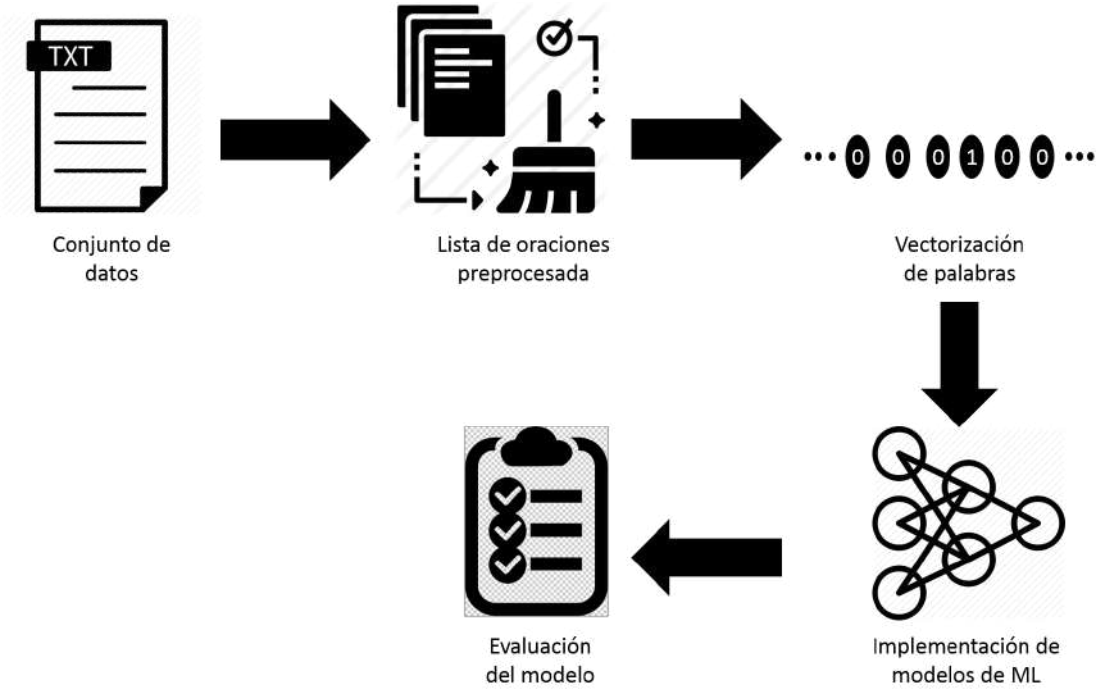


Figura 3.1: Estructura experimental.

3.3.1. Preprocesamiento

El procesamiento de texto significa llevar el texto en su forma original a una forma numérica la cual será posible ser procesado por las técnicas de ML. Esto, con el fin de que los clasificadores puedan analizar la información que este recibiendo. Para dicho trabajo es necesario emplear NLP, ya que su objetivo es interpretar el lenguaje natural para posteriormente convertirlo en un formato legible por la máquina. Los pasos para realizar dicha conversión son los siguientes:

- **Limpieza**. Consiste en deshacerse de las partes menos útiles del texto a través de la eliminación de palabras de ruido o *stopwords*, tratando con mayúsculas y signos de puntuación, caracteres especiales, entre otros.
- **Anotación**. Se trata de la aplicación de un esquema a textos. Las anotaciones pueden incluir el etiquetado *part-of-speech* y el marcado estructural.
- **Normalización**. Es la traducción (mapeo) de términos en el esquema o reducciones lingüísticas a través de *Stemming*, *Lemmazation* y otras formas de estandarización de texto.

- **Análisis**. Se trata de un sondeo estadístico, manipulación y generalización del conjunto de datos para el análisis de características.

3.3.2. Formato de entrada para el clasificador

Los clasificadores que serán presentados como pruebas para esta tesis esperan que los datos tengan la forma de una lista de cadenas de texto de requisitos(a los que se denominarán como ejemplos) en forma de un vector de palabras únicas *one-hot* como se observa en el Cuadro 3.2. Esto significa que existirá un vector de palabras únicas donde se realizará una búsqueda dentro de cada ejemplo de la lista y en caso de encontrar cualquiera de esas palabras únicas dentro del ejemplo se le asignará un *1*, en caso contrario será un *0*.

	Palabra 1	**Palabra 2**	**Palabra *m***
Ejemplo 1	1	0	1
Ejemplo 2	1	1	1
Ejemplo *n*	0	0	1

Cuadro 3.2: Ejemplificación de la vectorización de palabras únicas en formato *one-hot*

Como se puede observar tanto para los ejemplos como las clases se espera obtener un conjunto de arreglos para que el clasificador pueda procesar la información. Para ello es necesario esperar con que dichos arreglos cuenten con las siguientes características:

- Los ejemplos deben de estar limpios y tokenizados
- Los ejemplos deben estar en forma de lista e irse añadiendo a esta una vez que se ha realizado el paso anterior
- La lista de ejemplos debe tener la forma de un vocabulario, esta se realiza utilizando un extractor de características.

Posteriormente se procede a realizar la incrustación de palabras o *word embedding*, este método representa las palabras como vectores de palabras, ya que recopilan más información en menos dimensiones. Su objetivo es mapear el significado semántico en un espacio geométrico. Esto con el fin de obtener la entrada al clasificador el cual tendrá una forma de matriz de palabras incrustadas.

3.3.3. Estrategias de muestreo sobre datos no balanceados

El objetivo de las estrategias de muestreo es evitar el desequilibrio en la distribución de la clase que constantemente presentan los conjuntos de datos, dicho desequilibrio hace que los algoritmos de aprendizaje automático tengan un bajo rendimiento en la clase minoritaria; ya que el costo de clasificarla erróneamente suele ser mucho más alto que el costo de otras clasificaciones incorrectas [37,62,95]. Por ello, al observar la distribución del conjunto de datos Promise (ver Figura 3.2) se opta por implementar algunas estrategias de muestreo, como se señalan a continuación [37, 62, 95]:

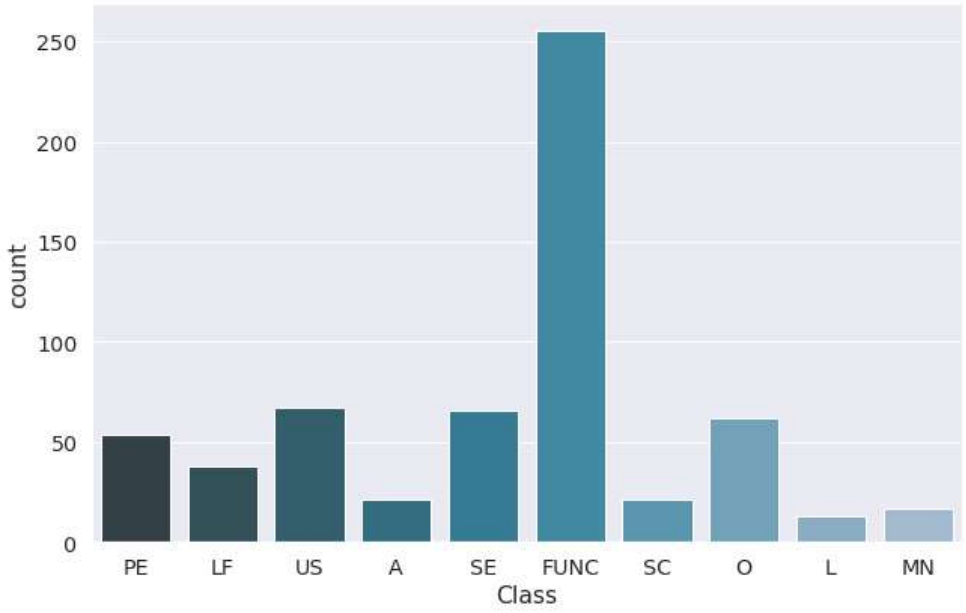


Figura 3.2: Distribución de clases del conjunto de datos Promise.

- **Undersampling**. La clase mayoritaria se submuestrea o reduce para lograr una distribución equilibrada (ver Figura 3.3).

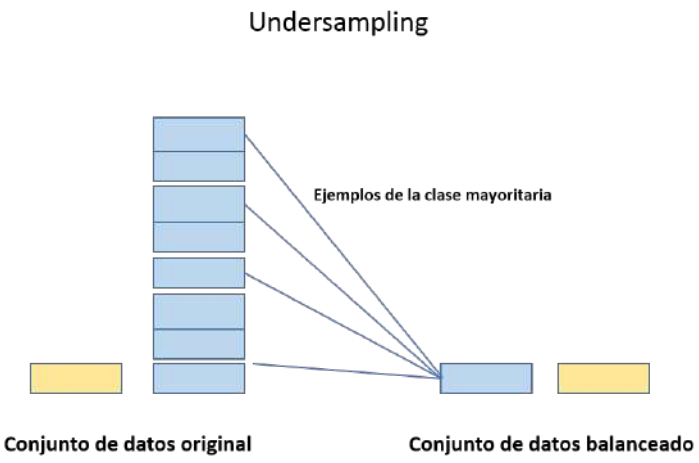


Figura 3.3: Undersampling.

- **Oversampling**. La clase minoritaria se sobremuestrea o complementa con muestras adicionales adquiridas de otro conjunto de datos para lograr una distribución equilibrada, como se observa en la Figura 3.4.

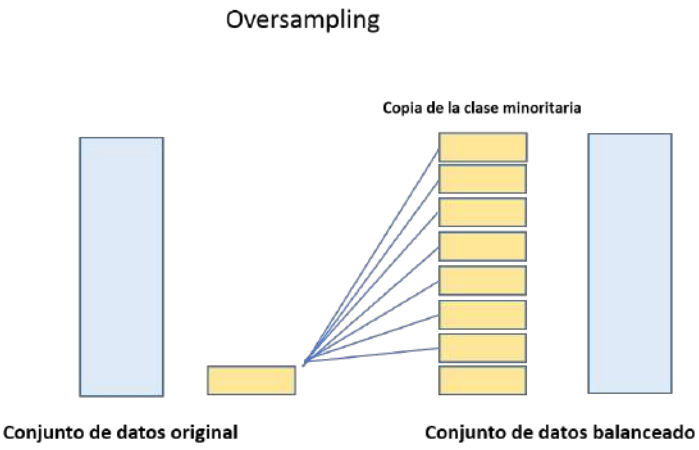


Figura 3.4: Oversampling.

3.3.4. Clasificadores y su configuración

Una vez que se hayan preparado los datos de entrada, el siguiente paso es crear la arquitectura de los clasificadores con los que se va a experimentar. Para ello es necesario establecer las configuraciones que se presentan en [37, 53, 97], esto con el fin de buscar la replicación de las propuestas para la clasificación de NFR que son CNN y Naïve Bayes, además se propone probar en una SVM para determinar el porque una CNN es mejor en este tipo de clasificación. A continuación se muestran las configuraciones para cada uno de los clasificadores.

3.3.4.1. SVM

La mejor configuración con respecto a la exactitud de la SVM girará entorno a la propuesta de una búsqueda de malla, la cual buscará los siguientes parámetros de configuración:

- ***Kernel***. Especifica el tipo de núcleo que se utilizará en el algoritmo. Debe ser uno de "lineal", "poli", "rbf", "sigmoide".
- ***C***. Que es un parámetro de regularización positivo. El cual tiene como valor predeterminado de 1.0, por lo que se necesita encontrar el mejor parámetro. La búsqueda de malla recibirá un arreglo de 7 números entre 0.001 y 1000, para está prueba.
- ***Gamma***. Es el coeficiente de kernel para "lineal", "poli", "rbf", "sigmoide", al igual que C la búsqueda de malla recibirá un arreglo de 7 números entre 0.00001 y 10, para está prueba.

3.3.4.2. Naïve Bayes

Los métodos Naïve Bayes son un conjunto de algoritmos de aprendizaje supervisado basados en la aplicación del teorema de Bayes de independencia condicional entre cada par de características dado el valor de la variable de clase. Estos métodos son tomados de la biblioteca que proporciona Scikit-learn.

- **GaussianNB**. Este método implementa el algoritmo gaussiano de Naïve Bayes, por lo que se supone que la probabilidad de las características es gaussiana. La configuración de dicho método recibe la matriz X, la matriz Y y se aplica una validación cruzada de 10.
- **MultinomialNB**. Implementa el algoritmo Bayes ingenuo para datos distribuidos multinomialmente, y es una de las dos variantes Naïve Bayes clásicas utilizadas en la clasificación de texto (donde los datos se representan típicamente como recuentos de vectores de palabras. La distribución está parametrizada por vectores. La configuración es la misma que en el método anterior. Cabe mencionar que éste método es utilizado por [37].

3.3.4.3. CNN

Las configuraciones de la arquitectura de la CNN presentadas por [37, 97], se basaron en el modelo de clasificación propuesto por [58]; para el caso de [53] no dejan ver cual fue la inspiración de su arquitectura, por lo que para esta tesis se pretende replicar los 3 casos, además de proponer una configuración de la arquitectura en base a los resultados de las pruebas que se obtengan de dichos trabajos. Cabe mencionar que se mostrarán a más detalle en el apartado de Experimentación y Resultados de esta tesis. No obstante,

en la Figura 3.5 se puede observar una configuración de la arquitectura general fusionada de los casos mencionados.

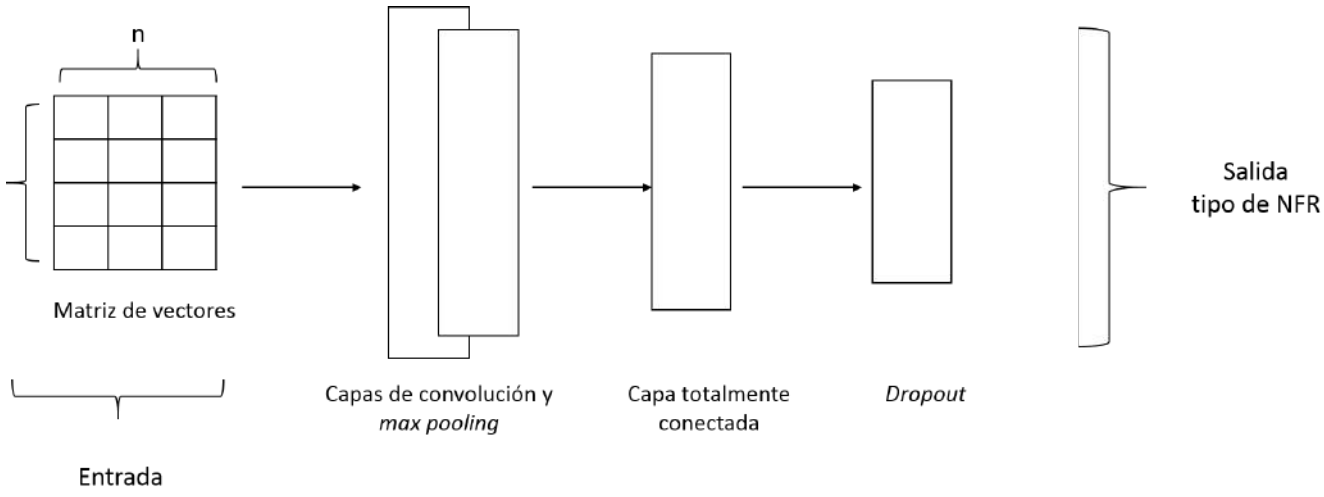


Figura 3.5: Configuración de arquitectura basada en [37, 53, 58, 97].

3.3.5. Optimización de hiperparámetros

La optimización de hiperparámetros consiste en buscar posibles variables de configuración para un algoritmo de entrenamiento con el fin de encontrar un conjunto de variables que le permita al algoritmo lograr resultados más deseables. Por ello se plantea implementar las búsquedas que se muestran a continuación para mejorar el rendimiento de los clasificadores.

- **Búsqueda de malla**. Lo que hace este método es que toma listas de parámetros y ejecuta el modelo con cada combinación que puede encontrar. Es la forma más completa pero también la forma más costosa computacionalmente.

- **Búsqueda aleatoria**. Esta búsqueda toma combinaciones aleatorias de parámetros. Para este caso se pretende aplicar la que ofrece la biblioteca de *Keras*: *KerasClassifier*, la cual tiene una clase llamada *RandomizedSearchCV* que implementa la búsqueda aleatoria con validación cruzada. La validación cruzada es una forma de validar el modelo y tomar todo el conjunto de datos y separarlo en múltiples conjuntos de datos de entrenamiento y pruebas.

3.3.6. Métricas de evaluación para rendimiento de los clasificadores

Como se mencionó al final del Capítulo anterior se utilizarán las mismas métricas de evaluación que utilizaron [37, 53, 97], ya que además de establecer la mejora del trabajo realizado, miden la rendimiento del modelo con respecto a las predicciones correctas que realiza este. A continuación se muestran brevemente:

- **Matriz de confusión**. Sirve para evaluar la precisión de una clasificación
- **Exactitud**. Es el porcentaje total de casos clasificados correctamente. Es decir, es la relación entre las predicciones correctas y el número total de predicciones.
- ***Recall***. Es el número de datos identificados correctamente como positivos del total de positivos verdaderos.
- **Puntaje *F1***. Se puede interpretar como un promedio ponderado de la precisión y el *recall*, donde un puntaje F1 alcanza su mejor valor en 1 y el peor puntaje en 0. La contribución relativa de precisión y *recall* al puntaje F1 es igual.
- **Precisión**. La precisión es la relación entre las predicciones correctas y el número total de predicciones correctas previstas. Esto mide la precisión del clasificador a la hora de predecir casos positivos.

3.4. Conclusiones

Con base en la investigación realizada en el estado del arte y la optimización de la experimentación de esta tesis, se pudieron encontrar agentes que a simple vista ofrecen mejorar los resultados de dicha experimentación por ello en el siguiente capítulo, se mostrará la implementación de la metodología que se construyó.

Capítulo 4

Experimentación y resultados

En este capítulo se presenta y analiza la experimentación y resultados sobre 3 modelos de clasificación de requisitos no funcionales de acuerdo a la comparativa que se desea establecer con [37, 53, 97]. Es importante dejar en claro que aunque los 3 autores utilizaron CNN's con diferentes arquitecturas, [37] implementó Naïve Bayes como parte de su investigación, es por ello que en está tesis también se presenta la experimentación sobre dicho modelo. Para cada problema de clasificación se definió lo siguiente:

- **Experimento 1.** Modelo base para Naïve Bayes de acuerdo con [37].
- **Experimento 2.** Modelo base para SVM.
- **Experimento 3.** Modelos de CNN con respecto a 3 matrices de incrustaciones de palabras previamente entrenadas.

Para los 3 experimentos, se incluyen la propuesta de eliminar *stopwords*, mencionado en el capítulo 2, antes de pasar por el extractor de características, así como el proceso de reducir palabras a su forma normal o raíz (*lematización*).También se busca resaltar la importancia del preprocesamiento del texto antes de integrarlo al modelo de clasificación. Además, se presenta la implementación de dos estrategias de muestreo como son: *Random Under sampling* (RUS, por sus siglas en inglés) y *Random Over sammpling* (ROS, por sus siglas en inglés), proporcionadas por la biblioteca de *Scikit-Learn*.

4.1. Experimento 1: Naïve Bayes

El primer clasificador a implementar es el de Naïve Bayes Multinomial (NBM, por sus siglas en inglés), la configuración base de este modelo parte de [37].

4.1.1. Resultados y comparativa

El Cuadro 4.1, concentra los resultados obtenidos en los experimentos realizados en esta tesis y los de [37] para el modelo NBM. A continuación se interpreta cada uno de ellos:

- **Cuadro 4.1a [37].** Se aprecia un promedio de *recall* del 0.0491, *precision* del 0.503 y una medida *F1* del 0.497. El modelo no clasificó las clases *L* y *MN*; la autora menciona que el motivo por el que el modelo no las identificó fue porque representan la minoría de ejemplos por clase en el conjunto de datos. Sin embargo, menciona que se lograron con puntuaciones aceptables para *PE, SE y US*.

- **Cuadro 4.1b.** El resultado en esta figura comprende de un promedio de *recall* del 0.632, *precision* del 0.720 y una medida *F1* del 0.649. A se puede apreciar que el modelo identificó de manera óptima incluso a las clases que menciona [37], sin embargo, llama la atención que *US* no fuera detectada por el clasificador.

- **Cuadro 4.1c.** Los promedios obtenidos fueron de *recall* del 0.607, *precision* del 0.807 y una medida *F1* del 0.645. Se puede observar que todos los NFR utilizados en la implementación de este modelo fueron identificados por el clasificador. Para obtener dichos resultados, tomando en cuenta que *L* y *MN* son clases con muy pocos ejemplos, tal como lo menciona [37] en sus resultados, su ubicación no está distribuida uniformemente dentro del conjunto de datos. Esa situación conlleva a que, cuando se realice la partición de los datos de entrenamiento y de prueba, algunos o ningún ejemplo de esas clases exista dentro de la partición realizada dentro el conjunto de prueba. Para dar una solución a esto se optó por utilizar uno de los argumentos que proporciona *ScikitLearn*, durante la partición de datos: *random_ state*, este generador de números enteros aleatorios, controla la combinación aplicada a los datos antes de realizar la partición; el cual al proporcionarle un valor fijo garantiza que se genere la misma secuencia de números aleatorios. Para obtener dichos resultados, se aplicó un *random_ state* de 21, se probó un rango de 13 a 25, para elegir dicho rango se tomó el número de ejemplos de la clase minoritaria.

Métrica	A	L	LF	MN	0	PE	SC	SE	US	Promedio
Recall	0.250	0.0	0.525	0.0	0.805	0.780	0.300	0.862	0.893	0.491
Precision	0.367	0.0	0.867	0.0	0.603	0.778	0.500	0.825	0.587	0.503
F1	0.297	0.0	0.654	0.0	0.690	0.779	0.375	0.843	0.708	0.497

(a) Resultados obtenidos de [37]

Métrica	A	L	LF	MN	0	PE	SC	SE	US	Promedio
Recall	0.900	0.800	0.800	0.667	0.857	0.429	0.571	0.667	0.0	**0.632**
Precision	0.643	0.500	0.923	1.00	0.750	1.00	0.667	1.00	0.0	**0.720**
F1	0.750	0.615	0.857	0.800	0.800	0.600	0.615	0.800	0.0	**0.649**

(b) Resultados obtenidos para el modelo de NBM

Métrica	A	L	LF	MN	0	PE	SC	SE	US	Promedio
Recall	0.778	0.800	0.688	0.667	0.833	0.500	0.615	0.250	0.33	**0.607**
Precision	0.438	0.800	0.647	1.00	0.714	1.00	0.667	1.00	1.00	**0.807**
F1	0.560	0.800	0.667	0.800	0.769	0.667	0.640	0.400	0.500	**0.645**

(c) Resultados obtenidos para el model NBM aplicando random_state en la partición de datos de entrenamiento y test

Cuadro 4.1: Resultados y comparativa del modelo *NBM* con [37].

4.1.2. Conclusiones

Principalmente se puede percibir, con base a los resultados obtenidos, las propuestas de eliminación de *stopwords*, definición de la lematización de las palabras dentro del texto y la partición balanceada de datos aportan una clasificación con un mejor rendimiento de las métricas de evaluación del modelo NBM. Es importante resaltar que la mejoría se debe principalmente al balancear el número de ejemplos, debido a que habrá ejemplos de las clases mencionados tanto en el conjunto de entrenamiento como en el de prueba, por lo que se podrá hacer una clasificación de estas y no etiquetarlas con la clase mayoritaria, es decir de forma errónea o que no la identifique. Aunque hasta este ejemplo no se ha obtenido una *recall* y *F1* con un promedio mayor a 0.800, la mejoría se observa entre de los promedios de 4.1a vs 4.1c con una diferencia del 11.6 % para *recall*, 30.4 % para *precision* y un 14.8 % para *F1*.

4.2. Experimento 2: SVM

La experimentación con SVM estaba considerada sólo para el modelo de Clasificación de Vectores (*SVC*, por sus siglas en inglés), pero también se presenta la implementación del modelo de clasificación de vectores de soporte lineal (*LinearSVC*, por sus siglas en inglés). Para ambos se muestra el mejor resultado encontrado por la Búsqueda de Malla que proporciona *Scikitlearn*, esta toma valores de un diccionario de parámetros para poder realizar la búsqueda de la mejor configuración. Los parámetros para cada clasificador son los siguientes:

- **SVC** Diccionario de parámetros. *C*: 0.001, 0.01, 0.1, 1, 10, 100, 1000, *gamma*: 0.00001,0.0001,0.001,0.01,0.1,1,10 y *kernel*: 'linear', 'poly', 'rbf'.

- **LinearSVC** Diccionario de parámetros.*C*:0.001, 0.01, 0.1, 1, 10, 100, 1000, *multi_class*:'ovr', 'crammer_singer'.

4.2.1. Resultados y comparativa

El cuadro 4.2 muestra los resultados de la experimentación sobre estos dos modelos de SVM, aunque se planeaba solo considerar un modelo (SVC), se decidió comparar con otro, en específico el de LinearSVC, ya que éste cuenta con parámetros que permiten realizar una clasificación multiclase, y esta es la naturaleza del conjunto de datos sobre el que se está trabajando. Los mejores parámetros fueron los siguientes:

- **SVC**. 'C': 100, 'gamma': 0.01, 'kernel': 'rbf'

- **LinearSVC**. 'C': 0.1, 'multi_class': 'crammer_singer'.

Al observar los resultados de las métricas de evaluación de ambos modelos se encuentra un mejor promedio de *recall* y *F1* en el Cuadro 4.2b que en el Cuadro 4.2a, esto concuerda con la documentación encontrada sobre el modelo LinearSVC, en la que dice que rara vez conduce a una mejor precisión y resulta más costosa computacionalmente. Por otro lado, se muestra en el Cuadro 4.2a con una *precision* del 0.771.

4.2.2. Conclusiones

De acuerdo a lo que se encontró en la documentación de estos modelos, *LinearSVC* tendría un mejor rendimiento con grandes conjuntos de datos. Cabe mencionar que los NFR contiene 359 ejemplos, por lo que ese sería uno de los motivos para obtener los resultados observados en el Cuadro 4.2.

Métrica	A	L	LF	MN	0	PE	SC	SE	US	Promedio
Recall	0.733	0.714	0.571	0.750	0.778	0.667	0.846	1.00	0.500	0.729
Precision	0.846	0.714	0.727	1.00	0.500	0.667	0.733	0.750	1.00	**0.771**
F1	0.786	0.714	0.640	0.857	0.609	0.667	0.786	0.857	0.667	0.731

(a) Resultados del modelo SVC

Métrica	A	L	LF	MN	0	PE	SC	SE	US	Promedio
Recall	0.733	0.857	0.571	0.750	0.778	1.00	0.692	1.00	0.750	**0.792**
Precision	0.846	0.857	0.800	0.750	0.500	0.500	0.900	0.750	0.750	0.739
F1	0.786	0.857	0.667	0.750	0.609	0.667	0.783	0.857	0.750	**0.747**

(b) Resultados del modelo LinearSVC

Cuadro 4.2: Resultados y comparativa de los modelos base de *SVC y LinearSVC*.

4.3. Experimento 3: CNN

En esta sección se tiene por objeto explorar, identificar y mejorar las bases de preprocesamiento y configuración de CNN propuestas por [37, 53, 97]. El punto clave para la clasificación de NFR radica en que su naturaleza es multiclase, por lo que se determinó, con base a los experimentos previos, construir modelos combinando las mejoras que se hicieron en cada experimento anterior y de ser posible adaptarlas en este modelo de clasificación con CNN para tratar de aumentar las métricas de evaluación.

- [97], no utiliza el conjunto de datos *Promise*, y realiza clasificación binaria. Su aporte ayudará a identificar una configuración de la CNN y entender cual es la función de cada capa que utilizó.

- [53], realiza clasificación multiclase, si dispone el conjunto de datos *Promise*, pero utiliza las 12 clases, es decir tanto Requisitos funcionales así como los No funcionales. Utilizó el modelo embebido de fastText.

- [37], realiza clasificación multiclase, también ocupa *Promise* e implementa experimentación sobre NFR, en especial sobre 9 clases, ya que las otras dos que pertenecen a ese tipo de requisitos, cuentan con pocos ejemplos, lo que impide el uso de las estrategias de muestreo; por lo que se requiere de al menos dos ejemplos para llevarla acabo. No hace referencia a que tipo de estrategia obtuvo los resultados que presenta, así que para los efectos de esta tesis se probaron con ambos ROS y RUS, sin embargo, en está sección solo se presenta el más óptimo que fue ROS, para ver los resultados de la experimentación con RUS ver el apartado en el índice. Utilizó los modelos embebidos de fastText y Word2Vec. También menciona el uso una matriz

embebida aleatoria, en donde se aprecian que los resultados son bajos, por lo que se determinó utilizar otra matriz embebida común en el NLP: Glove.

La experimentación busca determinar la influencia que tiene el preprocesamiento del texto, la vectorización de los datos, la estrategia de muestreo ROS, la implementación de las matrices embebidas pre entrenadas de Word2Vec, fastText y Glove en la capa embebida de la CNN y la hiperparametrización de sus capas consecuentes.

4.3.1. Resultados y comparativa

Los resultados mostrados a continuación se presentan de manera gradual; es decir, se señala la forma en la que al agregar las técnicas propuestas a una arquitectura base de CNN, el rendimiento de clasificación va mejorando. La Figura 4.1, muestra las capas con la que está propuesta en la red para esta tesis.

Figura 4.1: Arquitectura base de la CNN, para esta experimentación.

En esta experimentación se quiere mostrar como influye cada una de estas implementaciones en la mejora de la clasificación de los NFR, los puntos a considerar son los siguientes:

- **Preprocesamiento.** Incluir o no una limpieza previa del conjunto de datos Promise, además de aplicar lematización a las palabras. También se prueba con 3 tipos de vectorizadores: *TF-IDF*, *Tokenizer* y *CountVectorizer*.

- **Hiperparametrización en la partición de los datos**.

- **Arquitectura de la CNN**. Iniciando con la arquitectura base mostrada anteriormente incluyendo o no los pesos de las matrices preentrenadas.

El Cuadro 4.3 contiene la concentración de resultados obtenidos aplicando lo antes mencionado. En primer lugar, en el Cuadro 4.3a, en donde se logró un promedio de *Recall* de 0.11, *Precision* de 0.01 y *F1* con un 0.02 para la prueba con el vectorizador TF-IDF; por otro lado con el vectorizador Tokenizer se obtuvo un promedio de 0.44 para *Recall*, un 0.46 para *Precision* y un 0.44 para la métrica *F1*. En cambio, con el vectorizador CountVectorizer, se logra ver que el entrenamiento de la red con esa herramienta ganó un *Recall*

con 0.59, *Precision* con 0.53 y *F1* con 0.54. La segunda parte de la experimentación, es integrando la limpieza previa y lematización, además de hacer la hiperparametrización en la partición de los datos; por lo que los promedios resultantes de esta etapa son los que se muestran en el Cuadro 4.3b. Se observa un aumento en el *accuracy*, sin embargo, al dirigir la atención en el promedio las métricas se examina que al aplicar dichas propuestas se encuentra una notable mejora para el caso del vectorizador *CountVectorizer*, en donde *Recall* tiene un 0.70, *Precision* con un 0.69 y *F1* con un 0.67. Por lo tanto, el clasificador ha mostrado un comportamiento particular sobre *CountVectorizer*, tanto en esta prueba como en la anterior. Por último, se agrega una propuesta más que es entrenar el modelo con los pesos de la matriz pre entrenada *Word2vec*, ya que está fue un común denominador entre las propuestas de [37, 53, 97]. Los promedios obtenidos se pueden ver en el Cuadro 4.3c, *Recall* con 0.72, *Precision* con 0.74 y *F1* con 0.72; por lo que se pudo observar un incremento en los promedios utilizando dicha matriz de pesos.

Prueba	Preprocesamiento		Hiperparametrización en partición de datos	Pesos de matriz preentrenada	Acc	Loss	Promedio		
	Limpieza	Vectorizador					Recall	Precision	F1
1	No	TF-IDF	No	No	12.5	2.04	0.11	0.01	0.02
2	No	Tokenizer	No	No	47.22	6.70	0.44	0.46	0.44
3	No	CountVectorizer	No	No	**62.50**	**3.70**	**0.59**	**0.53**	**0.54**

(a) Resultados de modelo base

Prueba	Preprocesamiento		Hiperparametrización en partición de datos	Pesos de matriz preentrenada	Acc	Loss	Promedio		
	Limpieza	Vectorizador					Recall	Precision	F1
1	Si	TF-IDF	Si	No	19.44	2.06	0.11	0.02	0.04
2	Si	Tokenizer	Si	No	50.00	3.54	0.41	0.45	0.39
3	Si	CountVectorizer	Si	No	**69.44**	**2.41**	**0.70**	**0.69**	**0.67**

(b) Resultados de modelo base con integración de dos propuestas

Prueba	Preprocesamiento		Hiperparametrización en partición de datos	Pesos de matriz preentrenada	Acc	Loss	Promedio		
	Limpieza	Vectorizador					Recall	Precision	F1
1	Si	TF-IDF	Si	Si	19.44	2.07	0.11	0.02	0.04
2	Si	Tokenizer	Si	Si	58.33	2.49	0.42	0.41	0.40
3	Si	CountVectorizer	Si	Si	**73.61**	**2.46**	**0.72**	**0.74**	**0.72**

(c) Resultados de modelo base con integración de tres propuestas

Cuadro 4.3: Resultados obtenidos de la experimentación con integración progresiva de propuestas para la mejora de la clasificación de NFR utilizando NFR.

Durante las experimentaciones, se pudo observar como han mejorado los resultados del modelo de clasificación con respecto al estado del arte implementando las propuestas en el preprocesamiento, la hiperparametrización durante la partición de datos y el agregar una matriz de pesos preentrenados. Debido a esas mejoras, se ha decidido utilizar la configuración de la Tabla 4.3 correspondiente al vectorizador countvectorizer, al verificar que los otros vectorizadores no obtuvieron una mejora en las métricas de evaluación; como base para las siguientes experimentaciones.

4.3.1.1. CNN con la implementación de matrices preentrenadas y estrategia de muestreo ROS

Para este experimento se añadió una capa de *Dropout*, después de la capa embebida, así como otro grupo de una convolución seguida de una capa de *MaxPooling*. La arquitectura propuesta para esta experimentación se muestra en la Figura 4.2.

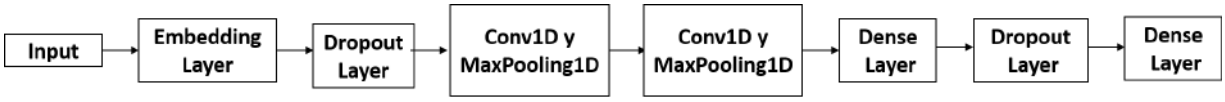


Figura 4.2: Arquitectura propuesta.

En el Cuadro 4.4 se pueden observar los resultados de las métricas obtenidas al implementar cada una de las diferentes matrices embebidas preentrenadas. El preprocesamiento del texto, la estrategia de muestreo ROS y la nueva arquitectura, fueron empleadas para este entrenamiento que se realizó con 100 épocas. Sin embargo, fue la vectorización de datos el factor importante para mejorar los resultados. Para dicha esta se utilizó la biblioteca de *CountVectorizer*, la cual además de obtener propiedades como eliminar *stopwords*, calcular la frecuencia de palabras entre otras, también posee el argumento de *ngram_range*. Este argumento determina el límite inferior y superior del rango de n valores para diferentes palabras, llamado *n-gramas* [71, 72].

De ahí que dicho argumento resultó importante y fue considerado para obtener una vectorización de palabras que contribuye a darle sentido a cada uno de los requisitos de acuerdo a su contexto. El *ngram_range* que permitió que se obtuvieran los resultados mostrados en el Cuadro 4.4 fue de (1, 4) para fastText y Word2Vec y de (1, 2) para Glove. Es importante resaltar que se realizaron pruebas sin *ngram_range* y con rangos de (1,1),(1,2),(1,3) y (1,4), para cada experimentación con las matrices embebidas seleccionadas. Por lo que se hicieron 5 pruebas por cada script.

Matriz embebida	LOSS	ACC.	Métrica	A	L	LF	MN	0	PE	SC	SE	US	Promedio
Word2vec	0.86	0.88	Recall	0.67	0.93	0.85	1.00	0.78	1.00	0.87	0.83	1.00	**0.88**
			Precision	0.91	0.81	0.92	0.80	0.88	1.00	0.83	0.91	1.00	**0.90**
			F1	0.77	0.87	0.88	0.89	0.82	1.00	0.85	0.87	1.00	**0.88**
FastText	0.84	0.83	Recall	0.67	0.93	0.62	1.00	0.67	1.00	0.78	1.00	0.83	**0.83**
			Precision	0.77	0.76	1.00	0.73	0.75	1.00	0.82	0.86	1.00	**0.85**
			F1	0.71	0.84	0.76	0.84	0.71	1.00	0.80	0.92	0.91	**0.83**
Glove	0.67	0.79	Recall	0.87	1.00	0.62	0.81	0.78	0.86	0.48	1.00	1.00	**0.82**
			Precision	0.93	0.88	0.80	0.76	0.70	0.60	0.92	0.71	0.80	**0.79**
			F1	0.90	0.93	0.70	0.79	0.74	0.71	0.63	0.83	0.89	**0.79**

Cuadro 4.4: Resultados de la CNN con 100 épocas para la clasificación de NFR utilizando 3 tipos de matrices embebidas.

Realizando la comparativa en el Cuadro 4.5, se puede observar como la propuesta base de esta tesis con 100 épocas reflejó una incremento notable con respecto a los resultados óptimos obtenidos por [37] con 140 épocas. Para determinar si realmente una mejora con respecto a las configuraciones iniciales se ha propuesto realizar un análisis estadístico en el siguiente apartado.

Matriz embebida	Tamaño de filtro	LOSS	ACC.	Métrica	Promedio
Random	{1,2}	0.909	0.731	Recall	0.657
				Precision	0.656
				F1	0.655
Word2vec	{1,2}	0.611	0.804	Recall	0.750
				Precision	0.794
				F1	0.767
FastText	{1,2,3}	0.632	0.792	Recall	0.732
				Precision	0.759
				F1	0.757

(a) Resultados obtenidos por [37] para 140 épocas

Matriz embebida	Tamaño de filtro	LOSS	ACC.	Métrica	Promedio
Glove	{1,2}	0.67	0.79	Recall	**0.82**
				Precision	**0.79**
				F1	**0.79**
Word2vec	{1,2}	0.86	0.88	Recall	**0.88**
				Precision	**0.90**
				F1	**0.88**
FastText	{1,2}	0.84	0.83	Recall	**0.83**
				Precision	**0.85**
				F1	**0.83**

(b) Resultados obtenidos en está tesis para 100 épocas como corrida inicial para arquitectura base

Cuadro 4.5: Resultados de las métricas obtenidas del preprocesamiento y arquitectura óptimos de [37] vs. resultados del preprocesamiento y arquitectura base propuestos en esta tesis.

4.3.2. Evaluación de las propuestas implementadas al modelo de la CNN con validación cruzada *k-fold* mediante el análisis de varianza

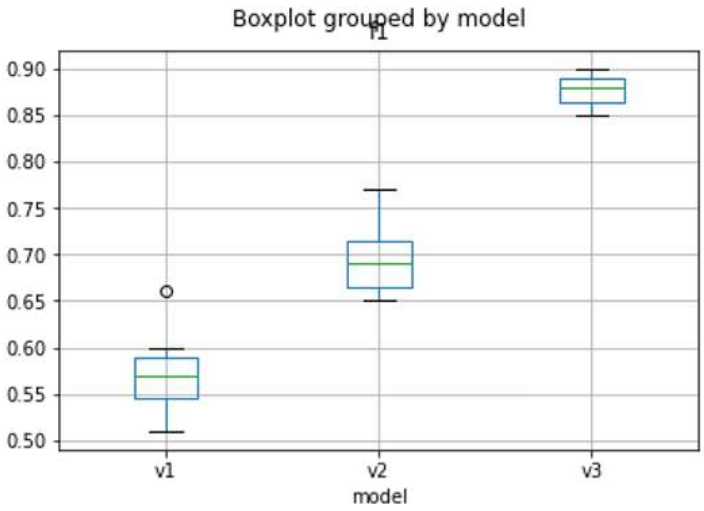
Para evaluar si la implementación de de las propuestas realizadas en esta tesis tuvieron una mejora significativa se determinó realizar el análisis de varianza (ANOVA, por sus siglas en inglés). Este análisis se encarga de probar que existe una diferencia general entre los grupos que se están estudiando. El Cuadro 4.6 presenta tres versiones y las propuestas que se implementaron en su experimentación. Es importante mencionar que la matriz preentrenada Word2vec se seleccionó ya que en las pruebas anteriores reflejó el mejor promedio de las métricas *Recall*, *Precision* y *F1*, así como la arquitectura base que se aprecia en la Figura 4.1 y la propuesta en la Figura 4.2.

Al realizar el análisis es necesario tomar un conjunto de promedios resultantes de la misma versión del modelo, por ello se hizo el entrenamiento con validación cruzada *k-fold* con k=10. Los promedios utilizados provienen de la métrica *F1* ya que representa el promedio entre *Recall* y *Precision*. La bondad de *F1* es útil cuando se tiene una distribución desigual en las clases, siendo este el caso del conjunto de datos que se está ocupando, por eso la elección de esa métrica.

Versión	Pre procesamiento	Vectorizador	Hiperparametrización en partición de datos	Estrategia de muestreo	Arquitectura	Matriz Preentrenada
v1	✗	CountVectorizer	✗	✗	Base	✗
v2	✓	CountVectorizer	✓	✗	Propuesta	✗
v3	✓	CountVectorizer	✓	✓	Propuesta	Word2vec

Cuadro 4.6: Propuestas implementadas a tres versiones de la CNN.

Primero para realizar el cálculo de ANOVA, se recopilaron los resultados promedio de la métrica F1 de cada despliegue obtenido del entrenamiento con validación cruzada de cada versión. Al obtener los 30 resultados, se hizo un un análisis de la distribución de dichos datos, en donde se observa a simple vista en la Figura 4.3a que para la versión 1 se refleja un valor atípico que varia del promedio de los datos. Por ejemplo, el contener un promedio de 0.66 (ver Cuadro 4.3b, los promedios sombreados resaltan el máximo promedio alcanzado) para el *fold* número 4; para el caso de la versión 2 el modelo reflejo un incremento en su promedio y se comportó arriba del 65 % hasta un 77 %. Sin embargo, su promedio general llegó a un 66 %; en constaste el modelo en su versión 3 sus promedios fueron del 85 % hasta un 90 %, teniendo un promedio general del 88 %. A simple vista se puede observar que si se marca la diferencia entre las versiones experimentadas.



(a) Diagrama de caja de las promedios de las métricas F1 pertenecientes a las 3 versiones del modelo

MODELO	F1									
	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6	Fold7	Fold8	Fold9	Fold10
v1	0.54	0.51	0.60	**0.66**	0.56	0.58	0.56	0.59	0.59	**0.51**
v2	0.69	0.69	0.65	0.72	0.68	0.66	0.73	0.70	**0.77**	0.66
v3	**0.90**	0.89	0.89	0.85	0.87	0.86	0.85	0.88	0.88	**0.90**

(b) Tabla de promedios de la métrica F1 para las 3 versiones del modelo

Figura 4.3: Cuadro de promedios de la métrica F1 para las 3 versiones del modelo y diagrama de distribución de dichos datos.

Ahora bien, como ya se mencionó el propósito de está propuesta fue el evaluar la diferencia entre cada versión del modelo y las propuestas implementadas con respecto a la métrica *f1*. El promedio general de la métrica *f1*, como se muestra en el Cuadro 4.7a, fue de 0.714 para 30 muestras (N) y su intervalo de confidencialidad (*CI*, por sus siglas en inglés) al 95 % fue de (0.6645,0.7635). Los promedios específicos para los grupos por versión del modelo con *N=10* cada uno, fueron los siguientes (ver Cuadro 4.7b): la versión 1 con un promedio de 0.570 y un CI al 95 % de (0.5378, 0.6022), para el caso de la versión 2 se obtuvo un promedio de 0.695 con un CI al 95 % de (0.6686, 0.7214), por otro lado la versión 3 arrojó un promedio de 0.877 y un CI al 95 % de (0.8635, 0.8905).

Variable	N	Promedio	SD	SE	95% Conf	Intervalo
f1	30	0.714	0.1327	0.0242	0.6645	0.7635

(a) Cuadro de resumen para el total de muestras con respecto a los promedios de la métrica F1.

Modelo	N	Promedio	SD	SE	95% Conf	Intervalo
v1	10	0.570	0.0450	0.0142	0.5378	0.6022
v2	10	0.695	0.0369	0.0117	0.6686	0.7214
v3	10	0.877	0.0189	0.0060	0.8635	0.8905

(b) Cuadro de promedios F1 de modelos individuales.

Cuadro 4.7: Promedios general y particular de los modelos analizados con respecto a F1.

Se tomó como referencia el valor de significancia estándar de $\alpha = 0.05$, ahora bien al calcular ANOVA para las muestras presentadas se puede observar en el Cuadro 4.8 que hay una diferencia estadísticamente significativa entre los modelos presentados debido a que se obtuvo a p=1.148591e-16. Por lo tanto, con los datos observados, hay evidencia suficiente para asumir una diferencia significativa entre los modelos expuestos en esta evaluación. Cabe mencionar que se verificaron los supuestos de la prueba haciendo uso de la prueba no perimétrica Kruskall-Wallis.

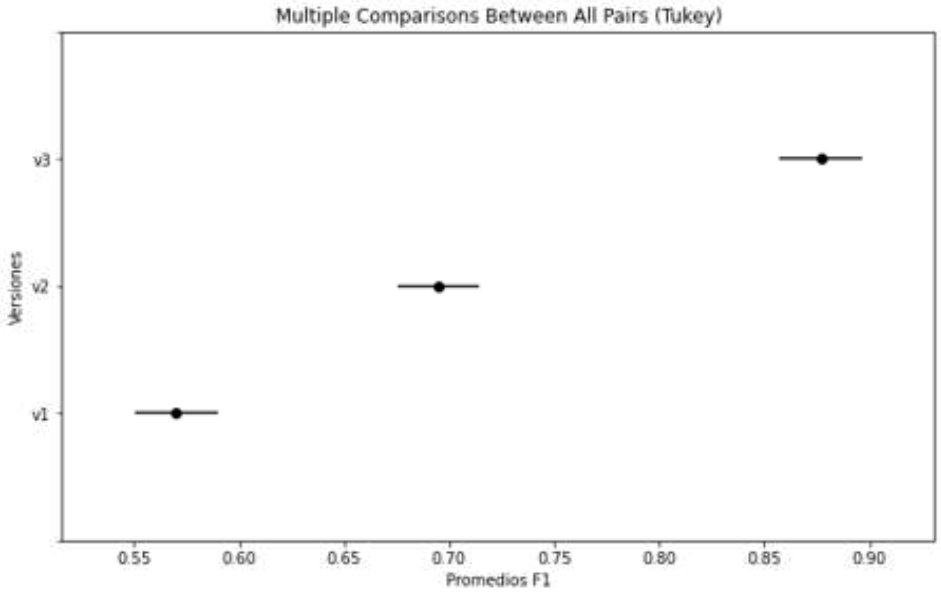
	Grados de libertad	Suma de cuadrados	Promedio de cuadrados	F	P
modelos	2.0	0.47666	0.238330	191.173797	1.148591e-16
Residual	27.0	0.03366	0.001247		

Cuadro 4.8: Resultado de ANOVA para los modelos con respecto a F1.

Además se realizo la prueba de diferencia honestamente significativa de *Tukey (HSD de Tukey, por sus siglas en inglés)*, se utiliza para probar las diferencias entre las medias de la muestra en cuanto a significancia, probando las diferencias por pares. Por lo que en el Cuadro 4.4a, se observa que entre todas las versiones de experimentos probados la diferencia fue verdadera, esto es debido al valor que arroja el valor p para cada una de las combinaciones. Gráficamente se puede observar en la Figura 4.4b que entre las combinaciones entre los grupos, la evaluación presenta una diferencia estadísticamente significativa sobre las versiones.

Grupo1	Grupo2	Dif. de promedios	p-adj	Inferior	Superior	Rechazar
V1	V2	0.125	0.001	0.0859	0.1641	Verdadero
V1	V3	0.307	0.001	0.2679	0.3461	Verdadero
V2	V3	0.182	0.001	0.1429	0.2211	Verdadero

(a) Cuadro de múltiples comparaciones de promedios Tukey HSD, menor a=0.05



(b) Gráfica Tukey de múltiples comparaciones de promedios entre pares.

Figura 4.4: Prueba post-hoc Tukey

4.3.3. Implementación del modelo con mejor rendimiento para la clasificación de NFR en una aplicación web

Se ha tomado el mejor modelo obtenido del entrenamiento con las implementaciones de las propuestas realizadas en esta tesis, con el fin de proporcionar una herramienta que auxilie a los analistas a la hora de determinar la clase a la que pertenece un requisito no funcional de acuerdo con la clasificación que proporciona el conjunto de datos. El manual y la configuración de dicha herramienta la puede encontrar en el Anexo B.2. Una muestra de dicho clasificador web se observa en la Figura 4.5, en donde se realiza la clasificación de requisitos no funcionales de un documento.

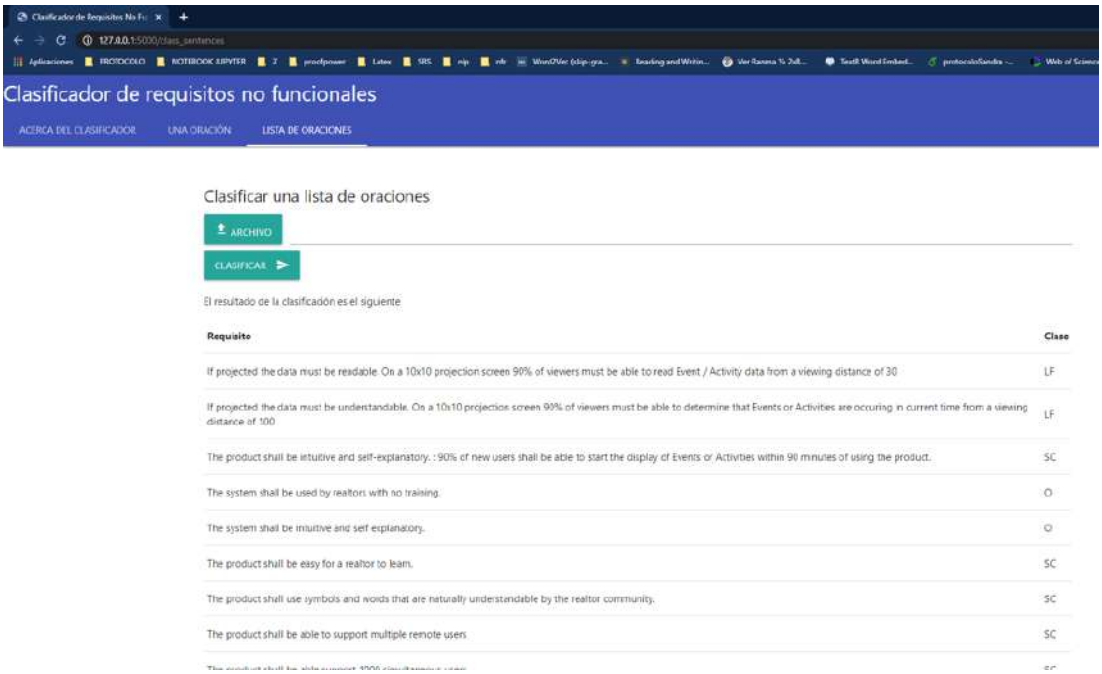


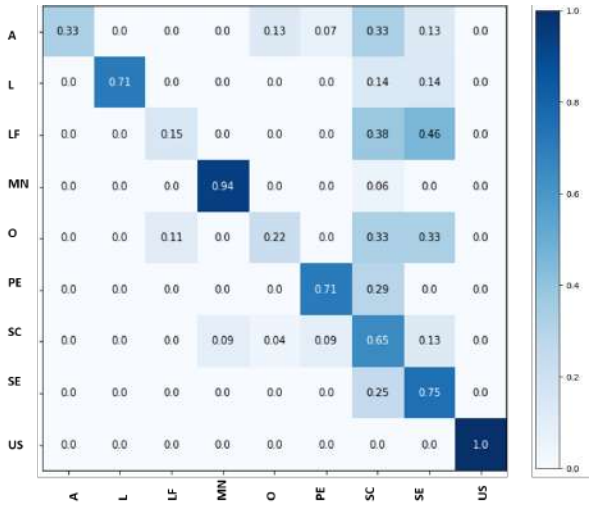
Figura 4.5: Pantalla de resultados de clasificación de una lista de oraciones/requisitos.

4.3.4. Conclusiones

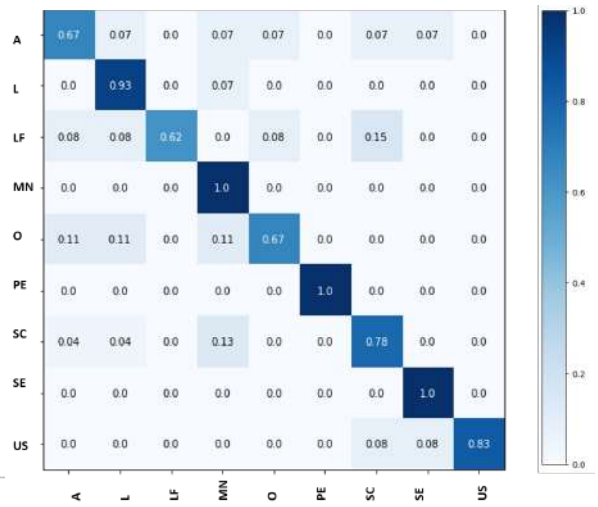
En los casos base que se presentaron al inicio de esta sección se pudo observar como el incremento de las métricas establecidas fueron creciendo, mostrando que al aplicar las propuestas como limpieza de texto, lematización de las palabras, vectorizar el texto, hacer hiperparametrización tanto de la partición de datos así como en la arquitectura de la red, además de probar como se comporta el clasificador con las 3 matrices preentrenadas propuestas, permitió mostrar una mejora significativa, para demostrar este hecho se probaron dos cosas, la evaluación del rendimiento de la red mediante el análisis de las matrices de confusión obtenidas con respecto de las matrices pre entrenadas y se realizó un ANOVA. A continuación se muestra la conclusión de dichas evaluaciones.

Para la experimentación con matrices embebidas preentrenada y la implementación de las propuestas se pudieron obtener las siguientes matrices de confusión para cada implementación con el mejor resultado del parámetro *ngram_range* y la matriz embebida correspondiente a Glove, FastText y Word2Vec. Se puede apreciar en la Figura 4.6a en donde se utilizó Glove que el modelo confunde a la clase A con la case SC al mismo porcentaje, esto implica que tiene la misma probabilidad de clasificar por cualquiera de las dos disminuyendo el rendimiento del clasificador, además, en menor cantidad confunde con otras tres clases. La clase LF clasifica más como SE; la clase O la confunde con SC y SE y en menor cantidad con LF, esto indica que estas 3 clases son las que afectan el rendimiento del modelo usando Glove. Por otro lado al usar FastText, como se aprecia en Figura 4.6b, el problema recae en la clase LF y en menor impacto en la clase A. La clase LF es confundida por el clasificador con otras cuatro clases, por lo que afecta el rendi-

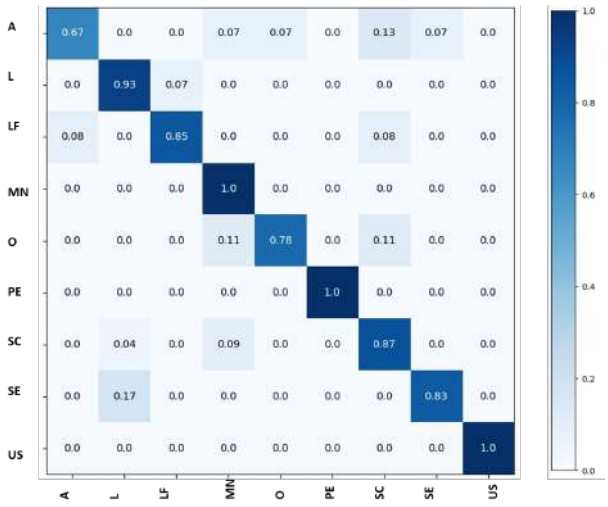
miento de este, presentando el mismo problema de clasificación sobre *LF* al igual que con Glove, en este caso de experimentación también confunde a las clase A con otras cuatro. Finalmente Word2Vec (ver Figura 4.6c), al igual que en los otros experimentos, la clase *A* es la más complicada de clasificar, la confunde también con cuatro clases. Sin embargo, su buen rendimiento se debe a que el modelo identifica mejor el resto de las clases, por lo que se observa menor confusión entre ellas, logrando con esto el mejor resultado de los 3 experimentos.



(a) Matriz de confusión de la CNN con Glove



(b) Matriz de confusión de la CNN con FastText



(c) Matriz de confusión de la CNN con Word2Vec

Figura 4.6: Matrices de confusión correspondientes a la mejor configuración de la CNN con los resultados más altos obtenidos de 100 épocas.

En el caso de la evaluación con ANOVA, se permitió, determinar de forma estadística que existe un mejoramiento del clasificador, reflejándose en el resultado de dicho análisis con un valor p=0.05. Esto significa que si se aplican herramientas y técnicas de NLP y

aprendizaje profundo se mejorará el rendimiento en la clasificación de NFR con CNN, como fue en el caso de está tesis. Ya que se implementaron las propuestas establecidas para esta experimentación y el resultado aparte de mejorarse con respecto al estado del arte [37, 53, 97] también pudo verificarse de forma estadística.

Capítulo 5

Conclusiones y Trabajo a futuro

El NLP y la IA, han sido grandes aliados para resolver problemas de clasificación de texto. Sin embargo, la solución de estos problemas generalmente están reservadas para trabajo a gran escala con grandes volúmenes de muestras de datos, lo que el lidiar con bases de datos con pocos ejemplos sugieren resultados bajos con respecto a al rendimiento de clasificación. Dicho problema ha estado presente al intentar optimizar la clasificación requisitos de software, ya que durante el análisis de estos están plasmados en un documento que frecuentemente están representados por oraciones con poco texto o información. Además, los conjuntos de datos de requisitos solo contienen de cientos a miles de documentos, lo que representa un volumen de órdenes de magnitud menor de lo que normalmente se considera necesario para el aprendizaje profundo. Además el tomar en cuenta que existen diferentes clases de requisitos, sobre todo los NFRs que son obtenidos del conjunto de datos Promise, lo que hace a la tarea de clasificación difícil de obtener resultados deseados.

Es por ello que en esta tesis, se realizó una investigación para determinar cuales eran las mejores estrategias empleadas en el estado del arte que llevaron a obtener mejores resultados en cada una de esas investigaciones, esto con el fin de unificarlas y observar si dichas estrategias juntas reflejaban una eficiencia en el rendimiento para la clasificación de NFRs mediante redes neuronales convolucionales. En principio, la experimentación inició con Naïve Bayes y SVM , ya que alguno de los autores realizó pruebas con el primer clasificador mencionado. Con respecto al segundo, surgió la curiosidad de como se comportaba el clasificador con respecto a la implementación de dichas estrategias. Se pudo observar, para la experimentación de esta tesis, como el vectorizar el conjunto de datos con incrustaciones de palabras, aplicar métodos de estrategias de muestreo Ros, hiperparametrizar la configuración al realizar la partición de datos se reflejó un aumento del promedio de las métricas Recall, Precision y F1 contra el estado del arte, ya que a diferencia de [37] se obtuvo hasta un 30 % en el incremento del promedio de las métricas mencionadas. Siendo esto una primera línea guía para aplicarlas a las CNN. Al implementar dichas propuestas a la arquitectura de la CNN, así como realizar la hiperparametrización de la misma, se optó por probar con una capa embebida con o sin pesos, mostrando así la importancia

de utilizar matrices preentrenadas que permitan mejora en cuanto a la clasificación de texto. Para determinar si realmente existió un mejoramiento en el clasificador, se realizó el análisis ANOVA que dejó ver un valor p de 0.05, por lo que, de acuerdo con significancia estándar de, si existe una mejora significativa entre los modelos presentados. De ahí que se puede decir que la aplicación de las propuestas para la clasificación de NFR con CNN dieron como resultado el mejoramiento del rendimiento del clasificador con respecto al estado del arte.

5.1. Trabajo a futuro

La investigación sobre clasificación de texto utilizando herramientas ML en SE (en específico la RE), ha tomado gran importancia, ya que se busca automatizar actividades que tengan inmersos el lenguaje humano debido a que resulta ambiguo y difícil de interpretar. Sobre todo en la RE, es de vital importancia que se definan requisitos de calidad para formar una base sólida que garantice un desarrollo de software de calidad. Por lo que se pueden proponer el desarrollo de trabajos a futuro, observados durante está investigación, que podrían motivar e interesar a la comunidad científica ya que podrían servir para mejorar el rendimiento de las redes neuronales con respecto a la clasificación de requisitos, además de ser un apoyo para los analistas de requisitos en está actividad. Algunas temas son los siguientes:

- Crear un conjunto de datos de NFR en español con más ejemplos (al menos 5000 ejemplos), al momento de empezar este trabajo se hizo una búsqueda de conjunto de datos en español y no se encontró. Esto podría servir como apoyo para la investigación de la comunidad hispana.
- Clasificar NFR utilizando la redes neuronales recurrentes, con un conjunto de datos con al menos 5000 ejemplos, para observar el rendimiento de las métricas de evaluación que se obtienen de estas redes.
- Diseñar, crear y adaptar un módulo de clasificación de requisitos a una herramienta de open source conocida dentro del SE o RE, para brindar ayuda a los analistas durante dicha tarea. Incluso, esta herramienta podría ser utilizada por la industria de desarrollo de software.

Referencias

[1] Abad, Zahra Shakeri Hossein, Oliver Karras, Parisa Ghazi, Martin Glinz, Guenther Ruhe y Kurt Schneider: *What works better? a study of classifying requirements.* En *Requirements Engineering Conference (RE), 2017 IEEE 25th International*, páginas 496–501. IEEE, 2017.

[2] Aguilar, José Alfonso, Aníbal Zaldívar-Colado, Carolina Tripp-Barba, Roberto Espinosa, Sanjay Misra y Carlos Eduardo Zurita: *A Survey About the Impact of Requirements Engineering Practice in Small-Sized Software Factories in Sinaloa, Mexico*. En *International Conference on Computational Science and Its Applications*, páginas 331–340. Springer, 2018.

[3] Albawi, S., T. A. Mohammed y S. Al-Zawi: *Understanding of a convolutional neural network.* En *2017 International Conference on Engineering and Technology (ICET)*, páginas 1–6, Aug 2017.

[4] Ali, Naveed y Richard Lai: *Requirements Engineering in Global Software Development: A Survey Study from the Perspectives of Stakeholders.* Journal of Software, vol.13:520–533, 2018.

[5] Alla, Sujatha, Pilar Pazos y Rolando DelAguila: *The Impact of Requirements Management Documentation on Software Project Outcomes in Health Care*. En *IIE Annual Conference. Proceedings*, páginas 1419–1423. Institute of Industrial and Systems Engineers (IISE), 2017.

[6] Alpaydin, Ethem: *Introduction to Machine Learning.* MIT press, 2009.

[7] Attarha, Mina y Nasser Modiri: *Focusing on the importance and the role of requirement engineering.* En *The 4th International Conference on Interaction Sciences*, páginas 181–184. IEEE, 2011.

[8] Aurum, Aybüke y Claes Wohlin: *A value-based approach in requirements engineering: explaining some of the fundamental concepts.* En *International Working Conference on Requirements Engineering: Foundation for Software Quality*, páginas 109–115. Springer, 2007.

[9] Bajpai, Vikas y Ravi Prakash Gorthi: *On non-functional requirements: A survey.* En *2012 IEEE Students' Conference on Electrical, Electronics and Computer Science*, páginas 1–4. IEEE, 2012.

[10] Becker, Christoph, Stefanie Betz, Ruzanna Chitchyan, Leticia Duboc, Steve M Easterbrook, Birgit Penzenstadler, Norbet Seyff y Colin C Venters: *Requirements: The key to sustainability.* IEEE Software, vol.33:56–65, 2016.

[11] Bengio, Yoshua: *Learning deep architectures for AI.* Foundations and trends® in Machine Learning, vol.2:1–127, 2009.

[12] Bhandare, Ashwin, Maithili Bhide, Pranav Gokhale y Rohan Chandavarkar: *Applications of convolutional neural networks.* International Journal of Computer Science and Information Technologies, vol.7:2206–2215, 2016.

[13] Bhasin, Manoj y GPS Raghava: *GPCRpred: an SVM-based method for prediction of families and subfamilies of G-protein coupled receptors.* Nucleic Acids Research, vol.32(suppl_2):W383–W389, 2004.

[14] Bhasin, Manoj y GPS Raghava: *GPCRsclass: a web tool for the classification of amine type of G-protein-coupled receptors.* Nucleic acids research, vol.33:W143–W147, 2005.

[15] Borque, P y RE Fairley: *SWEBOK v3. 0: Guide to the software engineering body of knowledge.* USA: IEEE, 2014.

[16] Bottou, Léon: *Stochastic gradient descent tricks.* En *Neural networks: Tricks of the trade*, páginas 421–436. Springer, 2012.

[17] Cambria, E. y B. White: *Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article].* IEEE Computational Intelligence Magazine, vol.9:48–57, 2014, ISSN 1556-603X.

[18] Casamayor, Agustin, Daniela Godoy y Marcelo Campo: *Identification of non-functional requirements in textual specifications: A semi-supervised learning approach.* Information and Software Technology, vol.52:436–445, 2010.

[19] Chablani, Manish: *Word2Vec (skip-gram model): PART 1 - Intuition.* url https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b, 2017. Accedido 22/02/2019.

[20] Chakraborty, Abhijit, Mrinal Baowaly, Utsho A Arefín y Ali Newaz Bahar: *The Role of Requirement Engineering in Software Development Life Cycle.* Journal of Emerging Trends in Computing and Information Sciences, vol.3:723–729, Mayo 2012.

[21] Chowdhury, Gobinda G: *Natural language processing.* Annual review of information science and technology, vol.37:51–89, 2003.

[22] Chung, Lawrence, Brian A Nixon, Eric Yu y John Mylopoulos: *Non-functional requirements in software engineering*. Springer Science & Business Media, 2012.

[23] Chung, Lawrence y Julio Cesar Sampaio do Prado Leite: *On non-functional requirements in software engineering*. En *Conceptual modeling: Foundations and applications*, páginas 363–379. Springer, 2009.

[24] Cleland-Huang, Jane, Adam Czauderna, Marek Gibiec y John Emenecker: *A machine learning approach for tracing regulatory codes to product specific requirements*. En *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volumen vol.1, páginas 155–164. IEEE, 2010.

[25] Cleland-Huang, Jane, Raffaella Settimi, Xuchang Zou y Peter Solc: *The detection and classification of non-functional requirements with application to early aspects*. En *Requirements Engineering, 14th IEEE International Conference*, páginas 39–48. IEEE, 2006.

[26] Collobert, Ronan y Jason Weston: *A unified architecture for natural language processing: Deep neural networks with multitask learning*. En *Proceedings of the 25th international conference on Machine learning*, páginas 160–167. ACM, 2008.

[27] Cruz, Pedro Ponce y Alejandro Herrera: *Inteligencia artificial con aplicaciones a la ingeniería*. Marcombo, 2011.

[28] Dalpiaz, F., A. Ferrari, X. Franch y C. Palomares: *Natural Language Processing for Requirements Engineering: The Best Is Yet to Come*. IEEE Software, vol.35:115–119, 2018, ISSN 0740-7459.

[29] Darwish, Nagy Ramadan, Ahmed Abdelaziz Mohamed y Abdelghany Salah Abdelghany: *A hybrid machine learning model for selecting suitable requirements elicitation techniques*. International Journal of Computer Science and Information Security, vol.14:1–12, 2016.

[30] De Gea, Juan M Carrillo, Joaquín Nicolás, José L Fernández Alemán, Ambrosio Toval, Christof Ebert y Aurora Vizcaíno: *Requirements engineering tools: Capabilities, survey and assessment*. Information and Software Technology, vol.54:1142–1157, 2012.

[31] Deeptimahanti, Deva Kumar y Muhammad Ali Babar: *An automated tool for generating UML models from natural language requirements*. En *Proceedings of the 2009 IEEE/ACM international conference on automated software engineering*, páginas 680–682. IEEE Computer Society, 2009.

[32] Dick, Jeremy, Elizabeth Hull y Ken Jackson: *Requirements engineering*. Springer, 2017.

[33] Duchi, John, Elad Hazan y Yoram Singer: *Adaptive subgradient methods for online learning and stochastic optimization.* Journal of Machine Learning Research, vol.12(Jul):2121–2159, 2011.

[34] Eckhardt, Jonas, Andreas Vogelsang y Daniel Méndez Fernández: *Are"nonfunctionalrequirements really non-functional? an investigation of non-functional requirements in practice.* En *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, páginas 832–842. IEEE, 2016.

[35] Eriksson, Magnus, Jürgen Börstler y Kjell Borg: *Managing requirements specifications for product lines–An approach and industry case study.* Journal of Systems and Software, vol.82:435–447, 2009.

[36] Ferrari, Alessio, Felice Dell'Orletta, Andrea Esuli, Vincenzo Gervasi y Stefania Gnesi: *Natural language requirements processing: a 4D vision.* IEEE Software, vol.34:28–35, 2017.

[37] Fong, Vivian Lin: *Software Requirements Classification Using Word Embeddings and Convolutional Neural Networks.* Tesis de Licenciatura, Cal Poly, 2018.

[38] Gea, Juan M Carrillo de, Joaquín Nicolás, José L Fernández Alemán, Ambrosio Toval, Christof Ebert y Aurora Vizcaíno: *Requirements engineering tools.* IEEE software, vol.28:86–91, 2011.

[39] Geogy, Manju y Andhe Dharani: *A scrutiny of the software requirement engineering process.* Procedia Technology, vol.25:405–410, 2016.

[40] Glinz, Martin: *On non-functional requirements.* En *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*, páginas 21–26. IEEE, 2007.

[41] Group, Standish *y cols.*: *The CHAOS report.* Capturado em: http://www. standishgroup. com, 1995.

[42] Gullapalli, Vijaykumar: *A stochastic reinforcement learning algorithm for learning real-valued functions.* Neural networks, vol.3:671–692, 1990.

[43] Hailesellasie, Muluken, Syed Rafay Hasan, Faiq Khalid, Falah Aw Wad y Muhammad Shafique: *FPGA-Based Convolutional Neural Network Architecture with Reduced Parameter Requirements.* En *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*, páginas 1–5. IEEE, 2018.

[44] Hamill, Maggie y Katerina Goseva-Popstojanova: *Common trends in software fault and failure data.* IEEE Transactions on Software Engineering, vol.35:484–496, 2009.

[45] Hoo-Chang, Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura y Ronald M Summers: *Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning.* IEEE transactions on medical imaging, vol.35:1285, 2016.

[46] Huang, Fei y Alexander Yates: *Exploring representation-learning approaches to domain adaptation.* En *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*, páginas 23–30. Association for Computational Linguistics, 2010.

[47] Hussain, Ishrar, Leila Kosseim y Olga Ormandjieva: *Using linguistic knowledge to classify non-functional requirements in SRS documents.* En *International Conference on Application of Natural Language to Information Systems*, páginas 287–298. Springer, 2008.

[48] Hussain, Ishrar, Olga Ormandjieva y Leila Kosseim: *Lasr: A tool for large scale annotation of software requirements.* En *2012 Second IEEE International Workshop on Empirical Requirements Engineering (EmpiRE)*, páginas 57–60. IEEE, 2012.

[49] IEEE, Computer Society Software Engineering Standards Committee y Standards Board IEEE-SA: *IEEE Recommended Practice for Software Requirements Specifications*. 1998.

[50] IEEE, Computer Society Software Engineering Technical Committee: *IEEE standard glossary of software engineering terminology*. 1983.

[51] International, Standish Group: *The chaos report.* United States of America, 2015.

[52] Jonathan Bertman, Neil Skolnik y Jane Anderson: *EHrs get a failing grade on usability.* Internal Medicine News, vol.43:50, 2010.

[53] Juárez, Reyes, Guillermo Licea *y cols.*: *Towards Supporting Software Engineering Using Deep Learning: A Case of Software Requirements Classification.* En *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*, páginas 116–120. IEEE, 2017.

[54] Kassab, Mohamad: *Non-functional requirements: modeling and assessment.* VDM Verlag, 2009.

[55] Kauppinen, Marjo, Juha Savolainen y Tomi Mannisto: *Requirements engineering as a driver for innovations.* En *15th IEEE International Requirements Engineering Conference (RE 2007)*, páginas 15–20. IEEE, 2007.

[56] Khan, Musawwer, Junaid Rashid y Muhammad Wasif Nisar: *A CMMI Complaint Requirement Development Life Cycle.* International Journal of Computer Science and Information Security, vol.14:1000, 2016.

[57] Khatter, Kiran y Arvind Kalia: *Impact of non-functional requirements on requirements evolution.* En *Emerging Trends in Engineering and Technology (ICETET), 2013 6th International Conference on*, páginas 61–68. IEEE, 2013.

[58] Kim, Yoon: *Convolutional neural networks for sentence classification.* arXiv preprint arXiv:1408.5882, 2014.

[59] Kingma, Diederik P y Jimmy Ba: *Adam: A method for stochastic optimization.* arXiv preprint arXiv:1412.6980, 2014.

[60] Ko, Youngjoong, Sooyong Park, Jungyun Seo y Soonhwang Choi: *Using classification techniques for informal requirements in the requirements analysis-supporting system.* Information and Software Technology, vol.49:1128–1140, 2007.

[61] Kumar, Pawan y RA Khan: *Classification of Software Requiremnt Errors: A Critical Review.* International Journal of Computer Applications, vol.132:9–14, 2015.

[62] Kurtanović, Zijad y Walid Maalej: *Automatically classifying functional and non-functional requirements using supervised machine learning.* En *Requirements Engineering Conference (RE), 2017 IEEE 25th International*, páginas 490–495. IEEE, 2017.

[63] Lai, Siwei, Liheng Xu, Kang Liu y Jun Zhao: *Recurrent convolutional neural networks for text classification.* En *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[64] LeCun, Yann, Yoshua Bengio y Geoffrey Hinton: *Deep Learning.* Nature, vol.521, 2015.

[65] Lehtinen, Timo OA, Mika V Mäntylä, Jari Vanhanen, Juha Itkonen y Casper Lassenius: *Perceived causes of software project failures–An analysis of their relationships.* Information and Software Technology, vol.56:623–643, 2014.

[66] Leung, Henry y Simon Haykin: *The complex backpropagation algorithm.* IEEE Transactions on signal processing, vol.39:2101–2104, 1991.

[67] Luisa, Mich, Franch Mariangela y Novi Inverardi Pierluigi: *Market research for requirements analysis using linguistic tools.* Requirements Engineering, vol.9:40–56, 2004.

[68] Mahmoud, Anas y Grant Williams: *Detecting, classifying, and tracing non-functional software requirements.* Requirements Engineering, vol.21:357–381, 2016.

[69] Mead, Nancy R y Ted Stehney: *Security quality requirements engineering (SQUARE) methodology*, volumen vol.30. ACM, 2005.

[70] Menzies, Tim, Bora Caglayan, Ekrem Kocaguneli, Joe Krall, Fayola Peters y Burak Turhan: *The promise repository of empirical software engineering data*, 2012.

[71] Mikolov, Tomáš: *Statistical language models based on neural networks*. Presentation at Google, Mountain View, 2nd April, vol.80, 2012.

[72] Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado y Jeff Dean: *Distributed representations of words and phrases and their compositionality*. En *Advances in neural information processing systems*, páginas 3111–3119, 2013.

[73] Mnih, Andriy y Geoffrey Hinton: *Three new graphical models for statistical language modelling*. En *Proceedings of the 24th international conference on Machine learning*, páginas 641–648. ACM, 2007.

[74] Niu, Nan, Sjaak Brinkkemper, Xavier Franch, Jari Partanen y Juha Savolainen: *Requirements engineering and continuous deployment*. IEEE software, vol.35(2):86–90, 2018.

[75] Pacheco, Carla, Ivan Garcia y Miryam Reyes: *Requirements elicitation techniques: a systematic literature review based on the maturity of the techniques*. IET Software, vol.12:365–378, 2018.

[76] Palmer, JD y Yiqing Liang: *Indexing and clustering of software requirements specifications*. Information and decision Technologies, vol.18:283–299, 1992.

[77] Papasaikas, PK, PG Bagos, ZI Litou y SJ Hamodrakas: *A novel method for GPCR recognition and family classification from sequence alone using signatures derived from profile hidden Markov models*. SAR and QSAR in Environmental Research, vol.14:413–420, 2003.

[78] Pohl, Klaus: *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010.

[79] Qi, Yanjun, Sujatha G Das, Ronan Collobert y Jason Weston: *Deep learning for character-based information extraction*. En *European Conference on Information Retrieval*, páginas 668–674. Springer, 2014.

[80] Ramadhani, Denni Aldi, Siti Rochimah y Umi Laili Yuhana: *Classification of Non-Functional Requirements Using Semantic-FSKNN Based ISO/IEC 9126*. TELKOMNIKA (Telecommunication Computing Electronics and Control), vol.13:1456–1465, 2015.

[81] Ramos, Erik: *Aprendizaje de representaciones de secuencia de aminoácidos utilizando arquitecturas profundas*. Tesis de Licenciatura, Universidad Tecnológica de la Mixteca, 2016.

[82] Rashwan, Abderahman, Olga Ormandjieva y René Witte: *Ontology-based classification of non-functional requirements in software specifications: a new corpus and svm-based classifier*. En *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*, páginas 381–386. IEEE, 2013.

[83] Rempel, Patrick y Parick Mäder: *Preventing defects: The impact of requirements traceability completeness on software quality*. IEEE Transactions on Software Engineering, vol.43:777–797, 2017.

[84] Robert, Christian: *Machine learning, a probabilistic perspective*, 2014.

[85] Ryan, K.: *The role of natural language in requirements engineering*. En *Proceedings of the IEEE International Symposium on Requirements Engineering*, páginas 240–242. IEEE, 1993.

[86] Sanders, Mark S y Ernest J McCormick: *Human factors in engineering and design*. McGRAW-HILL book company, 1987.

[87] Sayyad Shirabad, J. y T.J. Menzies: *The PROMISE Repository of Software Engineering Databases*. School of Information Technology and Engineering, University of Ottawa, Canada, 2005. `http://promise.site.uottawa.ca/SERepository`.

[88] Shanyour, Belal y Abdallah Qusef: *Global Software Development and its Impact on Software Quality*. En *2018 Fifth International Symposium on Innovation in Information and Communication Technology (ISIICT)*, páginas 1–6. IEEE, 2018.

[89] Singh, Prateek, Deepali Singh y Ashish Sharma: *Rule-based system for automated classification of non-functional requirements from requirement specifications*. En *Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on*, páginas 620–626. IEEE, 2016.

[90] Slankas, John y Laurie Williams: *Automated extraction of non-functional requirements in available documentation*. En *Natural Language Analysis in Software Engineering (NaturaLiSE), 2013 1st International Workshop on*, páginas 9–16. IEEE, 2013.

[91] Sommerville, Ian: *Software engineering, 9th Edition*. Pearson, 2011.

[92] Tahir, Amjed y Rodina Ahmad: *Requirement engineering practices-an empirical study*. En *2010 International Conference on Computational Intelligence and Software Engineering*, páginas 1–5. IEEE, 2010.

[93] Tiwari, Saurabh, Deepti Ameta y Asim Banerjee: *An Approach to Identify Use Case Scenarios from Textual Requirements Specification*. En *Proceedings of the 12th Innovations on Software Engineering Conference (formerly known as India Software Engineering Conference)*, página 5. ACM, 2019.

[94] Umar, Mahrukh y Naeem Ahmed Khan: *Analyzing non-functional requirements (nfrs) for software development.* En *2011 IEEE 2nd International Conference on Software Engineering and Service Science*, páginas 675–678. IEEE, 2011.

[95] Van Hulse, Jason, Taghi M Khoshgoftaar y Amri Napolitano: *Experimental perspectives on learning from imbalanced data.* En *Proceedings of the 24th international conference on Machine learning*, páginas 935–942, 2007.

[96] Van Lamsweerde, Axel: *Requirements engineering: From system goals to UML models to software.* Chichester, UK: John Wiley & Sons, 2009.

[97] Winkler, Jonas y Andreas Vogelsang: *Automatic classification of requirements based on convolutional neural networks.* En *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, páginas 39–45. IEEE, 2016.

[98] Zhang, Wen, Ye Yang, Qing Wang y Fengdi Shu: *An empirical study on classification of non-functional requirements.* En *The Twenty-Third International Conference on Software Engineering and Knowledge Engineering (SEKE 2011)*, páginas 190–195, 2011.

Anexos

Anexos A

Concentrado de resultados bajos para el modelo CNN

Matriz embebida	Tamaño de filtro	# de filtros por tamaño	LOSS	ACC.	Métrica	Promedio
Glove	{1,2}	256	1.29	0.79	Recall	0.819
					Precision	0.791
					F1	0.787
Word2vec	{1,2}	256	1.76	0.81	Recall	0.750
					Precision	0.794
					F1	0.767
FastText	{1,2,3}	256	1.11	0.78	Recall	0.810
					Precision	0.825
					F1	0.796

Figura A.1: Resultados de clasificación de NFR con CNN para Glove y Word2Vec sin parámetro n_range, y FastText con n_range=3.

Anexos B

Clasificación de NFR en una aplicación web

La aplicación se desarrollo en IDE de python *Pycharm*. Es recomendable usarlo en este entorno, por si se necesita mejorarlo; por lo que correrlo en dicho entorno provee de facilidad para un usuario inexperto. Para poder realizar la clasificación es necesario iniciar el servicio de *back-end* y posteriormente se ingresa a una dirección web en el navegador de su preferencia que permita ingresar a la aplicación con interfaz de usuario. Estos procedimientos se explican en las siguientes secciones.

B.1. Configuración back-end

Una vez instalado *Pycharm*, se selecciona Archivo/File >Nuevo proyecto/New Project y aparecerá la ventana que se muestra en la Figura B.1. Por lo que primero se debe cargar el proyecto, dando clic en la ubicación del proyecto y seleccionarlo de donde lo hayamos descomprimido. Además hay que verificar que el entorno virtual seleccionado sea Virtualenv y el interprete sea Python 3.7. Por último seleccionamos crear. Tardará algunos minutos para cargar el proyecto.

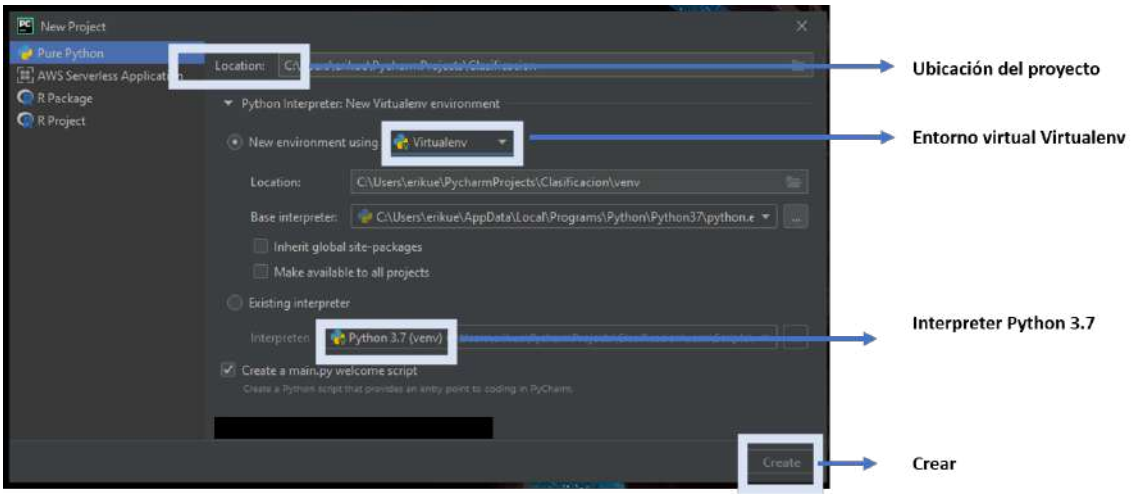


Figura B.1: Pantalla de carga del proyecto clasificación.

Una vez cargado el proyecto, es necesario verificar que efectivamente la versión de Python es la que se está seleccionando, así como los paquetes que necesitan estar instalados. Como se muestra en la Figura B.2, debe dirigirse a Archivo/File >Configuración/Settings.

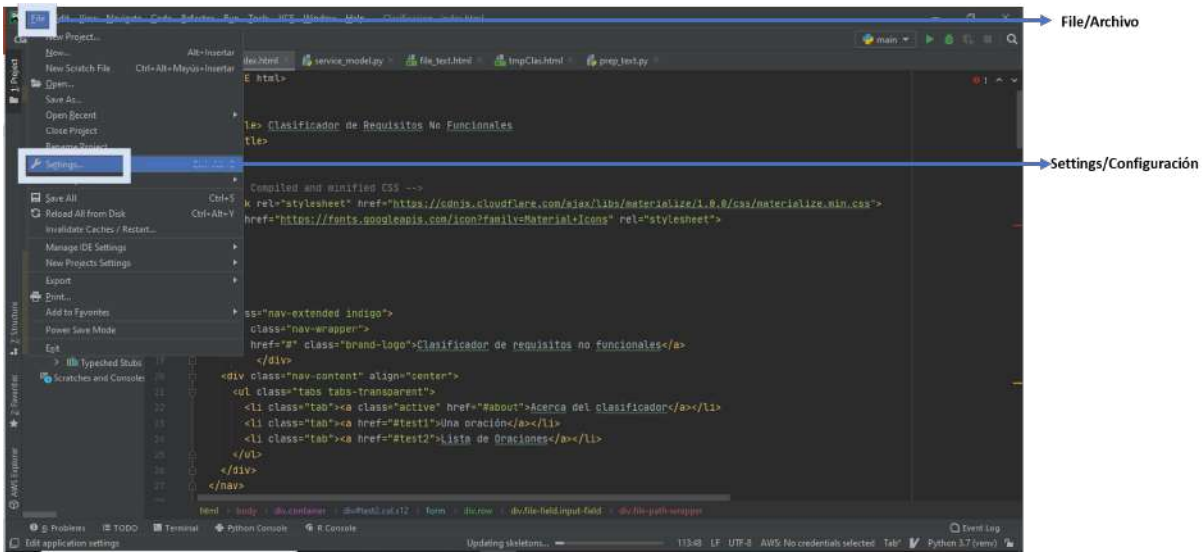


Figura B.2: Pantalla de selección de configuración del proyecto.

Al seguir la ruta anterior, aparecerá la pantalla como se muestra en la Figura B.3. Es necesario verificar que el interprete sea el mismo que se ha mencionado desde un inicio de este manual de configuración, ya que de no ser así el proyecto no podrá ser ejecutado correctamente y no funcionará.Otra tarea importante a realizar en esta sección es la de verificar que los paquetes que se en listan en el documento *package.txt*, que se proporciona en la raíz del directorio del proyecto, se encuentren instalados, para hacer debe de dar clic en el simbolo de + (Agregar paquetes/add), señalado en la Figura mencionada.

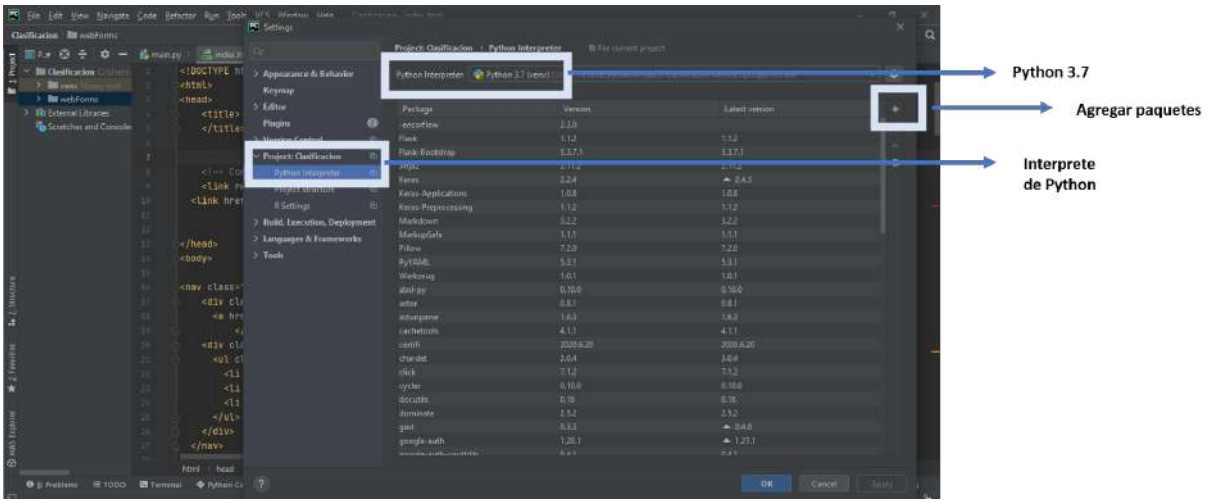


Figura B.3: Pantalla de configuración del proyecto

Al dar clic en agregar paquetes, podrá revisar si se encuentra instalado, en caso de que no, es necesario buscarlo e instalarlo con la versión que se especifica para que el proyecto pueda funcionar adecuadamente (Ver Figura B.4). Al finalizar la instalación, cierre la ventana actual y podrá visualizar la pantalla anterior (Figura B.3), al estar seguro de tener todo lo detallado en el documento de paquetes, proceder a dar clic en aplicar/apply y en botón de aceptar/ok. Puede cerrar la pantalla y se dirigirá a la pantalla inicial.

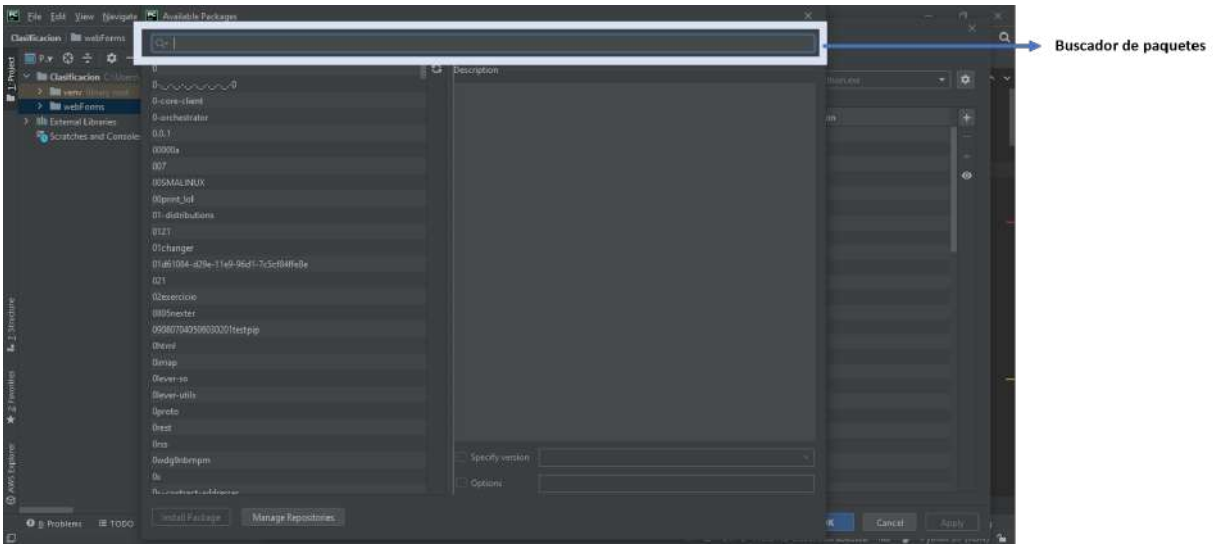


Figura B.4: Pantalla de configuración del interprete.

Ahora, ya configurado el entorno, en necesario ejecutar el script *main.py*, este está alojado en la carpeta *webForms*. La ubicación de estos se encuentra en en el panel izquierdo que muestra el directorio del proyecto. Una vez localizado el script *main.py*, dar clic derecho sobre éste y se desplegará un menú que contiene el comando Run, como se observa en la Figura B.5. Las Figuras B.6 y B.7, muestran el proceso de ejecución. La última Figura,

muestra el proyecto corriendo y provee la dirección para poder visualizar la página web. *NOTA:* Cuando termine de utilizar el sitio web, cerrar la página, volver a la aplicación Pycharm y dar clic en el icono de *stop*, como se muestra resaltado en la Figura *run2*.

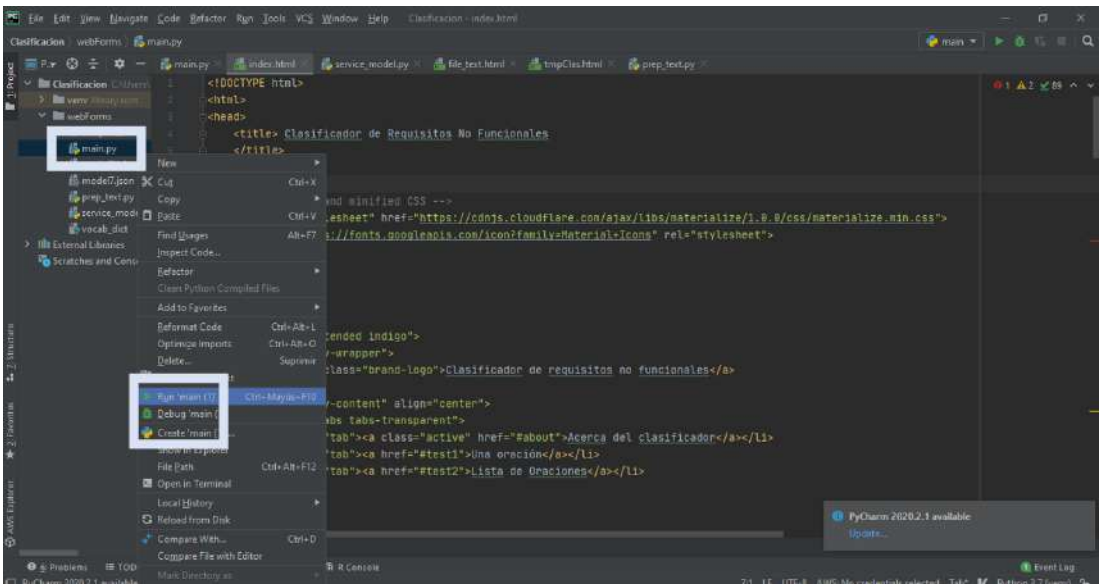


Figura B.5: Pantalla de ejecución del proyecto.

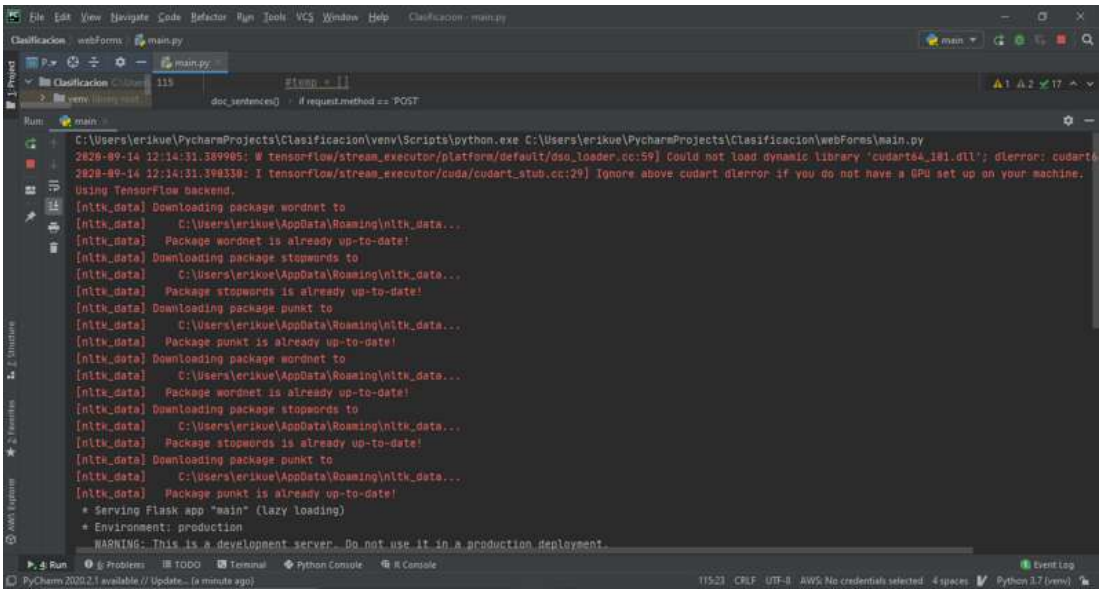


Figura B.6: Pantalla del proceso de ejecución.

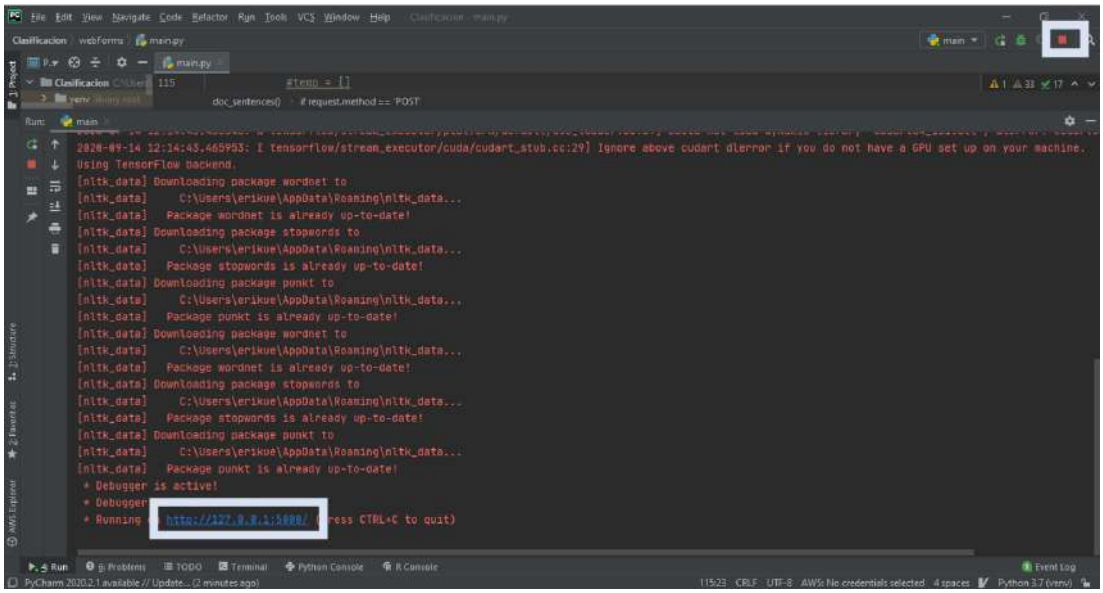


Figura B.7: Pantalla de ejecución exitosa.

B.2. Manual de usuario

Una vez realizada la configuración de *back-end*, se puede ingresar al sitio web dando clic en el link señalado en la Figura B.7. Esto es accediendo a través de la dirección *http://127.0.0.1:5000/*. Al dar clic desde Pycharm, automáticamente abre el navegador predeterminado y se puede ver la aplicación web como se muestra en la Figura B.8. Se observan 3 pestañas de navegación dentro de esta, la primera es **"Acerca del clasificador"**; en ella se hace una breve descripción del clasificador y el tipo de entrada para este, al dar clic en cada pregunta se desglosa la respuesta.

Figura B.8: Pantalla de aplicación web: Clasificador de requisitos no funcionales.

En la segunda pestaña llamada **"Una oración"**, se realiza la clasificación de una oración. Un ejemplo se muestra en la Figura B.9. Al dar clic en clasificar el resultado se

muestra en la Figura B.10. En donde se aprecia la oración/requisito a clasificar, la clase a la que pertenece según el clasificador y el porcentaje con el cual el clasificador la ha detectado. Cabe mencionar que en algunos casos el porcentaje es aparentemente bajo, pero debe recordarse que el resultado es el más alto de la predicción de la clasificación multiclase del tema de tesis, basado en 9 clases de requisitos no funcionales obtenidos del conjunto de datos Promise. Por lo que se toma el argumento máximo de la salida de la predicción.

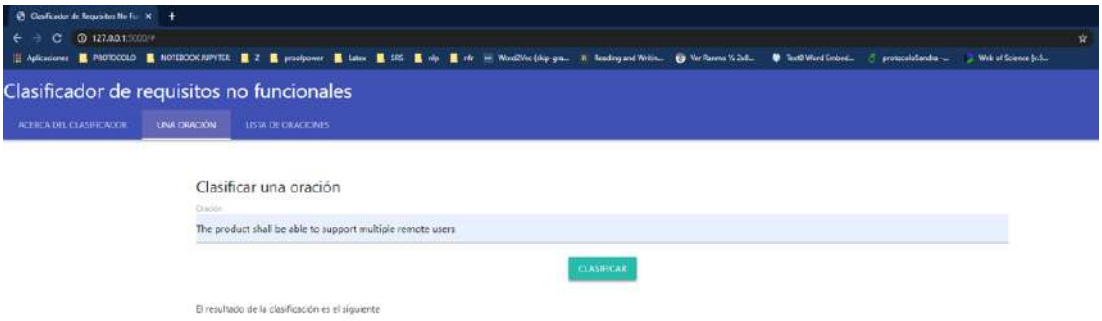


Figura B.9: Pantalla clasificación de una oración/requisito.

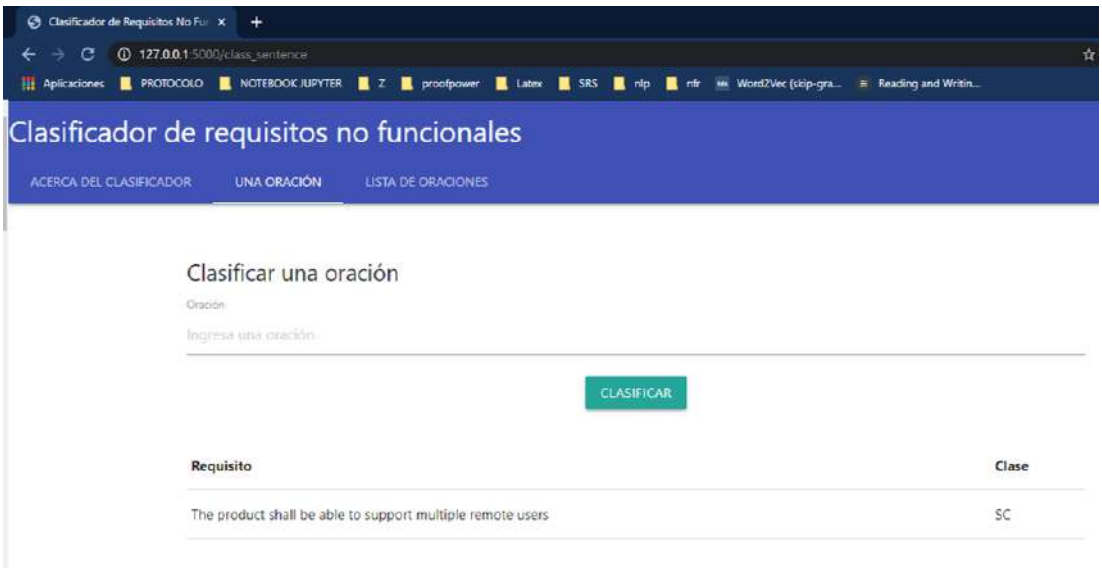


Figura B.10: Pantalla resultado de la clasificación de una oración/requisito.

La clasificación un documento de oraciones, se realiza en la pestaña que lleva de nombre **Lista de oraciones** (Ver Figura B.11). Para clasificar es necesario cargar un documento del tipo *CSV*, en donde se tenga almacenada la lista de requisitos en una sola columna. Una vez cargado el documento dar clic en el botón de clasificar y desplegará la lista

de requisitos mostrando: la oración/requisito, la clase a la que el modelo predijo y el porcentaje de predicción.

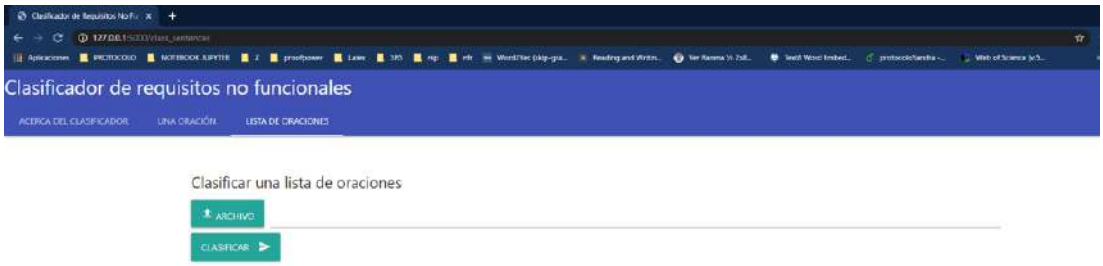


Figura B.11: Pantalla clasificación de una lista de oraciones/requisitos.

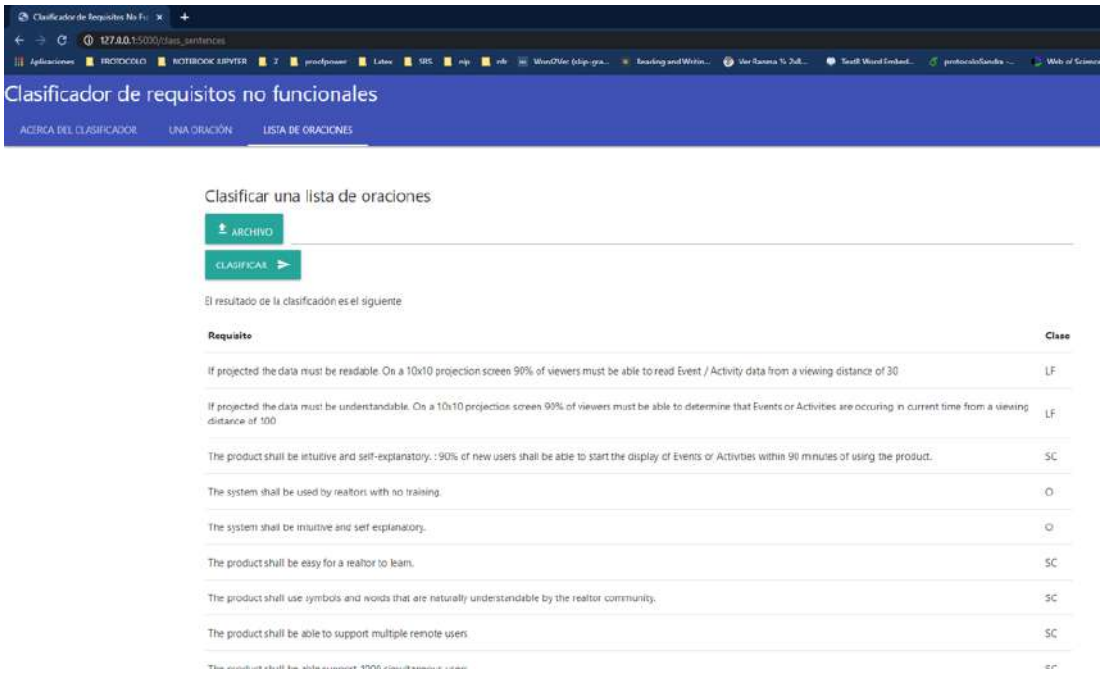


Figura B.12: Pantalla de resultados de clasificación de una lista de oraciones/requisitos.