Modelo de Clasificación de Requisitos de Software Mediante la Aplicación de Técnicas de Aprendizaje Automático

Sara Abadía Sarria

Nota de Aceptación

Certificamos que el presente Trabajo de Grado Satisface, en alcances y calidad, todos los requisitos que demanda un Trabajo de Grado de Maestría.

Maria Constanza Pabón
Director

Gloria Inés Álvarez Vargas
Jurado

Diego Luis Linares
Jurado

Aprobado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana Cali, para optar el título de Magister.

HERNÁN CAMILO ROCHA NIÑO Ph. D.
Decano Facultad de Ingeniería y Ciencias

JUAN CARLOS MARTÍNEZ ARIAS
Director Posgrados de Ingeniería y Ciencias

Santiago de Cali, 11 de mayo de 2022

Acta de Correcciones al Documento de Trabajo de Grado

Santiago de Cali, 11 de mayo de 2022

Autor: Sara Abadía Sarria

Título del Trabajo de Grado: "Modelo de Clasificación de Requisitos de Software Mediante la Aplicación de Técnicas de Aprendizaje Automático"

Director: Maria Constanza Pabon

Como indica el artículo 2.13 de las Directrices para Trabajo de Grado de Maestría, he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Trabajo de Grado definieron que se efectuaran, como consta en el Acta de Evaluación correspondiente.

Firma del Director del Trabajo de Grado

Datos del estudiante

Nombre completo: Sara Abadía Sarria

Dirección: Calle 30 # 1-15 Jamundí, Valle del Cauca

Correo electrónico: sara.abadia@gmail.com

Profesión: Ingeniera de Sistemas

Teléfono celular: +57 (602) 3006601081

MODELO DE CLASIFICACIÓN DE REQUISITOS DE SOFTWARE MEDIANTE LA APLICACIÓN DE TÉCNICAS DE APRENDIZAJE AUTOMÁTICO

SARA ABADÍA SARRIA

PONTIFICIA UNIVERSIDAD JAVERIANA DE CALI

MAESTRÍA EN INGENIERÍA CON ÉNFASIS EN SISTEMAS Y COMPUTACIÓN

CALI

2022

MODELO DE CLASIFICACIÓN DE REQUISITOS DE SOFTWARE MEDIANTE LA APLICACIÓN DE TÉCNICAS DE APRENDIZAJE AUTOMÁTICO

SARA ABADÍA SARRIA

Trabajo de Grado para optar por el título de Magister

Director:

María Constanza Pabón

Co - Director:

Juan Carlos Martínez Arias

PONTIFICIA UNIVERSIDAD JAVERIANA DE CALI

MAESTRÍA EN INGENIERÍA CON ÉNFASIS EN SISTEMAS Y COMPUTACIÓN

CALI

2022

CONTENIDO

LISTA DE CUADROS

LISTA DE FIGURAS

ABSTRACT

The definition of natural language requirements is a process that can be time-consuming in large software projects. Classifying natural language software requirements into functional and non-functional classes, and at the same time non-functional sub-classes such as performance, compatibility, usability, reliability, security, maintainability, and portability, is a task that contributes to the requirements definition for successful software projects. This classification task requires expert judgment and is time-consuming, being challenging because it is a manual process. Automating requirements classification is a strategy to streamline the activities of requirements engineers. Related studies show the existence of scarce software requirements data sets, which makes it difficult to promote, create and improve predictive models that facilitate the tasks of automatic requirements classification, in addition, these scarce data sets in most of them are defined in English, therefore, the predictive models developed cannot be used directly for projects in different languages because the grammar varies with the language. In view of the above, this work focuses on generating predictive models of natural language software requirements classification into functional and non-functional classes, and at the same time non-functional sub-classes according to the ISO / IEC 25010 standard, to contribute to the development of studies that apply machine learning techniques in the requirements engineering context for projects developed in Spanish. The results shown an F1-Score above 60% for all the five experiments where oversampling was used. The study was conducted with a sample of more than 2,800 software requirements described in Spanish that were previously translated and consolidated from multiple data sets in English widely used in other research, of which 1,887 requirements were manually tagged. The translation into Spanish was done through Google's automatic translation tool and subsequently, the translation was verified manually. This data set in Spanish will be available to the scientific community.

Keywords: *Machine learning, Requirements classification, Repository of software requirements in Spanish.*

RESUMEN

La definición de los requisitos del lenguaje natural es un proceso que puede llevar mucho tiempo en grandes proyectos de software. Clasificar los requisitos de software de lenguaje natural en funcionales y no funcionales, y al mismo tiempo categorías de no funcionales como rendimiento, compatibilidad, usabilidad, confiabilidad, seguridad, mantenibilidad y portabilidad, es una tarea que contribuye a la definición de requisitos para que proyectos de software sean exitosos. Esta tarea de clasificación requiere el juicio de un experto y requiere mucho tiempo, siendo un desafío porque es un proceso manual. La automatización de la clasificación de requisitos es una estrategia para agilizar las actividades de los ingenieros de requisitos. Estudios relacionados muestran la existencia de escasos conjuntos de datos de requisitos de software, lo que dificulta promover, crear y mejorar modelos predictivos que faciliten las tareas de clasificación automática de requisitos, además, estos escasos conjuntos de datos en la mayoría de ellos están definidos en inglés, por lo tanto, los modelos predictivos desarrollados no se pueden utilizar directamente para proyectos en diferentes idiomas porque la gramática varía con el idioma. En vista de lo anterior, este trabajo se enfoca en generar modelos predictivos de clasificación de requisitos de software de lenguaje natural en funcionales y no funcionales, y al mismo tiempo las categorías de no funcionales de acuerdo con el estándar ISO/IEC 25010, para contribuir al desarrollo de estudios que apliquen técnicas de aprendizaje automático en el contexto de la ingeniería de requisitos para proyectos desarrollados en español. Los resultados muestran un indicador F1 superior al 60% para la mayoría de los experimentos en donde se utilizó el aumento artificial de información. El estudio se realizó con una muestra de más de 2800 requisitos de software descritos en español que previamente fueron traducidos y consolidados a partir de varios conjuntos de datos en inglés ampliamente utilizados en otras investigaciones, de los cuales 1887 requisitos fueron etiquetados manualmente. La traducción al español se realizó a través de la herramienta de traducción automática de Google y posteriormente, la traducción se verificó manualmente. Este conjunto de datos en español estará disponible para la comunidad científica.

Palabras clave: *Aprendizaje automático, Clasificación de requisitos, Repositorio de requisitos de software en español.*

INTRODUCCIÓN

La ingeniería de requisitos es el primer paso que se realiza cuando se trata de desarrollar software, la cual consiste en recopilar, describir y especificar los requisitos del software (Pohl 2016). Los requisitos de software tienen un impacto significativo en el éxito del proyecto, pues las actividades siguientes del proceso de desarrollo de software, tales como diseño, implementación y pruebas, dependen de la precisión de los requisitos.

En ese contexto, la clasificación de los requisitos en funcionales y no funcionales es una práctica estándar que en la actualidad tiene amplia aceptación (Glinz 2011), porque aporta en que la descripción de requisitos sea más precisa. Pero, esta tarea requiere conocimientos, capacitación, experiencia y conocimiento del dominio de los arquitectos, analistas y desarrolladores de software, resultando desafiante y lenta por ser un proceso manual. Adicionalmente, previas investigaciones indican que el 80 % de los inconvenientes que provocan insatisfacción del cliente con el software que se entrega, se originan en el proceso de ingeniería de requisitos (Krasner 2018). Los diversos problemas que se encuentran en los requisitos aumentan los costos de mantenimiento y disminuyen la calidad del software, reiterando que la clasificación de los requisitos del software resulta un proceso crítico y necesario (Richard Berntsson, Tony y Regnell 2009).

Frente a lo anterior, recientemente se han desarrollado investigaciones para automatizar la clasificación de los requisitos de software. Para eliminar el análisis manual, se han aplicado diversas técnicas, entre ellas, el procesamiento del lenguaje natural y el aprendizaje automático. Esta automatización podría reducir enormemente el esfuerzo cognitivo necesario para dar sentido a las descripciones de los requisitos, contribuir a que el proceso de desarrollo de software sea más eficiente y mejorar la calidad del software (Baker y Al. 2019).

Sin embargo, debido a que los requisitos de software se describen principalmente en lenguaje natural, los modelos predictivos que se han generado en las investigaciones pierden la capacidad de generalizar el conocimiento para clasificar requisitos de otros proyectos, debido a diferencias en la gramática (Hey et al. 2020). Adicionalmente, la mayoría de los conjuntos de datos que se han utilizado en las investigaciones están conformados por requisitos escritos en inglés, este es otro factor que no permite usar los modelos generados en proyectos con requisitos escritos en otros idiomas, como el español (Gramajo, Ballejos y Ale 2019).

Por otra parte, la clasificación automática de requisitos es una tarea desafiante porque se encuentran pocos conjuntos de datos etiquetados y estos tienen pocos requisitos. Lo anterior, contrasta con los proyectos de aprendizaje automático realizados en otros dominios donde se encuentran conjuntos de datos de gigabytes, terabytes e incluso petabytes (Baker y Al. 2019).

Como se presenta en la Sección 1, este trabajo aborda la problemática de la clasificación de requisitos de software funcionales y no funcionales, más las categorías de los no funcionales, donde se desea aplicar técnicas de aprendizaje automático que permitan automatizar el proceso. Por otra parte, en la Sección 2 y 3, se presentan los objetivos y alcance del proyecto. En la Sección 4, se presentan los conceptos teóricos generales que se aplicaron en el proyecto. La Sección 5, de revisión de literatura, muestra que la

problemática del presente trabajo ha sido ampliamente estudiada, pero que no se encontraron investigaciones en proyectos de software con requisitos descritos en idioma español.

La Sección 6 presenta la metodología desarrollada para cumplir con los aportes de este trabajo, los cuales se enfocan en la aplicación de técnicas de aprendizaje automático para generar un comparativo de mínimo diez modelos de clasificación automática de requisitos funcionales y no funcionales, más las categorías de no funcionales consideradas en el estándar ISO/IEC 25010. De esta manera, se encontró el modelo entre los propuestos, que generó mejores predicciones sobre la clasificación de los requisitos para contribuir con el desarrollo de estudios que aplican técnicas de aprendizaje automático en el contexto de la ingeniería de requisitos y proyectos desarrollados en idioma español. Además, se generó un conjunto de datos de más de 2800 requisitos de software descritos en idioma español para disposición de la comunidad científica.

1. DEFINICIÓN DEL PROBLEMA

La definición de los requisitos de software es uno de los elementos más importantes en el desarrollo de software, ya que estos describen las características y propiedades que tendrá el sistema (Rahman et al. 2019). Estos requisitos se pueden clasificar, de manera general, en requisitos funcionales y requisitos no funcionales, donde los requisitos funcionales son declaraciones de los servicios que prestará el sistema, en la forma en que se comportará a determinados insumos. Por otro lado, los requisitos no funcionales son aquellos que no se refieren directamente a las funciones específicas suministradas por el sistema, sino a sus propiedades, por ejemplo, rendimiento, seguridad y disponibilidad.

De acuerdo con Navarro-Almanza, Juurez-Ramirez y Licea (2018), estos requisitos son generalmente definidos en la misma documentación, lo que genera un proceso operativo adicional de clasificar correctamente dichos requisitos en los que son funcionales y los que no, teniendo en cuenta que una clasificación inadecuada puede llevar a reprocesos en el desarrollo del software (aumentando su costo de producción) o, incluso, al fracaso de todo el proyecto.

Si bien, existen metodologías para la definición y clasificación de requisitos funcionales y no funcionales, como la propuesta por Sites (2020), en la práctica no se definen dichos elementos con la rigurosidad que se debe y, adicionalmente, para proyectos de alta envergadura, dicha definición se vuelve compleja, lo que conlleva a un alto esfuerzo por parte del equipo de desarrollo para la clasificación de los requisitos. Por esta razón, se ha abordado la clasificación de los requisitos mediante modelos de aprendizaje automático y utilizando técnicas estandarizadas para el proceso de clasificación de textos (Kowsari et al. 2019).

La clasificación de requisitos ha sido ampliamente abordada como un problema de aprendizaje supervisado. Estudios como el de Binkhonain y Zhao (2019) muestra un resumen de los procesos utilizados para el pre-procesamiento de los datos, las técnicas y algoritmos más empleados y las métricas de desempeño utilizadas. No obstante, como se describirá en la Sección 5, los estudios se enfocan en desarrollar modelos para clasificar requisitos definidos, en su mayoría, en el idioma inglés, lo que implica que los modelos no se pueden utilizar de manera directa a proyectos desarrollados en idiomas diferentes (Hey et al. 2020), porque la gramática varía con el idioma. Adicionalmente, se evidencia la escases de conjuntos de datos de requisitos de software para ampliar y fomentar el desarrollo de estudios que apliquen técnicas de aprendizaje automático (Lima et al. 2019) y (Dalpiaz et al. 2019), en el contexto de la ingeniería de requisitos.

De acuerdo con lo anterior, en este trabajo se aplicarán técnicas de aprendizaje automático supervisado para clasificar los requisitos funcionales, no funcionales, más las categorías de no funcionales consideradas en el estándar ISO/IEC 25010, con la finalidad de agilizar los procesos de definición de requisitos en proyectos de software que los describen en idioma español, desde el punto de vista de ahorrar a los analistas, arquitectos y desarrolladores, el análisis manual de requisitos, reducir el esfuerzo cognitivo requerido para dar sentido a las descripciones de los requisitos, aportar a que el proceso de desarrollo de software sea más eficiente y mejorar la calidad del software, por consiguiente, disminuir el riesgo de extensión de los presupuestos en los proyectos de software. Asimismo, se generará un conjunto de requisitos de software descritos en español, el cual se pondrá a disposición de

la comunidad científica para fomentar el desarrollo de estudios que articulen el aprendizaje automático con la ingeniería de requisitos de proyectos de software en español.

2. OBJETIVOS

2.1 OBJETIVO GENERAL

Proponer un modelo computacional basado en inteligencia artificial, como una herramienta de apoyo en el proceso de clasificación de requisitos de software funcionales y no funcionales, más las categorías de no funcionales consideradas en el estándar ISO/IEC 25010.

2.2 OBJETIVOS ESPECÍFICOS

- Recopilar la información de requisitos de software descritos en lenguaje natural de idioma español, con la finalidad de consolidar el conjunto de datos necesario para entrenar los diferentes modelos de clasificación.
- Generar modelos de clasificación usando técnicas de aprendizaje automático para entrenar cada modelo con un conjunto óptimo de parámetros de control.
- Evaluar cada uno de los modelos, a fin de seleccionar el mejor modelo que se adapte a las condiciones de eficiencia y capacidad predictiva.

3. ALCANCE

Los resultados obtenidos fueron:

- Nuevo conjunto de datos en español: se tradujo al español el conjunto de datos proporcionado por Dalpiaz et al. (2019), (Lima et al. 2019) y (Ferrari, Spagnolo y Gnesi 2017) utilizando la herramienta de traducción automática de Google, teniendo en cuenta los avances demostrados en Radford (2018) y la aplicación de dichas herramientas como apoyo para realizar traducciones en menor tiempo. Sin embargo, el conjunto de datos traducido se validó manualmente para garantizar su confiabilidad, dadas las imprecisiones en las traducciones que relacionan Alshemali y Kalita (2020), evidenciando que a pesar de que el desempeño de las herramientas es aceptable, los resultados de la traducción automática deben revisarse y ajustarse en algunos detalles. Se etiquetó manualmente el conjunto de datos de (Ferrari, Spagnolo y Gnesi 2017) y Dalpiaz et al. (2019).

- Se generaron tres modelos de clasificación de requisitos de software funcionales y no funcionales, más las categorías consideradas en el estándar ISO/IEC 25010, para comparar y evaluar el desempeño de las técnicas de aprendizaje automático en requisitos de software descritos en español.

4. MARCO TEÓRICO

4.1. APRENDIZAJE AUTOMÁTICO

De acuerdo con (Quinto 2020) el aprendizaje automático se define como un campo específico de la inteligencia artificial, la ciencia y la ingeniería detrás de la gestión del aprendizaje de las computadoras.

El aprendizaje automático se ha convertido en la herramienta para muchas de las tareas que hoy en día se ejecutan, de manera que se ofrezcan experiencias personalizadas para cada tipo de usuario y, al mismo tiempo, se recolecte información valiosa para el mejoramiento de los modelos (Kamath, Liu y Whitaker 2019).

Algunos usos de este campo son:

- **Ventas y mercadeo en retail:** Se utiliza el aprendizaje automático para la personalización de ventas en función de los gustos del usuario. Así como el uso de estos modelos para la segmentación de clientes y la predicción de la demanda.
- **Transporte:** En la optimización de rutas, estimación de tiempos de viajes, mantenimiento preventivo y preventivo, así como en el manejo de vehículos autónomos.
- **Servicios financieros:** En la predicción del ciclo de vida de las operaciones de crédito y tesorería, predicción del riesgo de crédito para la colocación de cartera, detección de operaciones sospechosas o inusuales, entre otros.
- **Sector salud:** Se utiliza principalmente como herramienta de apoyo para la detección y diagnóstico de enfermedades asociadas al corazón, cáncer o tuberculosis, mediante la detección de patrones en imágenes.

El aprendizaje automático se divide en tres grandes ramas:

- **Aprendizaje por Refuerzo:** Este tipo de aprendizaje se enfoca en maximizar una función de ganancia en función de un conjunto de acciones que un actor (la máquina) ejecuta, de manera que se premien o refuercen las acciones que ayudan al actor a alcanzar la meta establecida y se descarten aquellas que no. Generalmente se utiliza este tipo de aprendizajes en juegos o escenarios de simulación como la conducción autónoma.

- **Aprendizaje No Supervisado:** Este tipo de aprendizaje se utiliza cuando la información no se encuentra *etiquetada*, es decir, cuando el objetivo es realizar una segmentación o agrupación de los datos que se proporcionan en función de sus características. Generalmente se utiliza para sistemas de segmentación de clientes (recomendaciones de videos, de productos, entre otros), o para detección de anomalías en la información.

- **Aprendizaje Supervisado:** Este tipo de aprendizaje se utiliza cuando la información si se encuentra *etiquetada*, es decir, se cuenta con una variable de respuesta que se desea predecir por parte del modelo. Dicha variable puede ser categórica (en cuyo caso son problemas de clasificación), o numérica (en cuyo caso son problemas de regresión).

4.1.1. Técnicas de clasificación

- **Máquina de Vectores de Soporte:** Esta técnica construye un hiperplano que permite maximizar el margen entre dos clases, de manera que divide el conjunto de datos en n subconjuntos (determinado por el número de clases). Se utiliza para problemas de clasificación lineal, no obstante, para abordar problemas no lineales se realiza una transformación de los datos mediante el uso de un *kernel* con la finalidad de que estos se conviertan en un problema lineal (Mukhopadhyay 2018).

- **Modelo Ensemble Tipo Bagging:** la técnica toma un modelo individual, por ejemplo, una máquina de vectores de soporte, y realiza una serie de entrenamientos individuales de una cantidad n de modelos (definida a priori) sobre un subconjunto de datos el cual ha sido seleccionado de manera aleatoria con o sin reemplazamiento (Kumar y Jain 2020). Aporta la ventaja de generalizar mejor la información ganada a partir del conjunto de datos y evitar el sobre entrenamiento.

- **Perceptrón Multicapa:** Es un tipo de red neuronal artificial, la cual está compuesta de varias capas de neuronas. En la Figura 1, se muestra una estructura general de la técnica, que consta de los siguientes elementos:

  - **Capa de entrada:** Es aquella capa que está visible para el usuario. Usualmente está compuesta por un número de entradas igual al número de variables de entrada, es decir, una "neurona" por variable de entrada. La función de cada "neurona" de esta capa será pasar la información de las variables de entrada a las capas ocultas.

  - **Capas ocultas:** Son aquellas capas que están entre a capa de entrada y la capa de salida, la función de estas capas es para *linealizar* el problema, es decir, que a las variables de entrada (que pueden no ser linealmente separables) se les realice algún tipo de transformación para convertirlos en un problema que pueda ser separado por un hiperplano.

  - **Capa de salida:** Esta capa es la encargada de mostrar el resultado de la variable de respuesta (o variables de respuesta) del problema abordado.

  - **Conexiones entre neuronas:** Representan cómo están relacionadas las neuronas de una capa, con las neuronas de la capa siguiente. Estas conexiones en el perceptrón multicapa están relacionadas hacia adelante, es decir, todas las neuronas de la capa n están relacionas con todas las neuronas de la capa n+1, no existen relaciones entre neuronas de una misma capa ni con neuronas de capa anteriores.

  - **Neuronas:** Son la unidad básica de procesamiento del perceptrón, estas hacen la función de tomar los valores de entrada (provenientes de la capa anterior o directamente de los datos de entrada), realizar una transformación de los datos y dar un valor de salida para las neuronas de la siguiente capa.

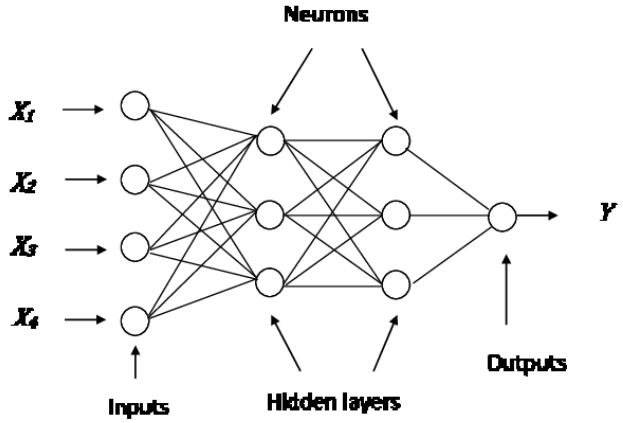

Figura 1*. Estructura perceptrón multicapa.*

Fuente: Tomado de (Kamath, Liu y Whitaker 2019)

- **Red Neuronal Convolucional**: Es un tipo de red neuronal que se utiliza particularmente para el análisis y extracción de características en problemas asociados con imágenes, texto y audio (Quinto 2020). Este tipo de redes utilizan una serie de filtros o *kernels* que permiten reconocer patrones en la información proporcionada y luego se pasa por una serie de capas de un perceptrón multicapa, el cual calcula la clase a la que pertenece la información.

- **Red Neuronal Recurrente**: Son un tipo de red neuronal donde para cada neurona de la red se ingresa la información obtenida a partir de la neurona anterior, de manera que se obtiene información sobre la posición de la información. La red neuronal recurrente es útil para diferentes tipos de problemas como análisis de series de tiempo, de palabras o, en general, de secuencias (Kamath, Liu y Whitaker 2019). Este tipo de redes han evolucionado a través del tiempo, implementando subcapas únicas como celdas LSTM, GRU o bidireccionales que tienen un mejor desempeño en problemas o casos específicos.

- **Transformer con Mecanismos de Atención:** Son un tipo de arquitectura de red neuronal desarrollada por (Vaswani et al. 2017), la cual posee capas de codificación y decodificación de la información que se ingresa, denominadas *mecanismos de atención* que priorizan o dan mayor relevancia a los segmentos más importantes. Esta técnica se utiliza generalmente en problemas de secuencia a secuencia como la predicción de texto o de series de tiempo.

4.1.2. Medidas de desempeño

La forma de evaluación de los modelos en los problemas de clasificación consiste en comparar las predicciones generadas por los algoritmos con los valores reales de la información proporcionada (Quinto 2020). La evaluación se calcula en función de los siguientes parámetros:

- Verdaderos positivos / verdaderos negativos, aquellos en donde la predicción y el valor real son iguales.

- Falsos negativos / falsos positivos. Aquellos en donde la predicción es diferente al valor real.

Existen diferentes métricas de desempeño, las cuales se describen a continuación:

- **Accuracy:** Se define como el número de predicciones correctas sobre el total de valores predichos:

$$Accuracy = \frac{Verdaderos\ Positivos + Verdaderos\ Negativos}{Verdaderos\ Positivos + Verdaderos\ Negativos + Falsos\ positivos + Falsos\ Negativos}$$

  Este tipo de métrica no es ideal cuando se tiene un problema con diferentes clases y el número de registros no está distribuido equitativamente, provocando que el indicador se sesgue por la clase predominante.

- **Precisión:** Se define como el número de predicciones correctas de la clase en específico, sobre el total de observaciones de dicha clase:

$$Precisión = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos + Falso\ Positivos}$$

  Esta métrica muestra qué tan bueno es el desempeño del modelo para predecir la clase positiva. Se utiliza cuando el costo de predecir un falso positivo es alto.

- **Recall:** Se define como el total de verdaderos positivos sobre el total de predicciones realizadas:

$$Recall = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos + Falsos\ Negativos}$$

- **F1 Score:** Este indicador realiza una ponderación entre el Recall y la Precisión, de manera que se tenga una visión holística del desempeño evaluado. Este indicador se utiliza generalmente para problemas de clasificación multiclase y se mide como se muestra a continuación:

$$Indicador\ F1 = 2\ x\ \frac{Precisión\ x\ Recall}{Precisión + Recall}$$

La métrica utilizada en este trabajo es el indicador F1.

4.1.3. Sobre muestreo

Cuando la base de datos con que se está entrenando un modelo de aprendizaje automático presenta desbalanceo en la información, es decir, se tiene mayor cantidad de registros de una clase específica, es recomendable utilizar técnicas para balancear la información y así evitar el sobre entrenamiento por parte de los modelos hacia la clase predominante. Para esto, se utilizan técnicas como *SMOTE (Synthetic Minority Over-sampling Technique* por sus siglas en inglés), que mediante el uso de algoritmos basados en vecinos más cercanos,

genera de manera artificial registros sobre las clases minoritarias manteniendo similitud en sus características, mejorando significativamente el desempeño de los modelos utilizados sobre las clases minoritarias (Chawla et al. 2002).

4.1.4. Validación Cruzada

La validación cruzada es un método de muestreo que permite a los modelos de aprendizaje automático generalizar la información obtenida de la base de datos y, de igual forma, prevenir el sobre entrenamiento (Berrar 2018). En esta metodología, se realiza un determinado número de iteraciones (conocido como *fold*) para el cual se divide la base en una sección de entrenamiento y prueba con la que se entrenará y validará el modelo, respectivamente. Este proceso se repite un número establecido de veces de manera que el desempeño del modelo será calculado con una medida de tendencia central de los desempeños individuales obtenidos en cada *fold*.

4.2. PROCESAMIENTO DE LENGUAJE NATURAL

Según (Fong 2018) el Procesamiento del Lenguaje Natural (PLN), estudia la interpretación y generación de texto a partir del análisis computacional. Este tipo de problemas se resuelven con modelos supervisados o no supervisados (algunos ejemplos de esos modelos se muestran en el capítulo 5).

4.2.1. Etapas de procesamiento de lenguaje natural

Las etapas se detallan a continuación:

- **Adquisición de información:** este paso consiste en obtener los textos que se utilizarán para el problema de PLN.
- **Limpieza de datos:** Este paso consiste en evaluar la calidad de los datos, remover caracteres que puedan afectar el desempeño de los modelos, establecer palabras clave dentro de los textos, entre otras acciones.
- **Preprocesamiento de información e ingeniería de características:** este paso consiste en seleccionar una metodología para representar textos como números. Dentro de la literatura existen diferentes métodos, entre ellos, el uso de *embeddings, skip-grams, vectorización y demás*.
- **Modelado del problema:** Este paso consiste en seleccionar un portafolio de modelos a entrenar para seleccionar el más apropiado para la tarea, teniendo en cuenta una o más métricas de desempeño que pueden ser diferentes según el problema a resolver.
- **Evaluación y mejora continua:** una vez que se selecciona el modelo, se monitorea para evaluar cuándo se debe reentrenar el modelo.

El problema de clasificación de los requisitos de software planteado en este proyecto se abordó como un problema de PLN.

4.2.2. Técnicas de procesamiento de lenguaje natural

Para el proceso de entrenamiento de los modelos de lenguaje natural, es necesario realizar el procesamiento de las palabras hacia un mapeo numérico que los modelos puedan entender. A continuación, se muestran las principales técnicas aplicadas para la representación numérica de las palabras, así como el proceso de normalización y regularización:

- **Tokenización:** Consiste en la segmentación de frases en grupos de palabras, subpalabras o caracteres (los cuales pueden ser letras o 1 sola palabra). Existen diferentes metodologías para la segmentación de las frases en las cuales se tienen en cuenta la puntuación o palabras claves conocidas como *stop words* (Kamath, Liu y Whitaker 2019). Este proceso es un paso previo a la representación numérica del texto el cual tiene una alta incidencia en el desempeño de los modelos.

- **Bag of Words:** Es una técnica de representación numérica de palabras, en donde se asigna un valor numérico del token o palabra en función de la frecuencia de aparición en el texto. Este tipo de técnicas genera una gran pérdida de información por cuanto las palabras más frecuentes en el texto no necesariamente aportan un valor semántico al texto, por tanto, un filtro inicial de remoción de *stop words* es necesario para mitigar este inconveniente (Kamath, Liu y Whitaker 2019).

- **N-gramas:** El modelo de *bag of words* no captura la semántica o la relación de cada palabra con respecto al texto adyacente. Por tanto, la agrupación de varias palabras como tokens es una técnica eficiente para capturar la relación gramatical entre estas. La cantidad de n-gramas utilizados generalmente no debe ser alto ya que afecta en la inferencia del texto por parte del modelo (Kamath, Liu y Whitaker 2019).

- **Steeming:** Es una técnica que consiste en *normalizar* las palabras, de manera que se reduzca la variabilidad del número de palabras relacionadas con un mismo significado, pero manteniendo su semántica. Por ejemplo, buscar una palabra asociada a una palabra conjugada. Esto permite un mejor análisis por parte del modelo y ayuda a mitigar problemas gramaticales y de escritura generados en los propios documentos (Salur y Aydin 2020).

- **Lemmatizacion:** Es otra técnica de normalización de palabras. A diferencia del *steeming*, esta técnica convierte a su versión base una palabra (como los verbos en infinitivos), esta técnica ayuda de igual forma para la simplificación de la dispersión de las palabras (Kamath, Liu y Whitaker 2019).

- **TF-IDF (Frecuencia de Término, Frecuencia Inversa de Documento):** Esta técnica consiste en otorgarle un mayor peso a las palabras menos frecuentes, dándole menor peso a palabras muy comunes que, se infiere, no aporta información semántica relevante (Kamath, Liu y Whitaker 2019).

- **Word Embeddings:** Esta técnica permite representar las palabras, frases o tokens agrupados mediante un vector o matriz de números que permitan una asociación que para un mejor desempeño y análisis por parte de los modelos entrenados. Generalmente se utilizan modelos específicos para crear estos *embeddings* (como GloVe o Word2Vec) los cuales se entrenan con un conjunto muy grande de palabras (Kamath, Liu y Whitaker 2019). El uso de esta técnica permite una representación de los textos que aportarán más información a los modelos.

4.3. INGENIERÍA DE REQUISITOS

Dentro del proceso de desarrollo de software, identificar las necesidades (requisitos) del cliente es clave para obtener los resultados deseados en el producto final (Pohl 2016). Por tanto, existe un marco teórico de buenas prácticas para que el proceso de elicitación de esas necesidades sea lo mejor posible. Las principales actividades que se deben tener en cuenta en el proceso de identificación de las necesidades del cliente son: a. Descubrimiento de requisitos, b. Clasificación y organización de requisitos, c. Priorización y negociación de requisitos, y d. Especificación de requisitos.

Dentro del proceso de clasificación en general, los requisitos se pueden agrupar en requisitos funcionales y no funcionales. Mientras que los requisitos funcionales están relacionados con los servicios que presta el software, los requisitos no funcionales son conocidos como los aspectos o atributos de calidad de un software.

Respecto a los requisitos no funcionales, (ISO 2011) define un modelo de calidad del producto de software compuesto por ocho características (que se subdividen en subcaracterísticas) que proporcionan una terminología para especificar, medir y evaluar la calidad del producto de software. También, proporcionan un conjunto de características de calidad con las que se pueden comparar los requisitos de calidad que se hayan establecido del producto de software particular, con el propósito verificar su cumplimiento.

Las ocho características generales son:

- Adecuación funcional
- Eficiencia de desempeño
- Compatibilidad
- Usabilidad
- Fiabilidad
- Seguridad
- Mantenibilidad
- Portabilidad

Lo anterior, será el modelo referente para la clasificación de los requisitos no funcionales en este trabajo.

Como se mencionó en la Sección de Introducción, un paso fundamental en la ingeniería de requisitos es el proceso de su clasificación. Si bien, este proceso se puede realizar de forma manual, los avances en las técnicas de aprendizaje automático, especialmente las relacionadas con PLN, han permitido automatizar el proceso de clasificación. Por ejemplo, (Fong 2018) implementa una metodología completa de clasificación de requisitos de software haciendo uso de redes neuronales artificiales. La sección 5 detalla cómo se han abordado este tipo de tareas a lo largo del tiempo.

5. REVISIÓN DE LA LITERATURA

El proceso de clasificación de requisitos en funcionales y no funcionales se puede abordar como un problema de clasificación de texto que puede solucionarse con técnicas de aprendizaje automático. En ese sentido, Korde (2012), realizó un estudio de la literatura sobre las metodologías empleadas para cada una de las fases dentro del proceso de generar un modelo que permita clasificar textos en un contexto determinado. En su revisión, se encontró que la metodología general para este tipo de problemas incluye la recolección y pre-procesamiento de la información, la selección de algoritmos de clasificación, entrenar los modelos y la medición del desempeño de dichos algoritmos. El autor concluye que, dentro de las técnicas de pre-procesamiento de texto, existe una vectorización de los documentos mediante el uso de técnicas como BoW *(Bag of Words, por sus siglas en inglés)*, en donde cada palabra dentro del documento tiene un valor numérico asociado. Adicionalmente, resaltó el uso de algoritmos de clasificación como las máquinas de vectores de soporte, los árboles de decisión, y las redes neuronales artificiales, encontrando a las máquinas de vectores de soporte como la técnica que obtuvo mejores resultados.

En ese orden de ideas, Kowsari et al. (2019) presentan una discusión del estado del arte sobre las metodologías utilizadas para abordar problemas de clasificación de textos, los autores hacen alusión a nuevos métodos para la extracción de la información, en los que se incluye el uso de técnicas como Word2Vec, GloVe, y otro tipo de *embeddings*, así como técnicas basadas en la frecuencia de surgimiento de las palabras dentro de la base de datos. De igual forma, los autores introducen el uso de técnicas de reducción de dimensionalidad de las variables, con técnicas como el análisis de componentes principales (PCA), discriminante lineal (LDA), y matrices de factorización negativa (NMF), con la finalidad de obtener un mejor desempeño y reducir los tiempos de entrenamiento de los algoritmos de clasificación. Por último, amplían la gama de técnicas de clasificación que se utilizan para este tipo de problemas, agregando modelos de aprendizaje profundo como las redes neuronales recurrentes, redes neuronales convolucionales y *autoencoders*, así como modelos tipo *ensemble*, con metodologías como *bagging y boosting.* Los autores exponen las ventajas y limitantes de cada metodología, así como sus aplicaciones en el área de la salud (para la clasificación de pacientes en función del reporte médico), academia (en el resumen de libros), ciencias sociales (para la trazabilidad de la semántica de los lenguajes), organizaciones (para la creación de sistemas de recomendaciones), y en el entorno público (para la optimización de procesos). Así, una de las aplicaciones de estos modelos será el de clasificar los requisitos de desarrollo de software, objetivo principal del presente trabajo.

Al-Azani y El-Alfy (2017) realizaron una aplicación específica del uso de algoritmos de aprendizaje automático para problemas de clasificación de lenguaje natural. Los autores usan Word2Vec para el *embedding* en modelos tipo *ensemble (Random Forest, Bagging, Boosting, AdaBoost, Gradient Boosting y Stacking)* con la finalidad de clasificar los sentimientos de los usuarios sobre una base de *tweets* de Syria en el lenguaje árabe. Los autores encontraron que la base de datos estaba desbalanceada, por lo que aplicaron técnicas de sobremuestreo, en concreto *SMOTE (Synthetic Minoritary Over-sampling Technique)*, con la finalidad de aumentar el desempeño de los modelos. Los resultados de los autores mostraron que al combinar las tres técnicas dentro del modelo de trabajo se logró un aumento medio del 15% sobre el F1 Score *Score* en comparación con los modelos clásicos utilizados como la base del estudio.

Por otra parte, en la última década el uso de los *Transformers con mecanismos de atención* se ha convertido en el tipo de modelos más utilizado para la solución de tareas relacionadas con el procesamiento de lenguaje natural (Wolf et al. 2020). Se han desarrollado diferentes modelos pre entrenados como BERT, GPT-3 , RoBERTa, entre otros, los cuales mediante *fine tuning* permiten adaptarse a cualquier problema específico obteniendo mejores resultados en la solución de tareas que los modelos de aprendizaje automático o aprendizaje profundo tradicionales.

Como se puede observar, existen metodologías muy bien definidas para abordar problemas de aprendizaje automático en la clasificación de textos, por lo que se tiene una línea metodológica base para abordar la clasificación de requisitos de software.

Con respecto a la aplicación de técnicas de aprendizaje automático en el problema de clasificación de requisitos de software descritos en lenguaje natural, Knauss y Ott (2014) hacen uso de un modelo de máquinas de vectores de soporte para clasificar en categorías de 2000 requisitos funcionales y de interfaz en diferentes partes de dos sistemas de automóviles Mercedes-Benz, un sistema de iluminación exterior y un sistema de control de velocidad (por ejemplo, categorías como: velocidad, encendido o comunicación). Para esto, los autores utilizaron una metodología de transformación de palabras basada en *k-grams*. Los resultados muestran que la metodología implementada genera un proceso automático de clasificación de requisitos con hasta un 70% en los indicadores de desempeño *recall y precisión*. Adicionalmente, los autores hacen hincapié en la reducción de los tiempos de proceso de esta tarea, en donde se tenía un proceso manual con un tiempo mayor a una hora, a menos de seis minutos con el modelo propuesto.

Navarro-Almanza et al. (2018) utilizaron algoritmos de aprendizaje profundo para la clasificación de requisitos de software. En su trabajo, los autores hicieron uso de una arquitectura de redes neuronales convolucionales en conjunto con el método de *embeddings Word2Vec* para un repositorio público de requisitos de software denominado ***PROMISE*** (Shirabad 2005)***.*** Los autores encontraron que la arquitectura propuesta clasificó correctamente la información, mostrando que el uso de este tipo de modelos puede clasificar información sin necesidad de preprocesar la base de datos. De igual forma, los autores hacen alusión a utilizar modelos basados en redes neuronales recurrentes para obtener un mejor desempeño en la clasificación. Por esta razón, Liao et al. (2020) utilizaron una arquitectura de aprendizaje profundo basado en redes neuronales convolucionales y redes neuronales recurrentes con la finalidad de resolver un problema de secuencia a secuencia para el análisis de sentimientos.

Por su parte, Lima et al. (2019) realizaron una expansión de 344 requisitos (55%) adicionales al conjunto de datos ***PROMISE*** (originalmente de 625 requisitos) para ponerlo a disposición de la comunidad científica. Mencionan la escasez de conjuntos de datos de requisitos de software disponibles y pertinentes en cantidad y calidad de muestras para realizar estudios que apliquen técnicas de aprendizaje automático. Su propuesta incorpora nuevos requisitos de software a fin de garantizar compatibilidad y calidad de los nuevos datos ingresados, por tanto, aplicaron técnicas de aprendizaje automático supervisado para validar la expansión del conjunto de datos. Los autores realizaron tres evaluaciones aplicando técnicas de árboles de decisión, k-NN, Naive Bayes y Máquina de Vectores de Soporte. La primera, clasificación binaria de requisitos funcionales y no funcionales; la segunda, clasificación de las categorías de los requisitos no funcionales y; la tercera,

clasificación de requisitos funcionales y las categorías de los no funcionales, logrando una mejora discreta pero prometedora en la clasificación debido al aumento de muestras. No obstante, menciona que, a pesar del aumento de la información, el número de requisitos adicionales no fue suficiente para equilibrar el conjunto de datos, lo que representa uno de los aspectos negativos que podrían influir en la validez de su estudio.

De igual forma, Hey et al. (2020) utilizaron una versión ampliada y re-etiquetada del conjunto de datos ***PROMISE*** proporcionada por Dalpiaz et al. (2019), que consta de 1502 requisitos. Los autores mencionan que el desempeño de los enfoques existentes de clasificación automática disminuye cuando los requisitos varían en redacción y estilo, pues a pesar de que existen estudios previos con modelos de resultados prometedores, son poco prácticos en su uso ya que están sobreajustados al conjunto de datos, dependiendo en gran medida de la redacción y la estructura de la oración o que necesiten un procesamiento manual previo, ocasionando que la capacidad de generalización no sea suficiente para aplicar los modelos a otros proyectos de diferentes dominios y redacción. Asimismo, mencionan que un factor de la situación podría ser por la falta de disponibilidad de datos para aplicar las técnicas de aprendizaje automático en la comunidad de ingenieros de requisitos. Frente a lo anterior, su propuesta aplica la técnica BERT, un modelo de lenguaje previamente entrenado con un gran conjunto de datos y que ajustaron para mejorar el rendimiento y mejorar la capacidad de generalización con menos datos, específicamente en la clasificación automática de requisitos funcionales y no funcionales. Los autores obtuvieron resultados convincentes que se pueden aplicar a proyectos con diferentes dominios y redacción.

Rajender Kumar Surana et al. (2019) abordaron la clasificación de los requisitos de manera más directa. Los autores generaron un *chatbot* con una arquitectura de aprendizaje profundo basado en redes neuronales convolucionales y recurrentes. La propuesta de los autores permite al *chatbot* conversar con los encargados de establecer los requisitos de software y este convierte el audio del encargado en texto y, simultáneamente, lo clasifica como un requisito funcional o no funcional. Los autores resaltaron que la propuesta permite recolectar la información de los requisitos de manera más natural y amigable con el usuario y, adicionalmente, obtiene una correcta clasificación de los requisitos.

Por otra parte, Mahmoud y Williams (2016) proponen una metodología diferente para la detección y clasificación de los requisitos no funcionales. Los autores utilizaron técnicas aprendizaje no supervisado y se basaron en el supuesto principal que los requisitos no funcionales tienden a ser impuestos implícitamente por los requisitos funcionales del sistema, de manera que dicho conocimiento se modeló y capturó para generar *clusters* que permitieran encontrar el conjunto de palabras clave que identifican a un requisito no funcional. Los autores concluyeron que el uso de algoritmos basados en *clústering jerárquico* obtuvieron mejores resultados que los basados en K-centroides, con indicadores de desempeño lo suficientemente relevantes para ser usado en aplicaciones prácticas.

En la misma línea, Arora et al. (2017) proponen una metodología basada en modelos de aprendizaje no supervisado que permite agrupar y ordenar por relevancia los términos dentro de un requisito de software con la finalidad de determinar si este es funcional o no funcional. Los autores utilizan una combinación de métricas para interpretar la similitud de la semántica y gramática y, en conjunto con algoritmos basados en K-centroides, obtuvieron

resultados hasta un 20% por encima de herramientas comerciales que realizan la misma tarea en idioma inglés.

Estos dos últimos trabajos permiten tener una visión diferente de cómo se puede abordar el problema de clasificación de requisitos, debido a que no es impositivo que deba ser mediante el uso de algoritmos de aprendizaje supervisado, sino que, al hacer uso de algoritmos de aprendizaje no supervisado, se puede ampliar el espectro de aplicación al omitir la necesidad de etiquetar la base de datos de entrenamiento.

Vogelsang y Borg (2019) realizan un primer acercamiento en explorar cómo los científicos de datos realizan el proceso de ingeniería de requisitos en el diseño e implementación de sistemas basados en aprendizaje automático. Si bien, los autores indagan si en el proceso de creación de sistemas basados en aprendizaje automático se utilizan las mismas metodologías de ingeniería de requisitos para la definición de especificaciones del software del cliente, los autores encuentran que hay una gran variedad de nuevos conceptos como explicabilidad, libertad de discriminar decisiones de diseño, e incluso ámbitos legales, que actualmente no se utilizan para un proceso de ingeniería de requisitos en otro tipo de sistemas. Esta documentación es relevante debido a que, si bien el presente trabajo se basará en cómo clasificar los requisitos de desarrollo de software, se puede extender para futuras investigaciones la aplicación de dichas metodologías en el propio diseño de un sistema basado en aprendizaje automático ya que, como lo explican los autores, tiene incluso más parámetros que abordar.

Si bien, se ha observado cómo se tienen diferentes perspectivas para la clasificación de los requisitos de software, de manera general, se opta por utilizar modelos de aprendizaje supervisado para la resolución del problema de estudio. (Binkhonain y Zhao 2019). Esto, en conjunto con las correctas técnicas de pre-procesamiento de texto, puede obtener una mejora significativa con respecto a los tiempos de clasificación de los requisitos de software (Iqbal, Elahidoost y Lucio 2018). No obstante, Gramajo, Ballejos y Ale (2019) resaltan que, si bien hay un amplio estudio en este campo, existe una escasez en conjuntos de datos de requisitos de software y de estudios enfocados a la clasificación de requisitos de software en el idioma español, lo cual propone una brecha de investigación que se intenta suplir con el presente trabajo.

Así, con un conjunto de datos de entrada preprocesado, se abordan diferentes técnicas de clasificación automática mediante el uso de algoritmos previamente entrenados (Hey et al. 2020), (Vaswani et al. 2017; Wolf et al. 2020) y (Lee y Hsiang 2020), o el entrenamiento de modelos propios.

6. DESARROLLO DE LA METODOLOGÍA

La metodología para desarrollar este trabajo se planteó con base en lo encontrado en la revisión de literatura presentada en la Sección 5. En primera instancia, se adquirió la información y se consolidó el conjunto de datos de requisitos de software en inglés ampliamente utilizados en otras investigaciones. Luego, se tradujo al español con la herramienta de traducción automática de Google y se verificó manualmente la calidad del texto generado. Después, se realizó el análisis exploratorio del comportamiento de la variable de respuesta. Finalmente, se generaron 5 experimentos que se describen a continuación, donde se inició con la idea de generar los 3 primeros experimentos con el fin de comparar los resultados y obtener conclusiones a partir de esas variaciones; una vez realizado ese análisis se decidió analizar también las posibilidades de los experimentos 4 y 5:

- **Experimento 1:** Generar el entrenamiento, optimización de hiperparámetros y evaluación final de los modelos seleccionados para un proceso de clasificación multiclase en 7 categorías, 6 correspondientes a las etiquetas de los requisitos no funcionales y una para el grupo de requisitos funcionales.
- **Experimento 2:** Generar el entrenamiento, optimización de hiperparámetros y evaluación final de los modelos seleccionados para un proceso de clasificación binario en requisito funcional o no funcional.
- **Experimento 3:** Generar el entrenamiento, optimización de hiperparámetros y evaluación final de los modelos seleccionados para un proceso de clasificación multiclase únicamente de requisitos no funcionales en sus seis categorías detalladas.
- **Experimento 4:** De acuerdo con los resultados obtenidos en el experimento 1 en la clasificación del requisito funcional, se propone otra opción con un experimento que combina modelos así: se entrenan de manera independiente un modelo para el problema binario (experimento 2) y uno para el problema multiclase sólo para los requisitos no funcionales (experimento 3). Una vez entrenados, se crea un flujo donde se prueban de manera combinada, de forma que primero se prediga si el requisito es o no funcional y, en caso de que la predicción sea no funcional, se prediga el detalle del tipo de requisito no funcional.
- **Experimento 5:** Para obtener resultados comparativos con el modelo final del experimento 1, se realiza un proceso de *fine tuning* sobre GPT-3, un modelo de lenguaje natural pre entrenado que es tendencia en sistemas de procesamiento de lenguaje natural (NLP), aquí se utilizó la versión beta privada de (Brown et al. 2020) que tiene la limitación de máximo 200 muestras en el conjunto de datos y la evaluación del desempeño se realizó a partir del conjunto de datos de prueba.

En la *Figura 2*, se muestra la metodología general que se aplicó en el trabajo para los experimentos 1, 2 y 3, allí se realizan dos subprocesos: el primero, aplicado a la base de datos sin generar datos artificiales; el segundo, aplicado a la base de datos aumentada por medio de SMOTE. En cada fase de entrenamiento (defecto, optimización de hiperparámetros y evaluación final) se realiza validación cruzada con 5 folds para medir el desempeño del modelo.

De acuerdo con Bissuel 2019, la optimización de hipeparámetros hace referencia al ajuste de las características de un modelo de aprendizaje automático, entendiendo que dichas características son definidas al momento del entrenamiento del modelo. En este proceso

se utilizó la técnica de búsqueda por grilla (Grid Search), donde se establecen una serie de subconjuntos de valores para los parámetros a optimizar, teniendo como métricas de desempeño la eficiencia del modelo con cada combinación de parámetros sobre la base de prueba.

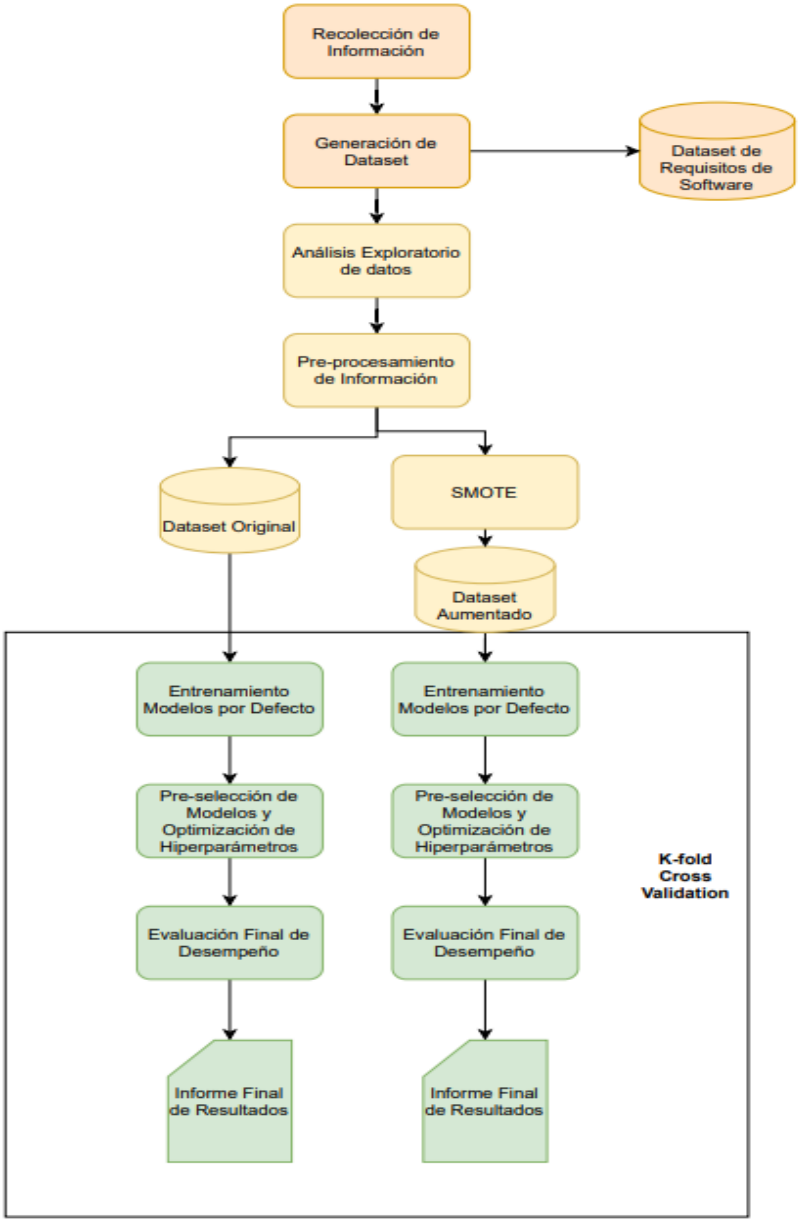

Figura 2. Metodología General, experimentos 1, 2 y 3. Fuente: Elaboración propia.

En la *Figura 3*, se muestra la metodología general que se aplicó en el trabajo para el experimento 4. Allí se entrenan los modelos binarios y multiclase de manera independiente y se realiza validación cruzada de 5 folds con una serie de registros que son comunes para ambos procesos. De esta manera, si el modelo binario indica que el requisito es funcional, el modelo multiclase no realizará ninguna predicción. Por el contrario, si el modelo binario predice un requisito no funcional, el modelo multiclase determinará la categoría del requisito.

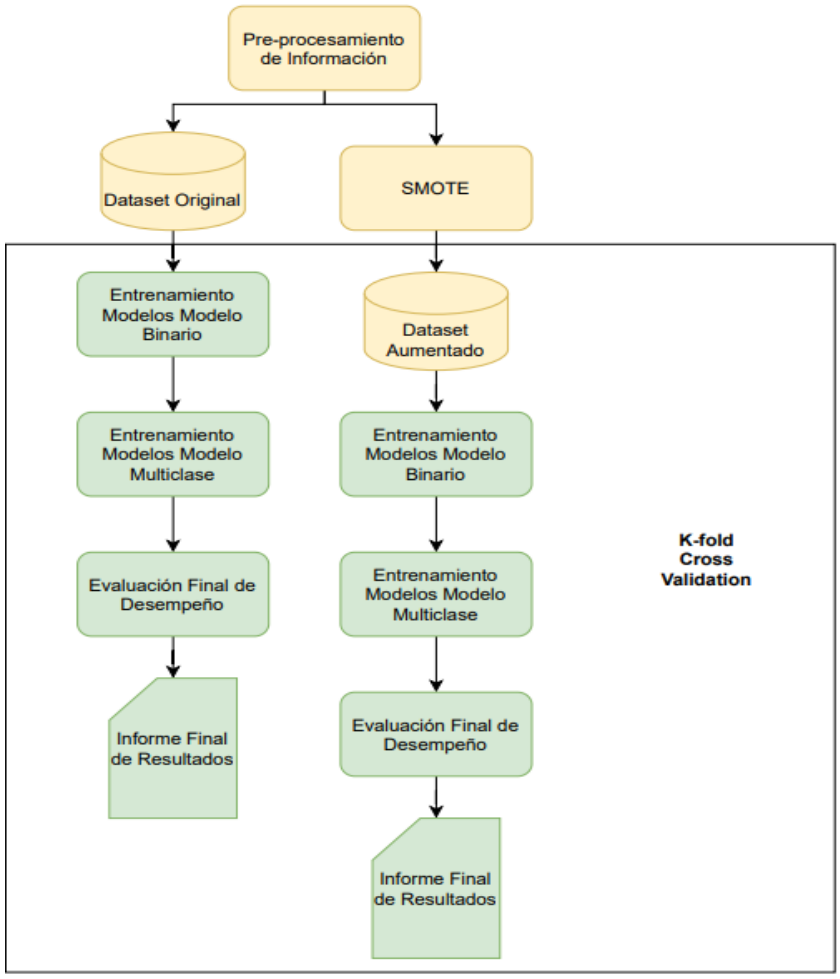

Figura 3. Metodología general, experimento 4. Fuente: Elaboración propia.

Para el experimento 5, debido a las limitaciones que tiene la API de GPT-3, no se realizó el proceso de validación cruzada, sino que se entrenó al modelo con un subconjunto de 200 registros distribuidos de manera equitativa.

6.1. RECOLECCIÓN DE INFORMACIÓN Y GENERACIÓN DEL CONJUNTO DE DATOS

La información de los requisitos de software en español se consolida a partir de cuatro conjuntos de datos descritos en inglés que fueron traducidos con la herramienta automatizada de Google, teniendo en cuenta que es uno de los traductores automáticos más utilizados (Ornat 2015), de fácil uso, acceso (Restrepo Klinge 2019) y que se ha obtenido resultados satisfactorios en la traducción de inglés a español (Contributions et al. 2019) y (Taira et al. 2014). Posteriormente, los requisitos traducidos fueron verificados y ajustados manualmente. Esta actividad termina con la construcción de un conjunto de datos en español que contiene 2858 requisitos de software de diferentes dominios de software, distribuidos así:

1. El primer conjunto de datos de 1502 requisitos suministrado por Dalpiaz et al. (2019) consta de dos subconjuntos de datos:

a. 877 requisitos que Dalpiaz et al. (2019) etiquetaron en clases funcional y no funcional, los cuales se etiquetaron manualmente para este trabajo en requisito funcional y las categorías de requisitos no funcionales definidas en el estándar ISO/IEC 25010.
b. 625 requisitos del tradicional dataset PROMISE que Dalpiaz et al. (2019) re-etiquetaron en clases funcional y no funcional y que previamente estaban etiquetados en requisito funcional y las categorías de requisitos no funcionales. El cual se ajustó para este trabajo en algunas de las etiquetas con las categorías consideradas en el estándar ISO/IEC 25010.

2. El tercer conjunto de datos de 1005 requisitos extraídos de los archivos XML de Ferrari, Spagnolo y Gnesi (2017), los cuales fueron etiquetados manualmente para este trabajo en requisito funcional y las categorías de requisitos no funcionales definidas en el estándar ISO/IEC 25010.
3. El cuarto conjunto de datos de 346 requisitos que Lima et al. (2019) etiquetaron en requisito funcional y las categorías de requisitos no funcionales. El cual se ajustó para este trabajo en algunas de las etiquetas con las categorías definidas en el estándar ISO/IEC 25010.

La distribución de los requisitos por cada conjunto de datos se observa en la Figura 4:

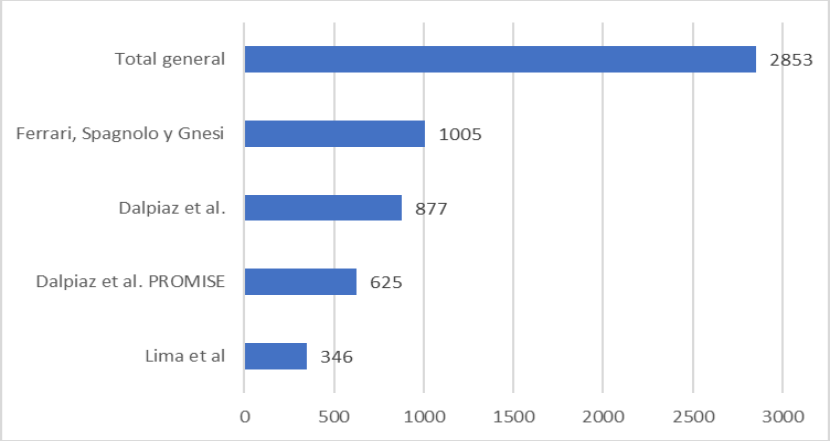

Figura *4*. Cantidad total de requisitos y por cada dataset. Fuente: Elaboración propia.

Se determinaron 2 etiquetas del conjunto de datos para identificar las dos grandes categorías de requisitos así: F para los requisitos funcionales y NF para los requisitos no funcionales. Asimismo, se determinaron 8 etiquetas en el conjunto de datos generado, según el estándar ISO/IEC 25010, así: USABILIDAD, SEGURIDAD, PORTABILIDAD, MANTENIBILIDAD, FIABILIDAD, DESEMPEÑO, COMPATIBILIDAD, FUNCIONAL para identificar las categorías de requisitos no funcionales y el requisito funcional.

También, se generaron 31 etiquetas para las subcategorías de los requisitos no funcionales, las cuales se procesaron manualmente, sin embargo, no se considerarán por el alcance de este trabajo y dado que el desglose en las subcategorías significa menor número de muestras para cada una y que disminuye las posibilidades de obtener desempeños aceptables en la clasificación de los modelos. El etiquetado de las 31 subcategorías quedará en el conjunto de datos a disposición de la comunidad científica para futuros

problemas que consideren la ampliación y diversidad de requisitos en el conjunto de datos, promoviendo estudios más ricos en el aprendizaje automático y la ingeniería de software.

El sistema de clasificación de las categorías de requisitos no funcionales del conjunto de datos PROMISE re-etiquetado de Dalpiaz et al. (2019) y el etiquetado manualmente de (Lima et al. 2019), se ajustó en este trabajo en algunas de las etiquetas para generar la analogía con las categorías consideradas en el estándar ISO/IEC 25010. La relación de las etiquetas de PROMISE con su significado vs etiquetas nuevas se detalla en el Cuadro 1:

Etiqueta PROMISE	Etiqueta Nueva Categoría	Etiqueta Nueva Subcategoría
A (AVAILABILITY)	FIABILIDAD	DISPONIBILIDAD
FT (FAULT TOLERANCE)	FIABILIDAD	TOLERANCIA A FALLOS
L (LEGAL)	FUNCIONAL	FUNCIONAL
LF (LOOK & FEEL)	USABILIDAD	ESTÉTICA
MN (MAINTAINABILITY)	MANTENIBILIDAD	SEGÚN SUBCATEGORÍA
O (OPERATION)	COMPATIBILIDAD	SEGÚN SUBCATEGORÍA
PE (PERFORMANCE)	DESEMPEÑO	SEGÚN SUBCATEGORÍA
PO (PORTABILITY)	PORTABILIDAD	SEGÚN SUBCATEGORÍA
SC (SCALABILITY)	MANTENIBILIDAD	CAPACIDAD DE SER MODIFICADO
SE (SEGURITY)	SEGURIDAD	SEGÚN SUBCATEGORÍA
US (USABILITY)	USABILIDAD	SEGÚN SUBCATEGORÍA

Cuadro 1. Sistema de clasificación de etiquetas ajustadas.

Respecto a las clases de requisitos funcionales (F) y no funcionales (NF). El conjunto de datos con mayor número de requisitos de la clase funcional es (Ferrari, Spagnolo y Gnesi 2017) y con mayor número de clase no funcional es Dalpiaz et al. (2019) en el subconjunto PROMISE, como se observa en la Figura 5:

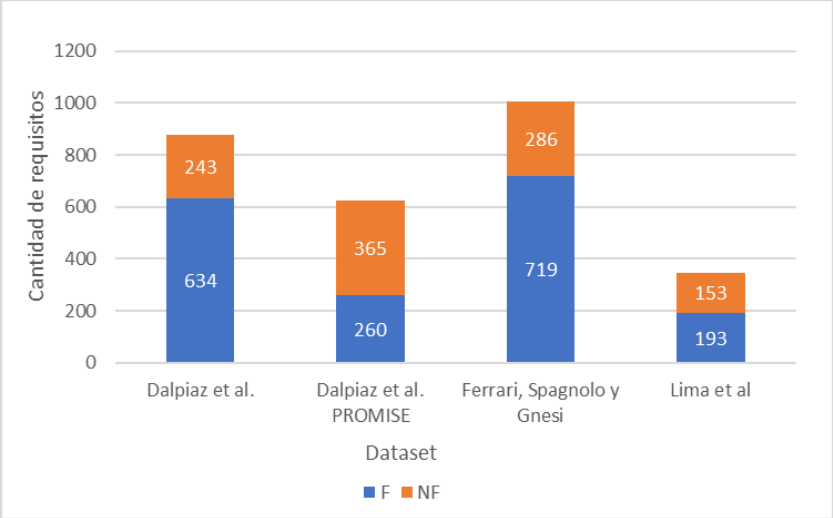

Figura *5*. Distribución de requisitos de las clases funcional y no funcional por cada conjunto de datos. Fuente: Elaboración propia

Asimismo, en la Figura 6, se observan las cantidades de requisitos de las 7 categorías de requisitos no funcionales por cada conjunto de datos:

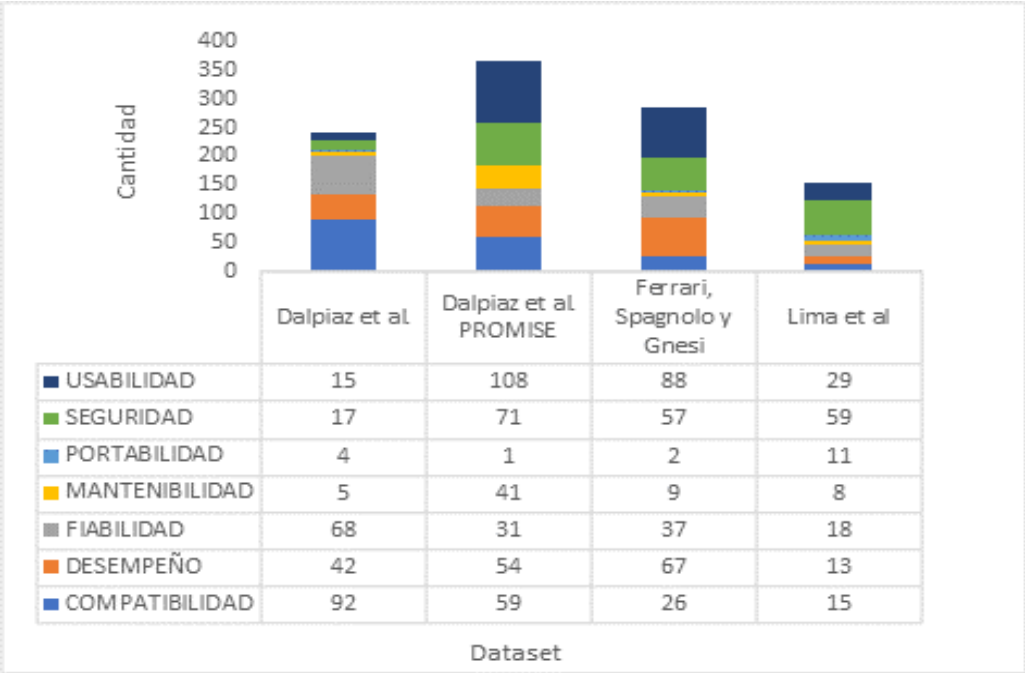

Figura *6*. Distribución de requisitos de las categorías de la clase no funcional por cada conjunto de datos.

En un experimento inicial que no se incluye en este documento porque se estaba explorando la metodología que se iba a aplicar, se notó el efecto de las clases MANTENIBILIDAD Y PORTABILIDAD con tan pocos datos, 63 y 18 muestras respectivamente, donde los modelos no obtuvieron más de 20% en el promedio general del indicador F1. Por tanto, se propuso unificar en una categoría OTROS a las categorías de los requisitos de MANTENIBILIDAD y PORTABILIDAD, de esta forma, fueron 81 requisitos en esta nueva categoría y pasaron de ser 7 categorías a 6 categorías de requisitos no funcionales así: USABILIDAD, SEGURIDAD, OTROS, FIABILIDAD, DESEMPEÑO Y COMPATIBILIDAD.

Por otra parte, en el Cuadro 2, se describen las variables de los conjuntos de datos con que se entrenaron los modelos predictivos:

Variable	Tipo de Variable	Descripción
Traducción - google_trans_new	Cadena	Hace referencia a la traducción al español del campo *RequirementText* mediante la API del traductor de Google.
Class	Cadena	Hace referencia a la clasificación de requisito de software en tipo clase funcional y categorías de no funcionales de acuerdo con el estándar ISO/IEC 25010.
ClassBinary	Cadena	Hace referencia a la clasificación de requisito de software en funcional (F) o no funcional (NF).

Cuadro 2. Variables de entrenamiento del conjunto de datos.

Asimismo, en el Cuadro 3, se describen las variables que no se incluyeron en el conjunto de datos de entrenamiento y que se registraron para relacionar el texto original en inglés, el proyecto, el conjunto de datos desde donde se obtuvieron y la subcategoría de requisito no funcional:

Variable	Tipo de Variable	Descripción
RequirementText	Cadena	Descripción del requisito de software en lenguaje natural de idioma inglés.
ProjectID	Cadena	Hace referencia al identificador en cada dataset del proyecto de donde se tomó el requisito de software.
Dataset	Cadena	Hace referencia a los autores del dataset de donde se extrajo originalmente el requisito de software.
ChildrenClass	Cadena	Hace referencia a la clasificación de requisito de software en tipo clase funcional y subcategorías de no funcionales de acuerdo con el estándar ISO/IEC 25010.

Cuadro 3. Variables complementarias del conjunto de datos.

De acuerdo con lo descrito en este capítulo, se generó el conjunto de datos que fue utilizado para el entrenamiento de los modelos en los diferentes experimentos.

6.2. ANÁLISIS EXPLORATORIO DE DATOS

De acuerdo con Nasukawa y Yi (2003), un análisis exploratorio de la información en los problemas asociados al procesamiento de lenguaje natural permite determinar las técnicas de preprocesamiento de la información, así como un primer acercamiento de los modelos de aprendizaje automático a utilizar.

En el desarrollo del experimento 1, donde se tomaron los requisitos de tipo funcional y las 6 categorías de tipo no funcional, se observó que la clase funcional predomina en el conjunto de datos, lo que puede representar dificultades para los modelos en cuanto a la generalización de conceptos asociados a requisitos que no son funcionales y, en mayor medida, a aquellas categorías con muy poca cantidad de registros. En la Figura *7*, se muestra la distribución de requisitos de la base de datos.

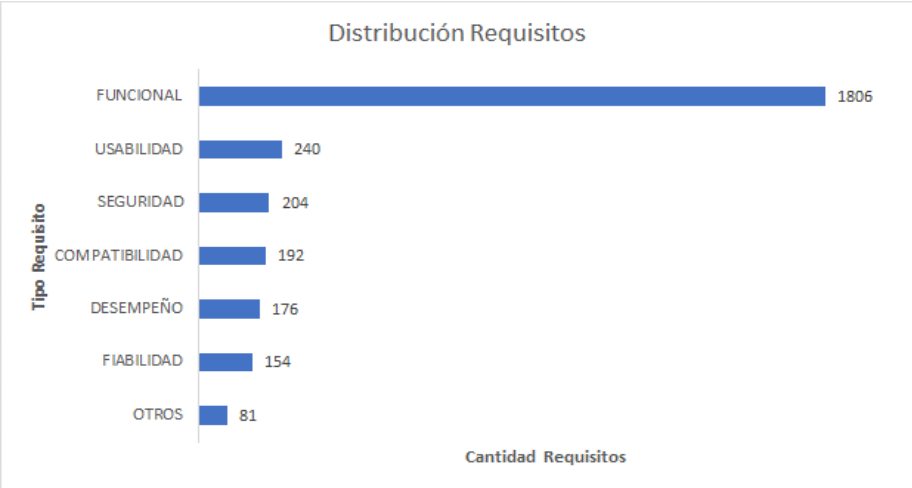

Figura 7. Distribución de requisitos, experimento 1.

En el desarrollo del experimento 2, donde se tomaron solo dos tipos de requisitos, funcionales (F) o no funcionales (NF), se observó una situación similar a la del experimento 1, pues la clase funcional predomina tal y como se presenta en la Figura *8*:

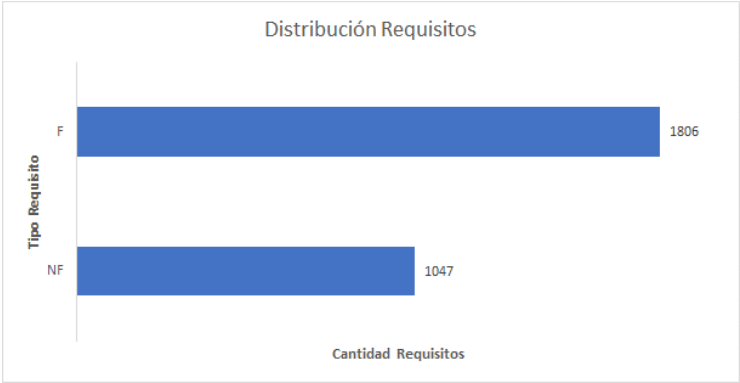

Figura 8. Distribución de requisitos, experimento 2.

Por otra parte, para el experimento 3, la distribución de las categorías de requisitos no funcionales se observa en la Figura *9*, donde la categoría USABILIDAD tiene la mayor cantidad de muestras:

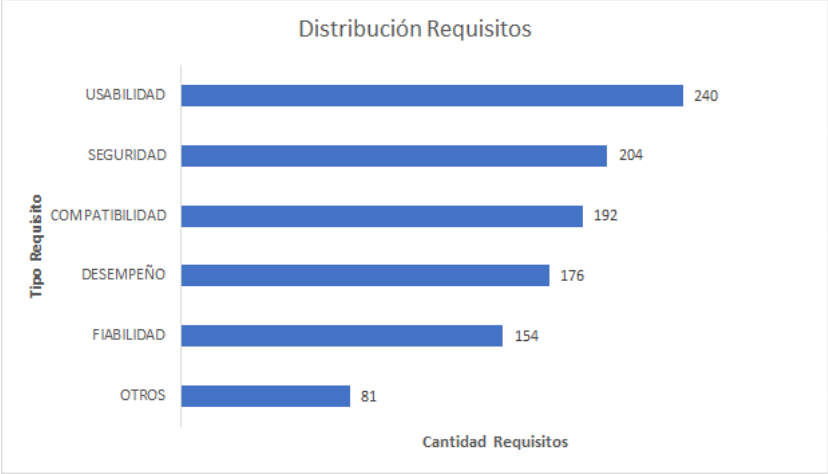

Figura 9. Distribución de requisitos, experimento 3.

Para el desarrollo del experimento 4, donde se tiene la combinación de los modelos a través de un flujo, se utilizó el conjunto de datos del experimento 2 y 3 respectivamente.

Por último, para el experimento 5, se utilizó un sub-conjunto de datos de 200 registros de entrenamiento, haciendo un muestreo que garantizara todas las clases de requisitos en el conjunto de datos de entrenamiento, de esa manera, hubo 27 requisitos por cada clase, excepto para PORTABILIDAD (18 requisitos) y USABILIDAD (20 requisitos).

6.3. PREPROCESAMIENTO DE LA INFORMACIÓN

Los requisitos están escritos principalmente en lenguaje natural y pueden aparecer en una variedad de formas, como listas de palabras individuales, oraciones, párrafos, textos cortos

que pueden incluir caracteres especiales u otros. Antes de aplicar un algoritmo de aprendizaje automático en dichos datos, se emplean diferentes pasos para transformar palabras en características, como es el procesamiento de lenguaje natural (NLP).

Por tanto, los requisitos de software traducidos al español se preprocesaron realizando las siguientes actividades mencionadas en (Iqbal, Elahidoost y Lucio 2018):

- Limpieza de texto: remover letras mayúsculas para establecer todos los caracteres en letra minúscula y simplificar (capitalización). También remover símbolos, reemplazándolos por espacios en blanco.
- Normalización de texto: mediante el uso de lematización, remover palabras de parada (stopwords), el uso del stemming POS tagging, con la finalidad de eliminar palabras habituales que no aportan significado y tener mayor eficiencia en el entrenamiento de los modelos al reducir variaciones en el vocabulario y evitando clasificaciones de falsos negativos (Gulati et al. 2021). Por tanto, se utilizó el Word Corpus en español de (Bird y Tan 2021) que elimina artículos, preposiciones y conjunciones.

Se utilizaron las siguientes técnicas para el proceso de conversión de texto a matrices:

- Vectorización por conteo de palabras (Sklearn 2021a), creando un diccionario del vocabulario del corpus, en donde a cada palabra dentro de cada requisito se le asigna un valor numérico en función de la frecuencia de la misma en el requisito.
- Vectorización por Frecuencia Inversa de Documento (Sklearn 2021b) (*TFID por sus siglas en inglés)*, en donde se toman los textos vectorizados por el conteo de palabras y se normalizan teniendo en cuenta su frecuencia en todo el dataset.

6.4. GENERACIÓN ARTIFICIAL DE REQUISITOS

No obstante, de acuerdo con la metodología planteada, se realizó aumento del conjunto de datos con muestras artificiales mediante la técnica *SMOTE*, de manera que durante el entrenamiento de los modelos se les otorgue más información sobre las demás clases diferentes a la funcional y así, puedan mejorar su desempeño. La nueva distribución de requisitos para el experimento 1 se muestra en la Figura *10*:

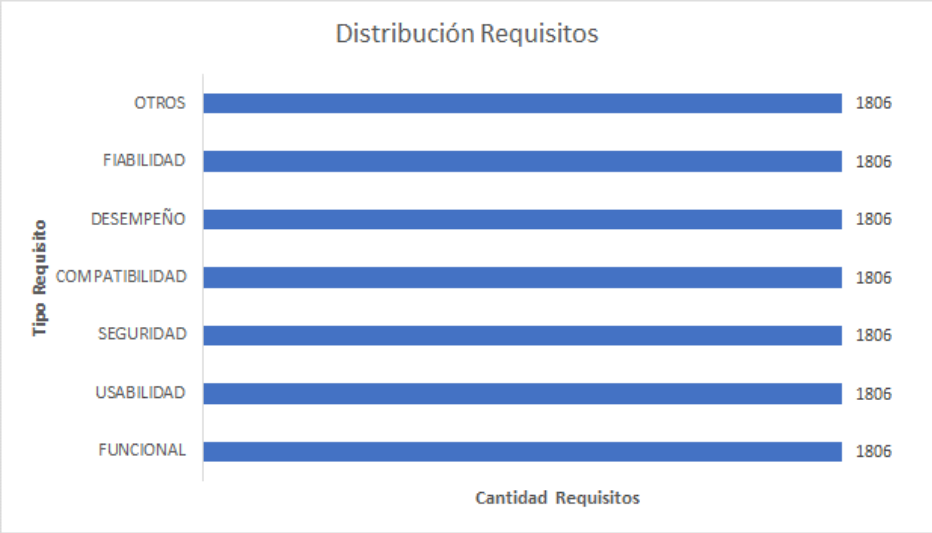

Figura 10. Distribución de requisitos aumentados, experimento 1.

Para el experimento 2, al igual que en el experimento 1, se generó un balanceo de clases mediante la técnica *SMOTE* para tener igual cantidad de registros de cada clase, como se muestra en la Figura *11*:

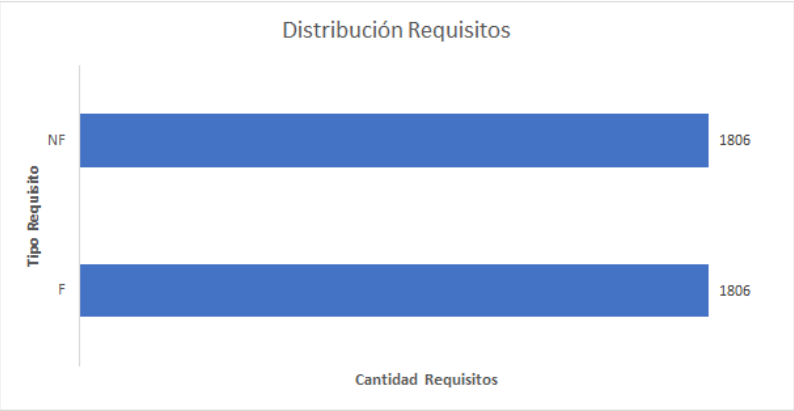

Figura 11. Distribución de requisitos aumentados, experimento 2.

Igualmente, en el experimento 3, para tener igual cantidad de requisitos de cada categoría, se aplicó la técnica SMOTE, de esta forma, la distribución de requisitos quedó como se observa en la Figura *12*:

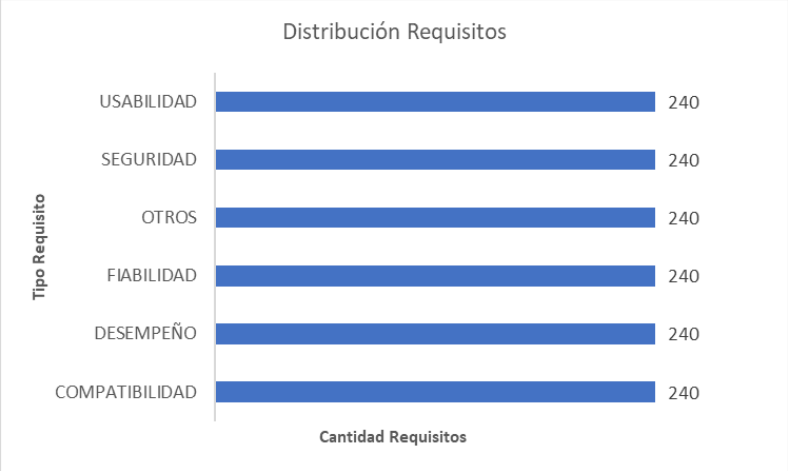

Figura 12. Distribución de requisitos aumentados, experimento 3.

Para el desarrollo del experimento 4, donde se tiene la combinación de los modelos a través de un flujo, se utilizó el conjunto de datos aumentado del experimento 2 y 3 respectivamente.

Por último, para el experimento 5, no se requirió aumento de datos.

6.5. ENTRENAMIENTO DE MODELOS

Una vez se ha realizado el preprocesamiento de la información y el aumento de datos, se seleccionaron las técnicas de entrenamiento de los modelos de aprendizaje automático. De acuerdo con lo revisado en la sección 5, existen muchas técnicas de aprendizaje automático para resolver un problema de clasificación de texto, por tanto, algunas de ellas se seleccionaron con base en lo realizado por Gulati et al. 2021:

- Máquina de Vectores de Soporte. Con el enfoque de entrenamiento SVC.
- Modelo Ensemble tipo Bagging con árboles de decisión como modelo base.
- Red Neuronal Convolucional.
- Red Neuronal Recurrente con celdas *LSTM.*
- Red Neuronal con capas convolucionales y celdas *LSTM.*
- Transformer con mecanismos de atención*.*

El entrenamiento de los modelos se realizó en la plataforma de desarrollo en línea Google Colab Pro con la versión de Python 3.7.13 y las librerías Scikit-learn 1.0.2 y Tensorflow 2.7.0

Los parámetros iniciales con los que se entrenaron los modelos son los predeterminados o por defecto incorporados en las librerías Scikit-learn y Tensorflow, los cuales se detallan en el Cuadro *4*:

Modelo	Parámetros
Máquina de Vectores de Soporte	**Penalización:** Referente al factor de penalización y regularización que aplicará el modelo a las variables de entrada, el valor por defecto es 1. **Kernel** Referente al tipo de transformación que se les aplica a los datos de entrada para que el problema se pueda resolver por medio de un clasificador lineal, el valor por defecto es *rbf* referente a función de base radial. **Gamma:** Referente al factor de escalado que tiene el kernel sobre la información de entrada, el valor por defecto se establece como el inverso del número de variables multiplicado por el inverso de la varianza de la información. **Número iteraciones:** Límite máximo de iteraciones de aplicación del algoritmo interno de optimización (entrenamiento) del modelo, el valor utilizado por defecto es infinito, de manera que algoritmo seguirá iterando hasta que el criterio de parada se cumpla.
Bagging de estimador base árboles de decisión	**Estimador base:** Referente al modelo individual al cual se aplicará el proceso de *bagging*, el modelo por defecto es un árbol de decisión. **Número de estimadores:** Referente a cuántos modelos individuales se usan en el *ensemble,* el valor por defecto es 10. **Máximas características:** Referente al porcentaje de variables de entrada que se utilizará para entrenar cada modelo individual, el valor por defecto es 100%. **Bootstrap:** Referente a si el muestreo para entrenar a cada uno de los modelos individuales se realiza con o sin remplazo de observaciones*,* el valor por defecto es *True.* **Bootstrap features:** Referente a si el uso de las variables de entrada para entrenar cada modelo puede o no tener reemplazos, el valor por defecto es *False.*

Red Neuronal Convolucional (CNN)	**Número de Neuronas CNN:** Indica el número de neuronas que tendrá cada capa convolucional, esto afectará el nivel de abstracción de la información que tenga la red, el valor por defecto es 32. **Número de Neuronas Densas:** Indica el número de neuronas que tendrán las capas densas del modelo, el valor por defecto es 256. **Batch size:** Indica la cantidad de muestras que se le pasan a la red para que esta realice el proceso de optimización de los parámetros de cada capa, el valor por defecto es 64.
Red Neuronal Recurrente con celdas LSTM	**Número de celdas LSTM:** Indica el número de grupos LSTM que tendrá cada capa recurrente, afectando la capacidad de la red para memorizar la información de corto y largo plazo, el valor por defecto es 50. **Número de Neuronas Densas:** Indica el número de neuronas que tendrán las capas densas del modelo, el valor por defecto es 100. **Batch size:** Indica la cantidad de muestras que se le pasan a la red para que esta realice el proceso de optimización de los parámetros de cada capa, el valor por defecto es 64.
Red Neuronal con capas Convolucionales y celdas LSTM	**Número de celdas LSTM:** Indica el número de grupos LSTM que tendrá cada capa recurrente, afectando la capacidad de la red para memorizar la información de corto y largo plazo, el valor por defecto es 50. **Número de Neuronas CNN:** Indica el número de neuronas que tendrá cada capa convolucional, esto afectará el nivel de abstracción de la información que tenga la red, el valor por defecto es 32. **Número de Neuronas Densas:** Indica el número de neuronas que tendrán las capas densas del modelo, el valor por defecto es 100. **Batch size:** Indica la cantidad de muestras que se le pasan a la red para que esta realice el proceso de optimización de los parámetros de cada capa, el valor por defecto es 64
Transformer con Mecanismos de Atención	**Num_heads:** Referente al total de bloques de celdas de atención que tendrá el modelo, el valor por defecto es 4. **Num_transformer_blocks:** Referente al número de bloques transformer (compuesto por capas de celdas de atención, de normalización, capas convolucionales y capas de dropout) que tendrá el modelo, el valor por defecto es 4.

Cuadro 4. Parámetros por defecto de los modelos de aprendizaje automático.

A continuación, se detallan los resultados para cada uno de los experimentos.

6.6. EXPERIMENTO 1 (PROBLEMA MULTICLASE PARA LA CATEGORÍA FUNCIONAL Y LAS 6 CATEGORIAS NO FUNCIONALES)

6.6.1. Resultados Modelos por Defecto (parámetros predeterminados)

En el Cuadro *5*. y Figura *13* se muestran los resultados de clasificación de los modelos con el conjunto de datos sin registros artificiales (original). Se puede observar que la clase predominante genera que los modelos se sesguen por predecir dicha clase con un promedio de 62% en el indicador F1:

Requisito / Modelo	Bagging	CNN	LSTM	LSTM + CNN	SVM	Transformer	Promedio
COMPATIBILIDAD	16%	6%	9%	6%	0%	4%	7%
DESEMPEÑO	18%	10%	8%	3%	0%	7%	8%
FIABILIDAD	14%	5%	5%	4%	0%	2%	5%
FUNCIONAL	77%	75%	7%	68%	78%	70%	62%
OTROS	2%	0%	2%	2%	0%	3%	2%
SEGURIDAD	16%	4%	5%	5%	0%	4%	6%
USABILIDAD	19%	8%	6%	7%	0%	5%	8%
Promedio	**23%**	**15%**	**6%**	**14%**	**11%**	**13%**	**14%**

Cuadro 5. Resultados Modelos Defecto, experimento 1, conjunto de datos original.

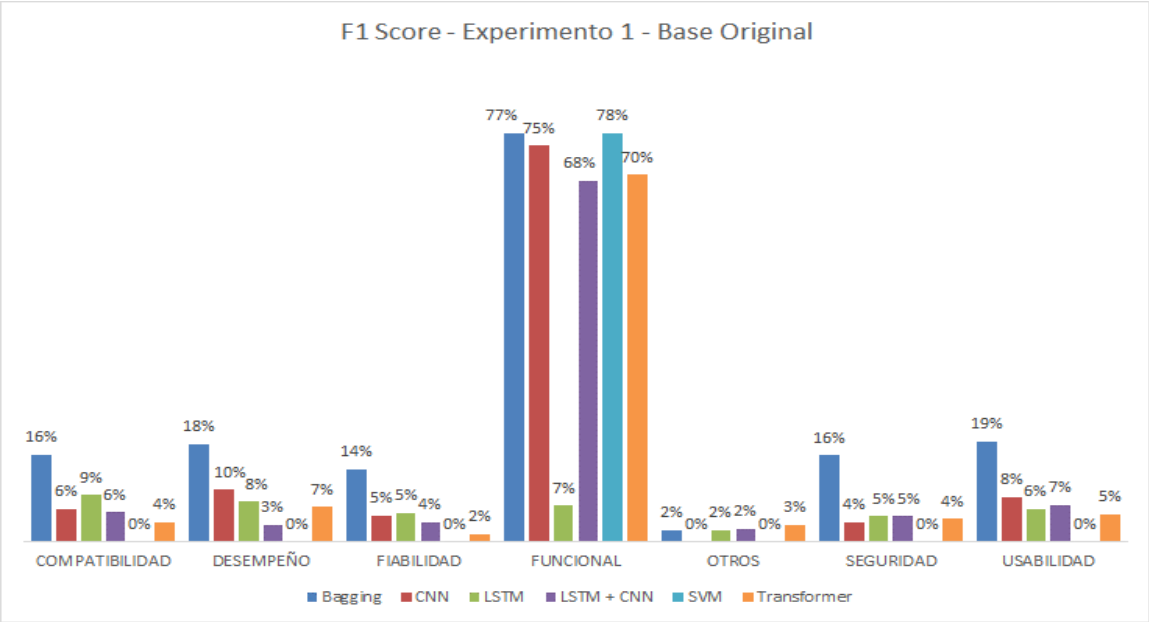

Figura 13. Desempeño modelos defecto, experimento 1, conjunto de datos original.

De acuerdo con lo anterior, se seleccionaron los modelos de Bagging y CNN que tuvieron el promedio de desempeño más alto para continuar con el proceso de optimización de hiperparámetros.

Por otra parte, los modelos mejoraron el desempeño en las clases no predominantes con el conjunto de datos aumentado tal como se muestra en el Cuadro *6* y la Figura *14*, el modelo bagging obtuvo el mejor desempeño con un promedio de 83% en el indicador F1, pero se observa que aunque mejora notablemente el desempeño en las categorías de los requisitos no funcionales, el desempeño en la clasificación del requisito funcional disminuye, pues pasa de 77% a 56% en el modelo bagging:

Requisito / Modelo	Bagging	CNN	LSTM	LSTM + CNN	SVM	Transformer	Promedio
COMPATIBILIDAD	87%	63%	3%	13%	64%	15%	41%
DESEMPEÑO	86%	66%	11%	9%	59%	14%	41%
FIABILIDAD	88%	68%	4%	12%	64%	15%	42%
FUNCIONAL	56%	35%	6%	7%	26%	21%	25%
OTROS	93%	76%	7%	21%	65%	18%	47%
SEGURIDAD	85%	60%	13%	11%	51%	13%	39%
USABILIDAD	86%	68%	11%	16%	61%	20%	44%
Promedio	**83%**	**62%**	**8%**	**13%**	**56%**	**17%**	**40%**

Cuadro 6. Resultados Modelos Defecto, experimento 1, conjunto de datos aumentado.

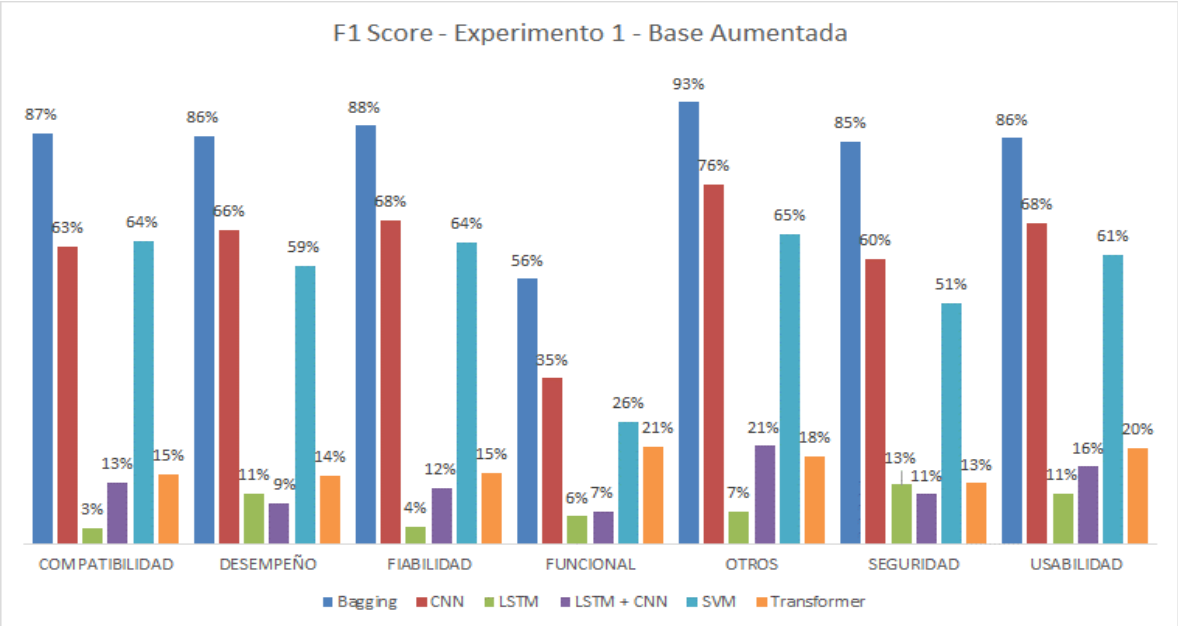

Figura 14. Desempeño modelos defecto, experimento 1, conjunto de datos aumentado.

De acuerdo con lo anterior, se seleccionaron los modelos de Bagging, CNN y SVM que tuvieron el promedio de desempeño más alto para continuar con el proceso de optimización de hiperparámetros.

6.6.2. Optimización de hiperparámetros

En el Cuadro *7* se muestran los parámetros que se optimizaron para cada modelo y el conjunto de valores o rangos para cada parámetro que se utilizaron en el gridsearch:

Modelo	Descripción de Parámetros
Bagging de estimador base árboles de decisión	**Número de estimadores:** valores utilizados *10 ,20, 50 y 100.* **Bootstrap:** valores utilizados *True y False.* **Bootstrap features:** valores utilizados *True y False.*
Máquina de Vectores de Soporte	**Penalización:** valores utilizados *0.25, 0.5 y 1.* **Kernel:** valores utilizados *linear, poly, rbf , sigmoid.* **Número iteraciones:** valor utilizado *10000.*
Red Neuronal Convolucional (CNN)	**Número de Neuronas CNN:** valores utilizados *32 y 64.* **Número de Neuronas Densas:** valores utilizados fueron *256 y 512.* **Batch size:** valores utilizados *32 y 64.*

Cuadro 7. Parámetros para optimizar, experimento 1.

Una vez realizada la optimización de parámetros, se obtuvieron los mejores resultados del indicador F1 de los modelos, que se observan en el Cuadro *8* y Cuadro *9*:

Requisito/Modelo	Bagging	CNN	Promedio
COMPATIBILIDAD	16%	4%	10%
DESEMPEÑO	19%	10%	14%
FIABILIDAD	14%	4%	9%
FUNCIONAL	79%	75%	77%
OTROS	0%	0%	0%
SEGURIDAD	14%	4%	9%
USABILIDAD	19%	10%	15%
Promedio	**23%**	**15%**	**19%**

Cuadro 8. Mejores resultados optimización hiperparámetros, experimento 1, conjunto de datos original.

Requisito / Modelo	Bagging	CNN	SVM	Promedio
COMPATIBILIDAD	94%	66%	64%	75%
DESEMPEÑO	94%	72%	58%	75%
FIABILIDAD	95%	73%	63%	77%
FUNCIONAL	77%	37%	26%	47%
OTROS	97%	78%	65%	80%
SEGURIDAD	94%	64%	51%	69%
USABILIDAD	93%	68%	63%	75%
Promedio	**92%**	**65%**	**56%**	**71%**

Cuadro 9. Mejores resultados optimización hiperparámetros, experimento 1, conjunto de datos aumentado.

En el Cuadro *10* y Cuadro *11* se muestran los parámetros con los que se obtuvieron los mejores resultados, por tanto, con los que se realizó el entrenamiento y evaluación final de los modelos:

Modelo	Parámetros Seleccionados
Bagging	**Número de estimadores:** 100 **Bootstrap:** Sin reemplazo. **Bootstrap features** Con reemplazo.
Red Neuronal Convolucional (CNN)	**Número de Neuronas CNN:** 32. **Número de Neuronas Densas:** 256. **Batch Size:** 64.

Cuadro 10. Hiperparámetros seleccionados, experimento 1, conjunto de datos original.

Modelo	Parámetros Seleccionados
Bagging	**Número de estimadores:** 100 **Bootstrap:** Sin reemplazo. **Bootstrap features** Con reemplazo.
Red Neuronal Convolucional (CNN)	**Número de Neuronas CNN:** 64. **Número de Neuronas Densas:** 256. **Batch Size:** 64.
Máquina de Vectores de Soporte	**Penalización:** 1 **Kernel:** Función de base radial. **Número de iteraciones:** 10000.

Cuadro 11. Hiperparámetros seleccionados, experimento 1, conjunto de datos aumentado.

6.6.3. Entrenamiento y evaluación final de desempeño

En el Cuadro *12* y la Figura *15* se muestran los resultados de los modelos optimizados y entrenados con el conjunto de datos original. En comparación con los modelos por defecto entrenados con el conjunto de datos original, el desempeño tuvo una pequeña mejora únicamente sobre la clase funcional, lo que confirma aún más el problema de tener un conjunto de datos desbalanceado. Para efectos de comparación en las secciones posteriores, se utilizarán los resultados obtenidos por el modelo Bagging.

Requisito / Modelo	Bagging	CNN	Promedio
COMPATIBILIDAD	13%	0%	7%
DESEMPEÑO	18%	3%	10%
FIABILIDAD	13%	1%	7%
FUNCIONAL	79%	76%	78%
OTROS	0%	0%	0%
SEGURIDAD	14%	0%	7%
USABILIDAD	16%	5%	11%
Promedio	**22%**	**12%**	**17%**

Cuadro 12. Resultados Modelos Definitivos, experimento 1, conjunto de datos original.

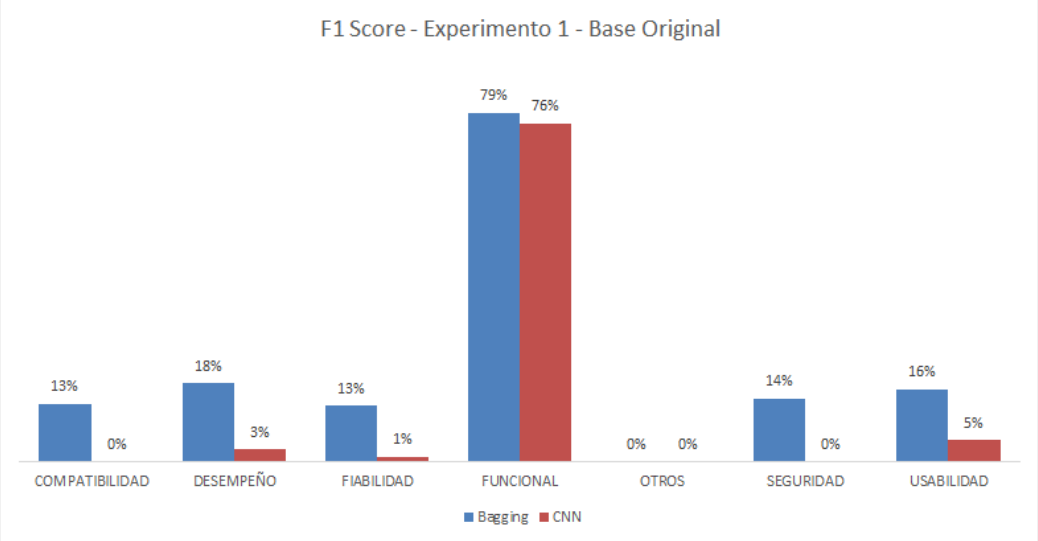

Figura 15. Desempeño Modelos Definitivos, experimento 1, conjunto de datos original.

No obstante, como se observa en el Cuadro *13* y la Figura *16* para los modelos entrenados con el conjunto de datos aumentado, el desempeño mejoró en más de un 20% para cada modelo debido a la selección de los hiperparámetros. De esta manera, se utilizará al modelo Bagging con un promedio de 92% para la comparación en las secciones posteriores.

Requisito / Modelo	Bagging	CNN	SVM	Promedio
COMPATIBILIDAD	94%	76%	64%	78%
DESEMPEÑO	94%	81%	58%	78%
FIABILIDAD	95%	79%	63%	79%
FUNCIONAL	77%	47%	26%	50%
OTROS	97%	85%	65%	82%
SEGURIDAD	94%	74%	51%	73%
USABILIDAD	93%	76%	63%	77%
Promedio	**92%**	**74%**	**56%**	**74%**

Cuadro 13. Resultados Modelos Definitivos, experimento 1, conjunto de datos aumentado.

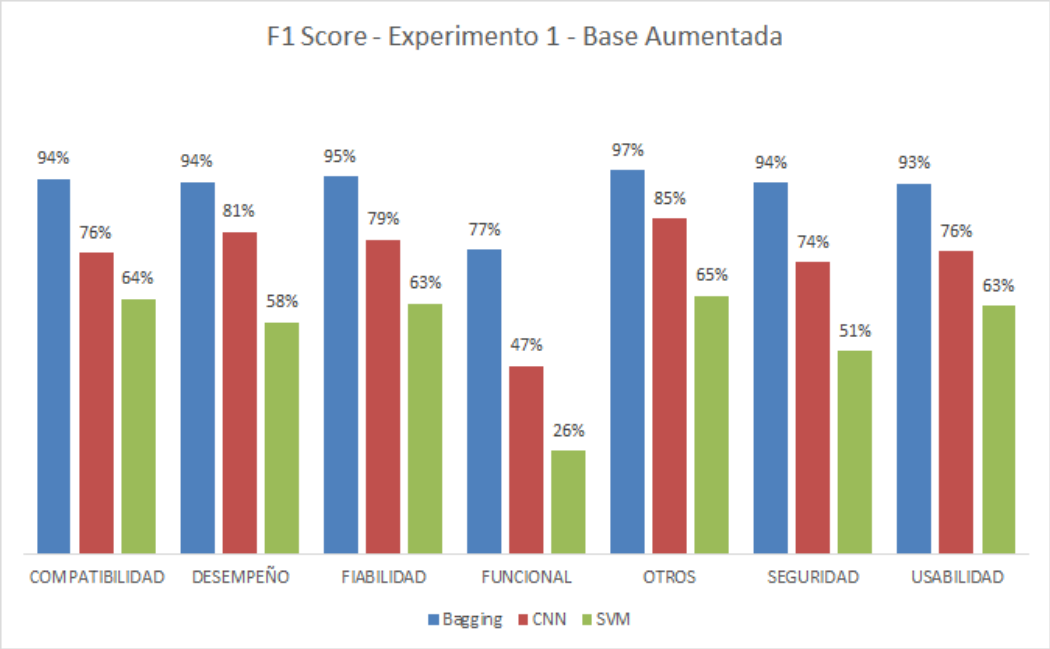

Figura 16. Desempeño Modelos Definitivos, experimento 1, conjunto de datos aumentado.

6.7. EXPERIMENTO 2 (CLASIFICACIÓN BINARIA DE REQUISITOS EN FUNCIONALES O NO FUNCIONALES)

Para este experimento se utilizaron las mismas técnicas de preprocesamiento y los mismos modelos de aprendizaje automático expuestos en la Sección 6.6, teniendo en cuenta únicamente que el problema será de clasificación binaria y no multiclase.

6.7.1. Resultados Modelos por Defecto (parámetros predeterminados)

En el Cuadro *14* y la Figura *17* se muestran los resultados del experimento 2 con el conjunto de datos sin aumentar (original). Allí, se observa que el predominio de muestras de la clase funcional afecta el desempeño de los modelos, siendo el modelo bagging el de mejor desempeño con un promedio de 59%, sin embargo, no se sesgan tanto como en el experimento 1 debido a que todos los requisitos no funcionales se agrupan y así, los modelos son capaces de generalizar mejor.

Requisito / Modelo	Bagging	CNN	LSTM	LSTM + CNN	SVM	Transformer	Promedio
F	76%	69%	66%	32%	77%	71%	65%
NF	43%	39%	36%	44%	23%	37%	37%
Promedio	**59%**	**54%**	**51%**	**38%**	**50%**	**54%**	**51%**

Cuadro 14. Resultados Modelos Defecto, experimento 2, conjunto de datos original.

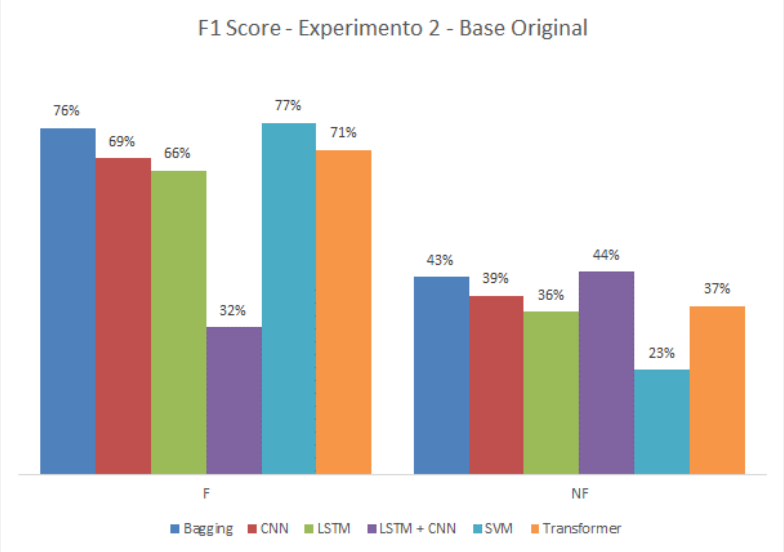

Figura 17. Desempeño Modelos Defecto, experimento 2, conjunto de datos original.

De esta manera, se seleccionaron los modelos bagging, CNN y transformer en el experimento con conjunto de datos original para el proceso de optimización de hiperparámetros.

Por otra parte, para el experimento con la información aumentada, se observó una disminución en el desempeño de ciertos modelos (como bagging o CNN) en la clase funcional (pasó de 76% a 69% en bagging y 69% a 63% en CNN), pero con una mejora en el desempeño hacia la clase no funcional (pasó de 43% a 67% en bagging y 39% a 55% en CNN, tal y como se muestra en el Cuadro *15* y la Figura *18*. Así, al igual que sucedió con el experimento 1, el aumento artificial de la información permite un mejor desempeño general de los modelos donde la clasificación de clase minoritaria se beneficia de este proceso.

Requisito / Modelo	Bagging	CNN	LSTM	LSTM + CNN	SVM	Transformer	Promedio
F	69%	63%	43%	59%	62%	54%	58%
NF	67%	55%	46%	44%	54%	55%	54%
Promedio	**68%**	**59%**	**45%**	**52%**	**58%**	**55%**	**56%**

Cuadro 15. Resultados Modelos Defecto, experimento 2, conjunto de datos aumentado.

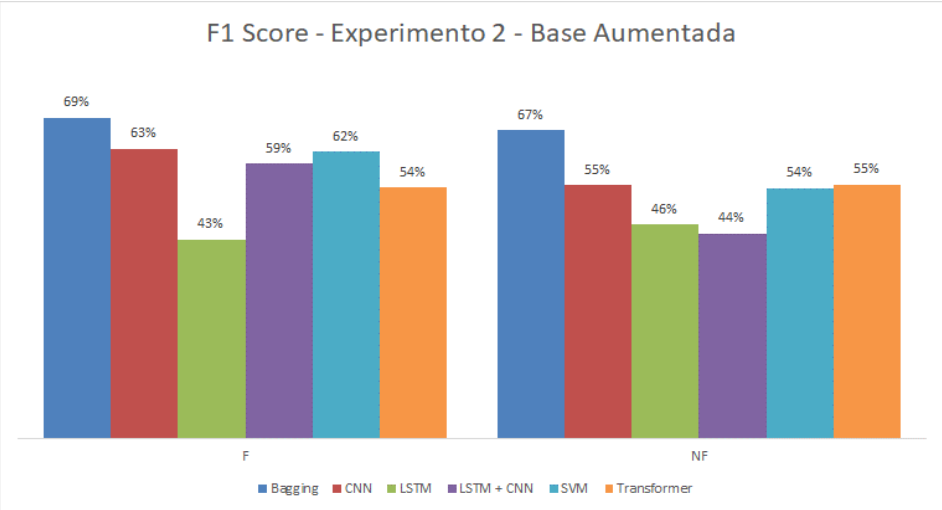

Figura 18. Desempeño Modelos Defecto, experimento 2, conjunto de datos aumentado.

De esta manera, se seleccionaron los modelos bagging, CNN y SVM en el experimento con conjunto de datos aumentado para el proceso de optimización de hiperparámetros.

6.7.2. Optimización Hiperparámetros

En el Cuadro *16* se muestran los parámetros que se optimizaron para cada modelo y el conjunto de valores o rangos para cada parámetro que se utilizaron en el gridsearch:

Modelo	Descripción de Parámetros
Bagging de estimador base árboles de decisión	**Número de estimadores:** valores utilizados *10, 20, 50 y 100.* **Bootstrap:** valores utilizados *True y False.* **Bootstrap features:** valores utilizados *True y False.*
Máquina de Vectores de Soporte	**Penalización:** valores utilizados *0.25, 0.5 y 1.* **Kernel** valores utilizados *linear, poly, rbf, sigmoid.* **Número iteraciones:** valor utilizado *10000.*
Red Neuronal Convolucional (CNN)	**Número de Neuronas CNN:** valores utilizados *32 y 64.* **Número de Neuronas Densas:** valores utilizados *256 y 512.* **Batch size:** valores utilizados *32 y 64.*
Transformer con Mecanismos de Atención	**Num_heads:** valores utilizados *4 y 8.* **Num_transformer_blocks:** valores utilizados *4 y 8.*

Cuadro 16. Parámetros para optimizar, experimento 2.

Una vez realizada la optimización de parámetros, se obtuvieron los mejores resultados del indicador F1 de los modelos, que se observan en el Cuadro *17* y Cuadro *18*:

Requisito / Modelo	Bagging	CNN	Transformer	Promedio
F	79%	73%	70%	74%
NF	48%	35%	39%	41%
Promedio	**64%**	**54%**	**55%**	**58%**

Cuadro 17. Mejores resultados optimización hiperparámetros, experimento 2, conjunto de datos original.

Requisito / Modelo	Bagging	CNN	SVM	Promedio
F	75%	62%	62%	66%
NF	75%	46%	54%	58%
Promedio	**75%**	**54%**	**58%**	**62%**

Cuadro 18. Mejores resultados optimización hiperparámetros, experimento 2, conjunto de datos aumentado.

En el Cuadro *19* y Cuadro *20* se muestran los parámetros con los que se obtuvieron los mejores resultados, por tanto, con los que se realizó el entrenamiento y evaluación final de los modelos:

Modelo	Parámetros Seleccionados
Bagging	**Número de estimadores:** 50 **Bootstrap:** Sin reemplazo. **Bootstrap features** Con reemplazo.
Red Neuronal Convolucional (CNN)	**Número de Neuronas CNN:** 32. **Número de Neuronas Densas:** 256. **Batch Size:** 64.
Transformer con mecanismos de atención	**Num_heads:** 8 **Nume_transformer_blocks:** 8

Cuadro 19. Hiperparámetros seleccionados, experimento 2, conjunto de datos original.

Modelo	Parámetros Seleccionados
Bagging	**Número de estimadores:** 100 **Bootstrap:** Sin reemplazo. **Bootstrap features** Con reemplazo.
Red Neuronal Convolucional (CNN)	**Número de Neuronas CNN:** 64. **Número de Neuronas Densas:** 512. **Batch Size:** 64.
Máquina de Vectores de Soporte	**Penalización:** 1 **Kernel:** Función de base radial. **Número de iteraciones:** 10000.

Cuadro 20. Hiperparámetros seleccionados, experimento 2,conjunto de datos aumentado.

6.7.3. Entrenamiento y evaluación final del desempeño

En el Cuadro *21* y la Figura 19 se presenta el desempeño de los modelos entrenados con los parámetros del cuadro 14 y el conjunto de datos original. Se observa que el desempeño mejoró en comparación con el entrenamiento de los modelos de los parámetros por defecto tanto para la clase predominante como para los requisitos no funcionales (en bagging pasó de 76% a 78% en la clase funcional y de 43% a 48% en la clase no funcional):

Requisito / Modelo	Bagging	CNN	Transformer	Promedio
F	78%	71%	76%	75%
NF	48%	39%	14%	34%
Promedio	**63%**	**55%**	**45%**	**54%**

Cuadro 21. Resultados Modelos Definitivos, experimento 2, conjunto de datos original.

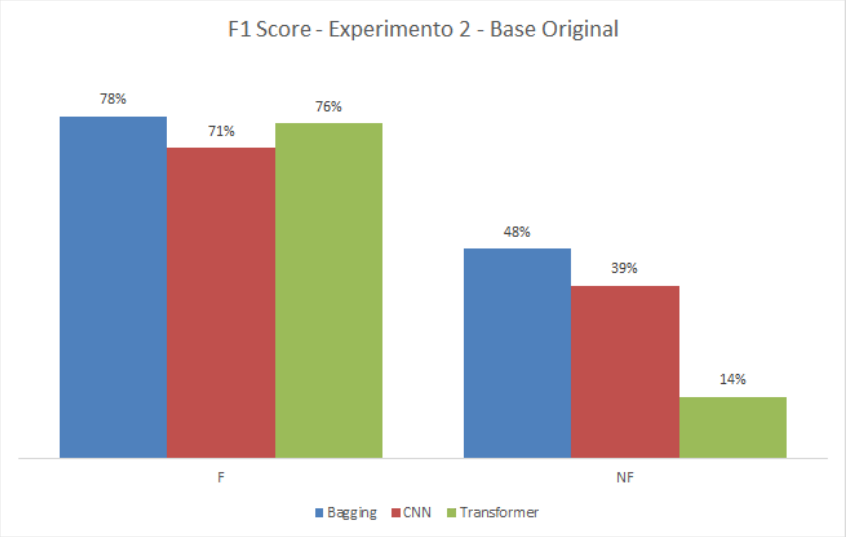

Figura 19. Desempeño Modelos Definitivos, experimento 2, conjunto de datos original.

Sin embargo, se observó más mejoría en el desempeño con el conjunto de datos aumentado, donde se logró un indicador F1 medio de aproximadamente el 60% para la clase no funcional, donde en el modelo bagging pasó de 69% a 75% en la clase funcional y 67% a 75% en la clase no funcional, tal y como se muestra en el Cuadro *22* y la Figura 20:

Requisito / Modelo	Bagging	CNN	SVM	Promedio
F	75%	64%	62%	67%
NF	75%	56%	54%	61%
Promedio	**75%**	**60%**	**58%**	**64%**

Cuadro 22. Resultados Modelos Definitivos, experimento 2, conjunto de datos aumentado.

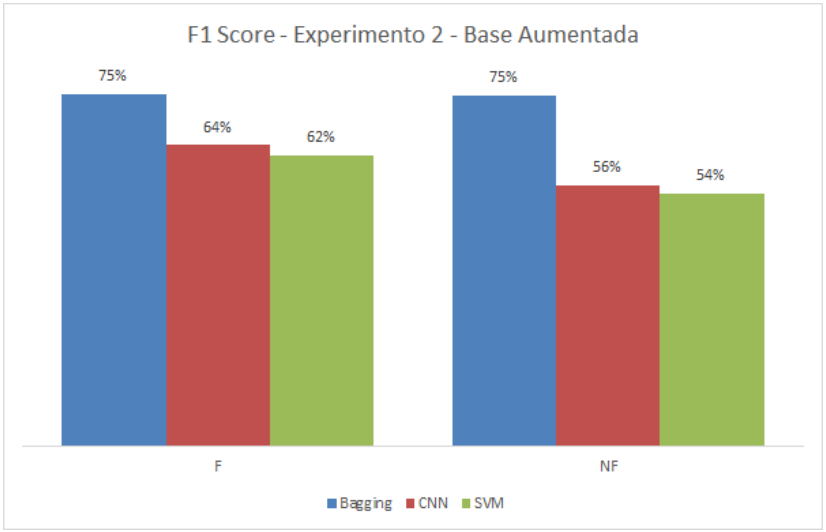

Figura 20. Desempeño Modelos Definitivos, experimento 2, conjunto de datos aumentado.

Para ambos casos del experimento con parámetros optimizados, ya sea con conjunto de datos original o aumentado, se observó que el modelo con mejor desempeño fue bagging.

6.8. EXPERIMENTO 3 (PROBLEMA MULTICLASE ÚNICAMENTE PARA LAS 6 CATEGORÍAS NO FUNCIONALES)

Para este experimento, se utilizaron las mismas técnicas de preprocesamiento y los mismos modelos de aprendizaje automático expuestos en la Sección 6.6 con la diferencia de que el conjunto de datos para el entrenamiento de los modelos no contenía la clase predominante (requisitos funcionales), por lo que los modelos se enfocaron en generalizar las categorías de los requisitos no funcionales.

6.8.1. Resultado Modelos por Defecto (parámetros predeterminados)

En el Cuadro *23* y la Figura *21* se muestran los resultados del experimento 3 con el conjunto de datos sin aumentar (original). Se observa que a pesar de que se ha eliminado la información de los requisitos funcionales, los modelos no logran generalizar los patrones para poder predecir cada categoría de los requisitos no funcionales. Lo anterior podría ser por dos motivos generales: primero, que no exista las suficientes muestras de cada tipo de requisito para que los modelos aprendan sin tener en el sobre ajuste (el cual se está mitigando con la validación cruzada); que el tiempo de entrenamiento de los modelos no sea lo suficientemente extenso para que los modelos aprendan de la información.

Requisito / Modelo	Bagging	CNN	LSTM	LSTM + CNN	SVM	Transformer	Promedio
COMPATIBILIDAD	35%	23%	14%	19%	28%	19%	23%
DESEMPEÑO	39%	22%	6%	15%	22%	20%	21%
FIABILIDAD	24%	13%	16%	11%	4%	12%	13%
OTROS	7%	8%	4%	3%	0%	8%	5%
SEGURIDAD	33%	19%	14%	12%	33%	22%	22%
USABILIDAD	34%	25%	15%	2%	34%	29%	23%
Promedio	**29%**	**18%**	**12%**	**10%**	**20%**	**18%**	**18%**

Cuadro 23. Resultados Modelos Defecto, experimento 3, conjunto de datos original.

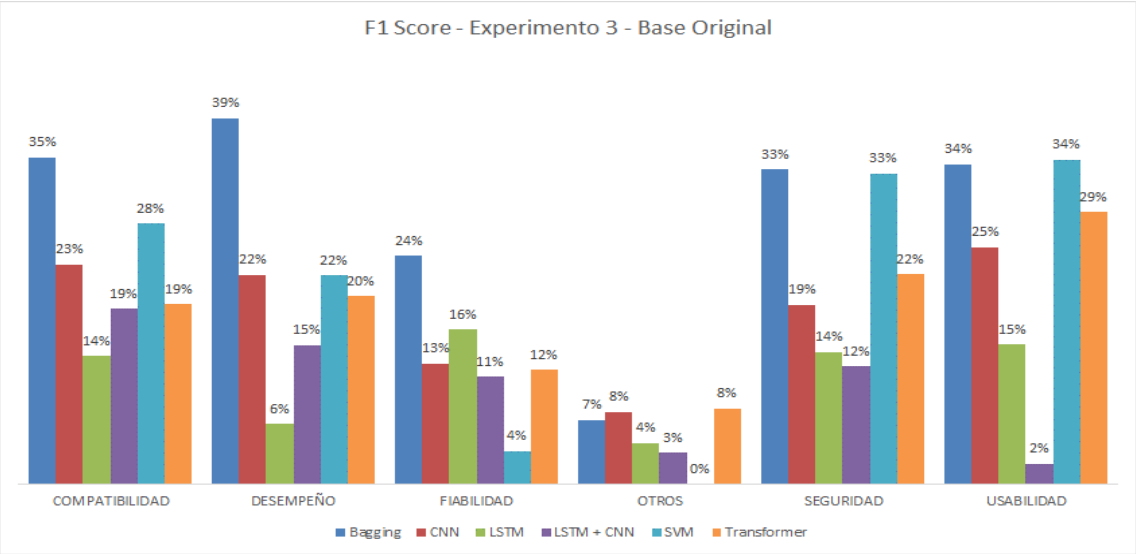

Figura 21. Desempeño Modelos Defecto, experimento 3, conjunto de datos original.

Ahora bien, el desempeño de los modelos con el conjunto de datos aumentado se puede observar en el Cuadro *24* y la Figura *22*, en general, el desempeño en cada requisito mejoró, lo que confirma que el desempeño de los modelos del punto anterior se afecta por la poca cantidad de muestras por tipo de requisito no funcional. Sin embargo, los resultados no son sobresalientes, pues en ningún modelo el desempeño supera el 50%, por tanto, se procedió con el siguiente paso de optimización de parámetros y se descartan ciertos modelos, sobre todo los basados en redes neuronales recurrentes.

Requisito / Modelo	Bagging	CNN	LSTM	LSTM + CNN	SVM	Transformer	Promedio
COMPATIBILIDAD	43%	33%	14%	13%	33%	21%	26%
DESEMPEÑO	44%	44%	8%	17%	31%	23%	28%
FIABILIDAD	51%	35%	16%	11%	32%	18%	27%
OTROS	59%	51%	12%	14%	39%	23%	33%
SEGURIDAD	47%	29%	39%	15%	25%	21%	29%
USABILIDAD	36%	31%	11%	8%	36%	19%	24%
Promedio	**47%**	**37%**	**17%**	**13%**	**33%**	**21%**	**28%**

Cuadro 24. Resultados Modelos Defecto, experimento 3, conjunto de datos aumentado.

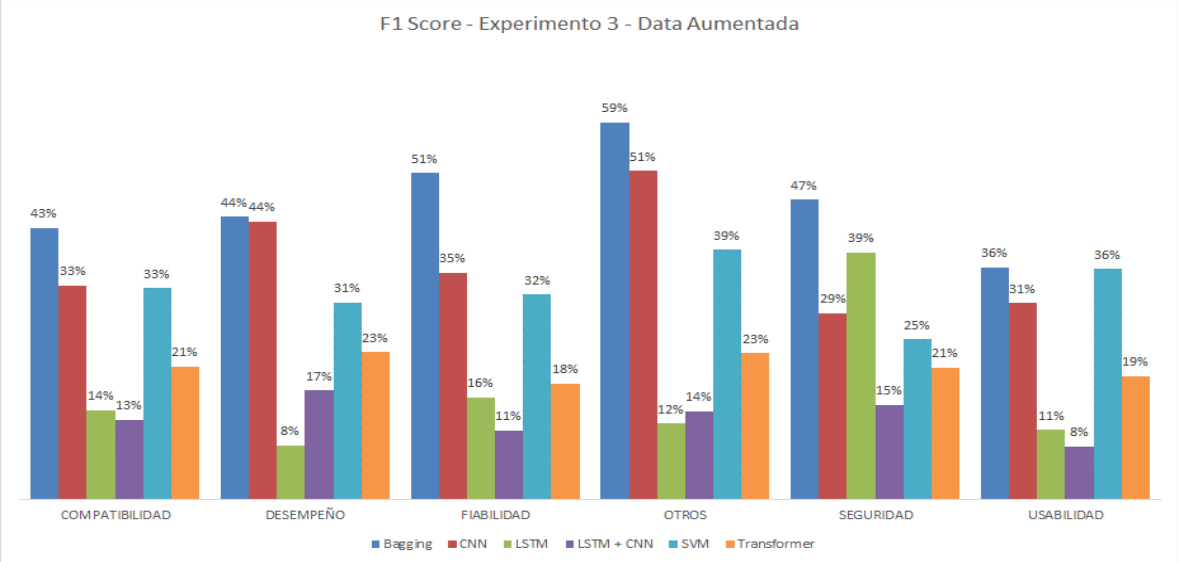

Figura 22. Desempeño Modelos Defecto, experimento 3, conjunto de datos aumentado.

De esta manera, para el proceso de optimización de hiperparámetros, se seleccionaron los modelos bagging, CNN y SVM que obtuvieron el mejor desempeño.

6.8.2. Optimización de Hiperparámetros

En el Cuadro *25* se muestran los parámetros que se optimizaron para cada modelo y el conjunto de valores o rangos para cada parámetro que se utilizaron en el gridsearch:

Modelo	Descripción de Parámetros
Bagging de estimador base árboles de decisión	**Número de estimadores:** valores utilizados *10, 20, 50 y 100 (para el conjunto datos original) y 500, 1000 , 2000 (para el conjunto datos aumentado).* **Bootstrap:** valores *True y False.* **Bootstrap features** valores utilizados *True y False.*
Máquina de Vectores de Soporte	**Penalización:** valores utilizados *0.25, 0.5 y 1.* **Kernel:** valores utilizados *linear, poly, rbf, sigmoid.* **Número iteraciones:** valor utilizado *10000.*
Red Neuronal Convolucional (CNN)	**Número de Neuronas CNN:** valores utilizados *32 y 64 (para el conjunto de datos original) y 64 y 128 (para el conjunto de datos aumentado).* **Número de Neuronas Densas:** valores utilizados *256 y 512 (para el conjunto de datos original) y 512 y 1024 (para el conjunto de datos aumentado).* **Batch size:** valores utilizados f*32 y 64 (para el conjunto de datos original) y 16 y 32 (para el conjunto de datos aumentado).*

Cuadro 25. Parámetros para optimizar, experimento 3.

Una vez realizada la optimización de parámetros, se obtuvieron los mejores resultados del indicador F1 de los modelos, que se observan en el Cuadro *26* y Cuadro *27*:

Requisito / Modelo	Bagging	CNN	SVM	Promedio
COMPATIBILIDAD	39%	23%	31%	31%
DESEMPEÑO	42%	21%	22%	28%
FIABILIDAD	31%	20%	5%	19%
OTROS	5%	8%	0%	4%
SEGURIDAD	45%	26%	33%	35%
USABILIDAD	47%	29%	33%	36%
Promedio	**35%**	**21%**	**21%**	**26%**

Cuadro 26. Mejores resultados optimización hiperparámetros, experimento 3, conjunto de datos original.

Requisito / Modelo	Bagging	CNN	SVM	Promedio
COMPATIBILIDAD	55%	28%	28%	37%
DESEMPEÑO	61%	33%	41%	45%
FIABILIDAD	61%	36%	30%	42%
OTROS	73%	49%	40%	54%
SEGURIDAD	55%	26%	32%	38%
USABILIDAD	44%	19%	29%	31%
Promedio	**58%**	**32%**	**33%**	**41%**

Cuadro 27. Mejores resultados optimización hiperparámetros, experimento 3, conjunto de datos aumentado.

En el Cuadro *28* y el Cuadro *29* se muestran los parámetros con los que se obtuvieron los mejores resultados, por tanto, con los que se realizó el entrenamiento y evaluación final de los modelos:

Modelo	Parámetros Seleccionados
Bagging	**Número de estimadores:** 100 **Bootstrap:** Con reemplazo. **Bootstrap features** Con reemplazo.
Red Neuronal Convolucional (CNN)	**Número de Neuronas CNN:** 64. **Número de Neuronas Densas:** 256. **Batch Size:** 64.
Máquina de Vectores de Soporte	**Penalización:** 1 **Kernel:** Función de base radial. **Número de iteraciones:** 10000.

Cuadro 28. Hiperparámetros seleccionados, experimento 3, conjunto de datos original.

Modelo	Parámetros Seleccionados
Bagging	**Número de estimadores:** 500 **Bootstrap:** Con reemplazo. **Bootstrap features** Con reemplazo.
Red Neuronal Convolucional (CNN)	**Número de Neuronas CNN:** 64. **Número de Neuronas Densas:** 512. **Batch Size:** 32.
Máquina de Vectores de Soporte	**Penalización:** 1 **Kernel:** Función de base radial. **Número de iteraciones:** 10000.

Cuadro 29. Hiperparámetros seleccionados, experimento 3,conjunto de datos aumentado.

6.8.3. Entrenamiento y evaluación final de desempeño

En el Cuadro *30* y la Figura *23* se presentan los resultados definitivos con el conjunto de datos original. Si bien, existió una mejoría con respecto a los modelos ejecutados con los parámetros por defecto, la insuficiencia de ejemplos afectó el desempeño general de los modelos, sobre todo en requisitos con muy pocos registros como fiabilidad u otros:

Requisito / Modelo	Bagging	CNN	SVM	Promedio
COMPATIBILIDAD	39%	22%	28%	30%
DESEMPEÑO	47%	22%	23%	31%
FIABILIDAD	27%	18%	3%	16%
OTROS	4%	9%	0%	4%
SEGURIDAD	43%	27%	33%	34%
USABILIDAD	43%	32%	34%	36%
Promedio	**34%**	**22%**	**20%**	**25%**

Cuadro 30. Resultados Modelos Definitivos, experimento 3, conjunto de datos original.

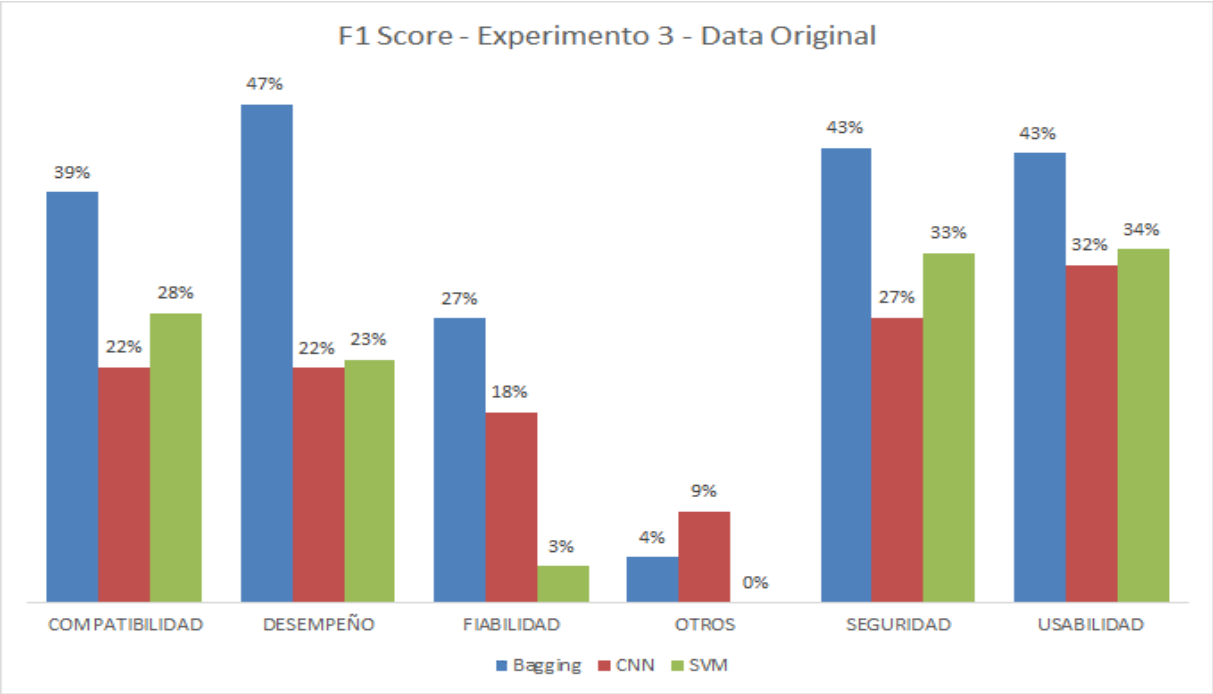

Figura 23. Desempeño Modelos Definitivos, experimento 3, conjunto de datos original.

El aumento artificial del conjunto de datos permitió mejorar el desempeño en los modelos definitivos que, en conjunto con la optimización de parámetros, incrementó el desempeño de los modelos con respecto a sus homólogos sin optimizar, por ejemplo, el modelo bagging pasó de un promedio de 47% a 56% y CNN pasó de 37% a 40%, tal como se muestra en el Cuadro *31* y la Figura *24*, sin embargo, estos resultados no fueron sobresalientes:

Requisito / Modelo	Bagging	CNN	SVM	Promedio
COMPATIBILIDAD	55%	37%	28%	40%
DESEMPEÑO	57%	43%	38%	46%
FIABILIDAD	59%	46%	22%	42%
OTROS	75%	62%	43%	60%
SEGURIDAD	50%	34%	29%	38%
USABILIDAD	43%	22%	32%	32%
Promedio	**56%**	**40%**	**32%**	**43%**

Cuadro 31. Resultados Modelos Definitivos, experimento 3, conjunto de datos aumentado

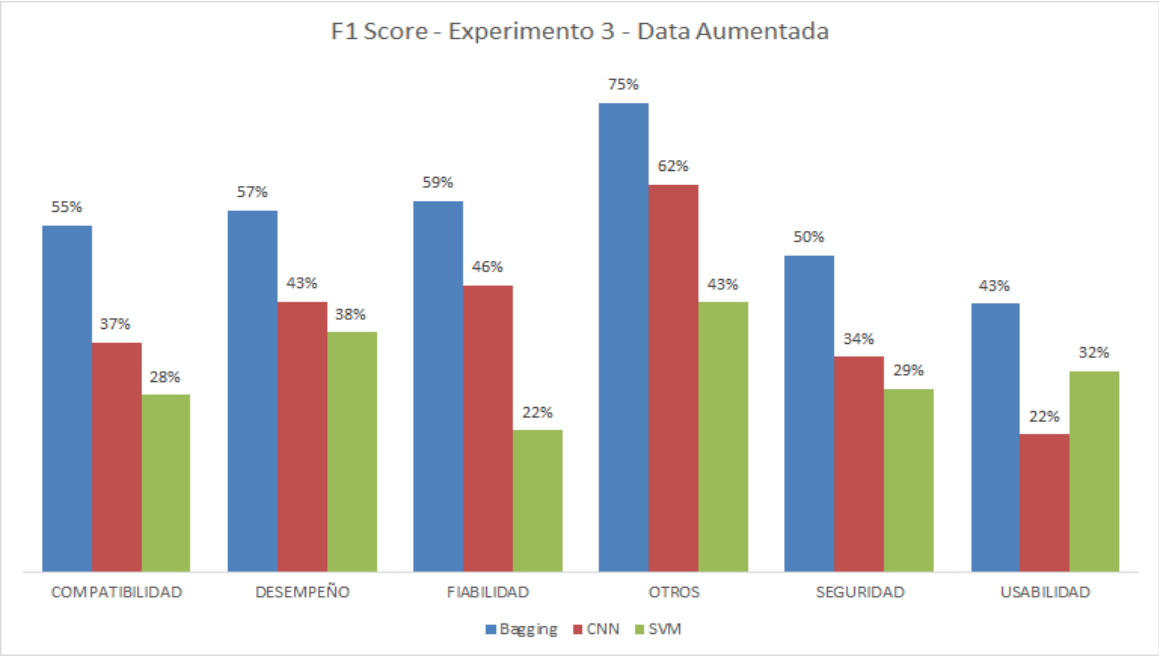

Figura 24. Desempeño Modelos Definitivos, experimento 3, conjunto de datos aumentado.

6.9. EXPERIMENTO 4 (MODELOS COMBINADOS PARA CLASIFICACIÓN BINARIA Y LUEGO MULTICLASE DE 6 CATEGORÍAS NO FUNCIONALES)

Teniendo en cuenta los resultados obtenidos en el experimento 1 en la categoría de requisito funcional, se propone realizar un experimento adicional de combinación de los modelos finales del experimento 2 (funcionales y no funcionales) y del experimento 3 (6 categorías de no funcionales).

Para este experimento se realizó el entrenamiento individual de los modelos binarios y multiclase y, una vez entrenados, se realizó la prueba de desempeño. Este proceso se realizó con validación cruzada con la finalidad de evitar el sobre ajuste de los modelos.

Así, se estableció un flujo donde los tres mejores modelos finales del experimento 2 clasifiquen los requisitos en funcionales y no funcionales, y en caso de que el requisito sea no funcional, pasa a los tres mejores modelos finales del experimento 3 para que se clasifique la categoría de requisito no funcional.

La evaluación del desempeño se calculó manualmente y se determinó como predicción correcta si los modelos aciertan en los dos casos, es decir, para identificar un verdadero positivo se tuvo en cuenta lo siguiente:

- Cuando en la predicción del experimento 2, los modelos aciertan con la clasificación de funcional.
- Cuando en la predicción del problema tipo 2, los modelos aciertan con la clasificación de no funcional y en la predicción del problema tipo 3, los modelos aciertan en la clasificación de la categoría de no funcional.

Para el desarrollo del flujo se iteró entre todas las parejas de los tres mejores modelos finales que del experimento 2 y el experimento 3, tanto para el conjunto de datos original y

el conjunto de datos aumentado, de esta forma, se puede observar el par de modelos que genera mejores resultados. Los modelos que se iteraron son:

- Bagging entrenado para el experimento 2 combinado con modelos entrenados para el experimento 3 Bagging, CNN, SVM.
- CNN entrenado para el experimento 2 combinado con modelos entrenados para el experimento 3 Bagging, CNN, SVM.
- Transformer entrenado para el experimento 2 combinado con modelos entrenados para el experimento 3 Bagging, CNN, transformer para el conjunto de datos original y SVM para el conjunto de datos aumentado.

En el Cuadro *32* se muestran los resultados para los modelos entrenados con el conjunto de datos original. Se observa el fenómeno que sucedió en los experimentos 1 y 2, donde los modelos tienden a ajustarse únicamente a los requisitos funcionales producto de la falta de ejemplos para los requisitos no funcionales.

Requisito / Modelo	Binario - Bagging			Binario - CNN			Binario - Transformer			Promedio
	Bagging	CNN	SVM	Bagging	CNN	SVM	Bagging	CNN	Transformer	
FUNCIONAL	78%	77%	78%	70%	70%	71%	76%	75%	76%	75%
COMPATIBILIDAD	35%	17%	18%	18%	10%	11%	6%	6%	5%	14%
DESEMPEÑO	31%	17%	15%	17%	15%	9%	6%	8%	3%	13%
FIABILIDAD	17%	8%	0%	11%	12%	0%	2%	2%	1%	6%
FUNCIONAL	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
OTROS	3%	5%	0%	2%	5%	0%	0%	6%	2%	2%
SEGURIDAD	22%	12%	18%	20%	6%	15%	4%	3%	2%	11%
USABILIDAD	36%	27%	25%	26%	15%	24%	15%	8%	7%	20%
Promedio	**28%**	**21%**	**19%**	**20%**	**17%**	**16%**	**14%**	**14%**	**12%**	**18%**

Cuadro 32. Resultados Modelos Combinados, experimento 4, conjunto de datos original.

Por otra parte, para los modelos con información aumentada, se observó una mejoría en el desempeño para las clases no funcionales a pesar de una disminución para la clase funcional (paso de un promedio de 75% a 66%), como se puede observar en el Cuadro *33*, la pareja de modelos finales que obtuvo el mejor promedio fue Bagging del experimento 2 y Bagging del experimento 3 con un promedio en el indicador F1 de 54%:

Requisito / Modelo	Binario - Bagging			Binario - CNN			Binario - Transformer			Promedio
	Bagging	CNN	SVM	Bagging	CNN	SVM	Bagging	CNN	SVM	
FUNCIONAL	77%	76%	77%	53%	66%	50%	64%	64%	64%	66%
COMPATIBILIDAD	58%	49%	29%	44%	39%	27%	45%	41%	28%	40%
DESEMPEÑO	60%	50%	31%	45%	40%	22%	44%	33%	23%	39%
FIABILIDAD	61%	51%	31%	44%	36%	28%	46%	45%	27%	41%
FUNCIONAL	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
OTROS	57%	48%	23%	37%	38%	20%	44%	36%	21%	36%
SEGURIDAD	59%	50%	29%	41%	43%	18%	46%	37%	21%	38%
USABILIDAD	63%	49%	33%	48%	41%	27%	50%	41%	31%	42%
Promedio	**54%**	**47%**	**32%**	**39%**	**38%**	**24%**	**42%**	**37%**	**27%**	**38%**

Cuadro 33. Resultados Modelos Combinados, experimento 4, conjunto de datos aumentado.

En la Figura *25* se observa el desempeño de los modelos combinados con mejores resultados y la mejoría en el desempeño en cada clase al aumentar los datos:

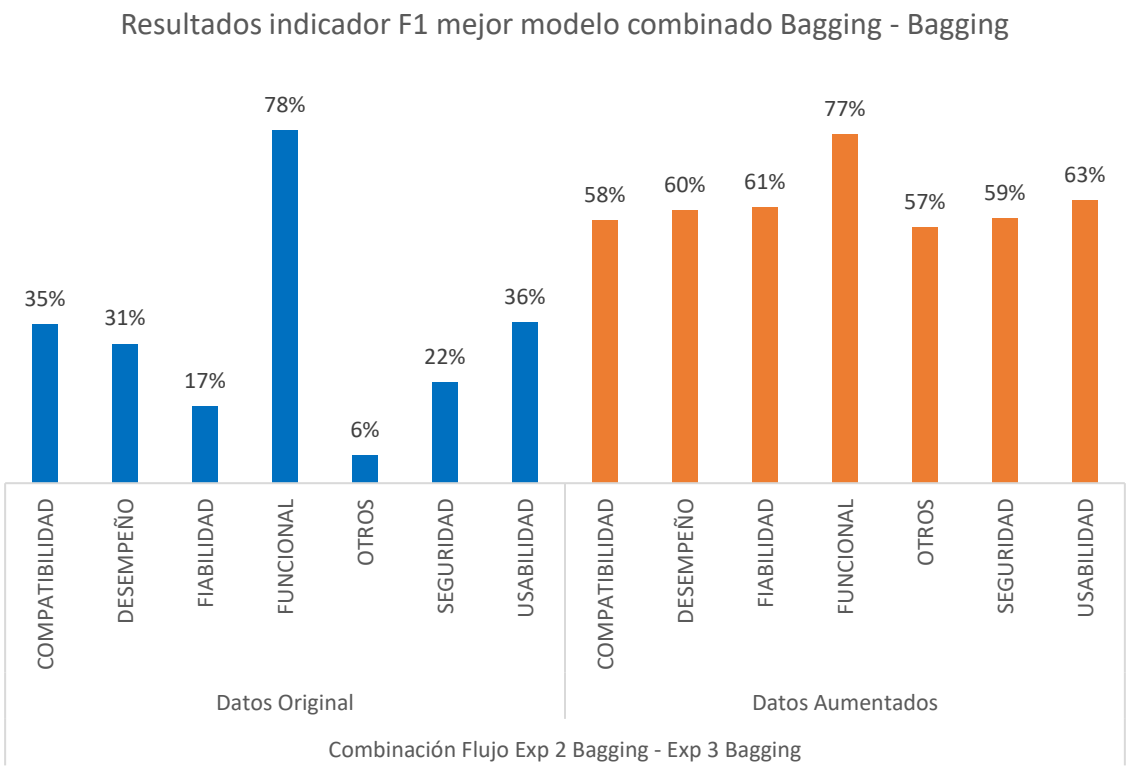

Figura 25 Resultados indicador F1 mejor modelo combinado Bagging - Bagging

Al observar los resultados de este experimento, se puede mencionar que debido a que el modelo binario y el modelo multiclase no presentan un desempeño sobresaliente para la clasificación de los requisitos (indicador F1 por debajo del 80% y del 60% respectivamente), sumado a que el modelo multiclase intenta predecir la clase del requisito a pesar de que debió detectarse como funcional, se afecta el indicador general de desempeño del experimento.

6.10. EXPERIMENTO 5 - VARIANTE DE GPT-3

Durante el transcurso del proyecto, se tuvo acceso a la versión beta cerrada del *Transformer Generativo Pre Entrenado (GPT-3 por sus siglas en inglés)*, que es un modelo de lenguaje natural para realizar diferentes tareas asociadas a la comunicación humana con mejor nivel de precisión respecto a anteriores modelos de menor capacidad. Entre ellas, la traducción del lenguaje, generación de conversaciones, clasificación de ejemplos, e incluso traducción de código de un lenguaje de programación a otro (Brown et al. 2020). La versión completa de GPT-3, lanzada en mayo de 2020, tiene una capacidad de 175.000 millones de parámetros de aprendizaje automatizado y es parte de una tendencia en sistemas de procesamiento de lenguaje natural (NLP) basados en representaciones de lenguaje pre-entrenadas.

Con el ánimo de comparar resultados en el experimento 1, se realizó *fine tuning* al modelo con una base de 200 requisitos de software (ya que la API es limitada en el número de usos). Los resultados obtenidos se observan en la Figura *26*.

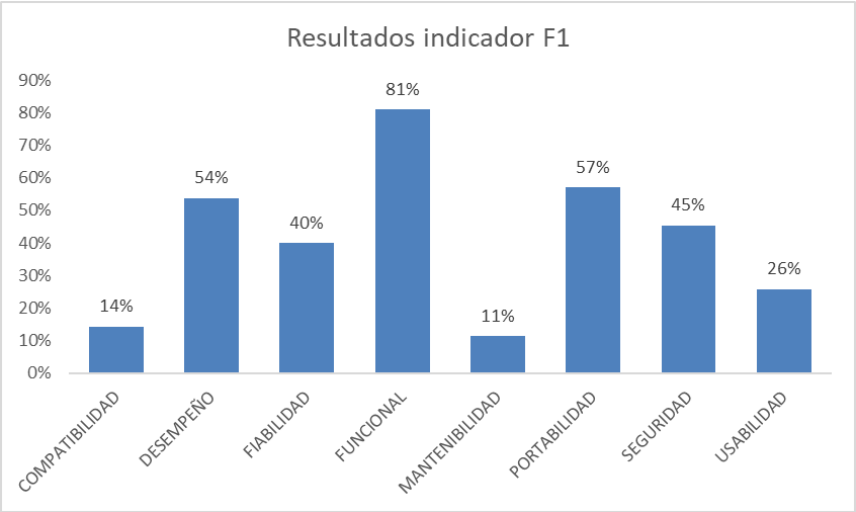

Figura 26. Resultados Indicador F1 GPT-3.

Como se puede observar, el modelo generó resultados similares a los obtenidos con los modelos entrenados con los parámetros por defecto y sin aumento de datos de la Sección 6.6. Por tanto, es una opción prometedora en el caso de acceder a la versión completa, pues al aumentar los datos, se esperaría un mejor desempeño.

6.11. DISCUSIÓN DE RESULTADOS

De acuerdo con los resultados obtenidos en cada uno de los experimentos evaluados en las secciones anteriores, se encontró que los modelos no basados en aprendizaje profundo (Bagging, Máquina de Vectores de Soporte) estuvieron siempre entre los modelos con mejor desempeño para los experimentos tanto con conjunto de datos original, como con conjuntos de datos aumentado.

Por otra parte, para los modelos basados en aprendizaje profundo, se encontró que la red neuronal convolucional (CNN) fue el tipo de modelo que mejor desempeño obtuvo, y que los modelos basados en redes neuronales recurrentes (LSTM / CNN + LSTM) no tuvieron buen desempeño para los experimentos realizados. Lo anterior, podría ser por la complejidad de encontrar una arquitectura adecuada para cada tipo de modelo debido a la cantidad considerable de hiperparámetros que tienen (número y tipo de capas, funciones de activación, número de neuronas por cada tipo de capa, entre otros), también por el número máximo de iteraciones aplicadas en el proceso de entrenamiento, el cual fue de 100, ya que este tipo de modelos requiere de miles de iteraciones, lo que implica un esfuerzo computacional relevante (Vaswani et al. 2017), por ello, modelos tan complejos como la arquitectura del *Transformer* que a nivel teórico debería obtener mayor desempeño, no se encontró entre los mejores modelos de los experimentos realizados.

El desempeño promedio de los modelos con mejores resultados en las dos categorías (no aprendizaje profundo y aprendizaje profundo), en cada experimento con sus respectivos escenarios (modelos por defecto, modelos optimizados, modelos definitivos) se observa en la Figura *27*, Figura *28* y Figura *29*. Para el experimento 1, se evidencia un promedio en los modelos definitivos con datos aumentados en el desempeño de Bagging con 92%, superando a la red neuronal convolucional (CNN) de 74%; para el experimento 2, se evidencia un promedio en los modelos definitivos con datos aumentados en el desempeño de Bagging con 75%, superando a la red neuronal convolucional (CNN) de 60%; y para el experimento 3 se evidencia un promedio en los modelos definitivos con datos aumentados en el desempeño de Bagging con 56%, superando a la red neuronal convolucional (CNN) de 40%:

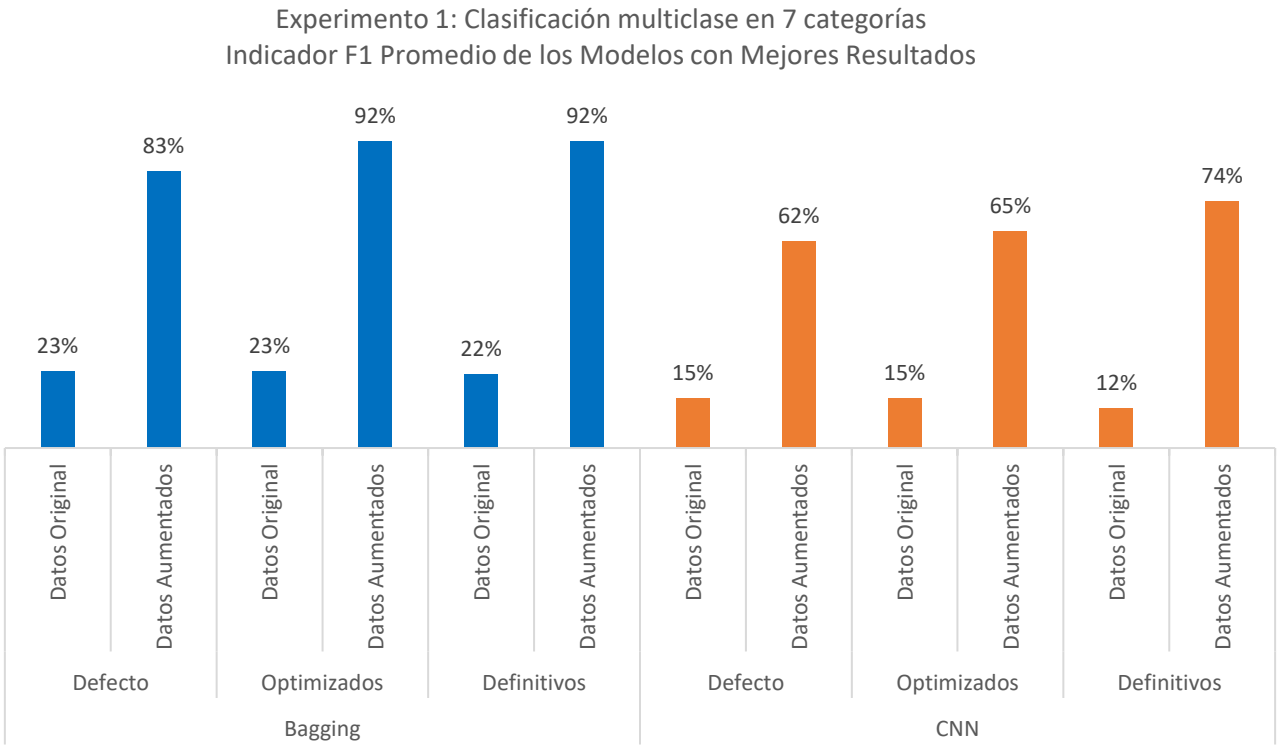

Figura 27. Resultados Indicador F1 Promedio de los Modelos con Mejores Resultados en los Tres Escenarios: Modelos por Defecto - Optimizados – Definitivos, Experimento 1

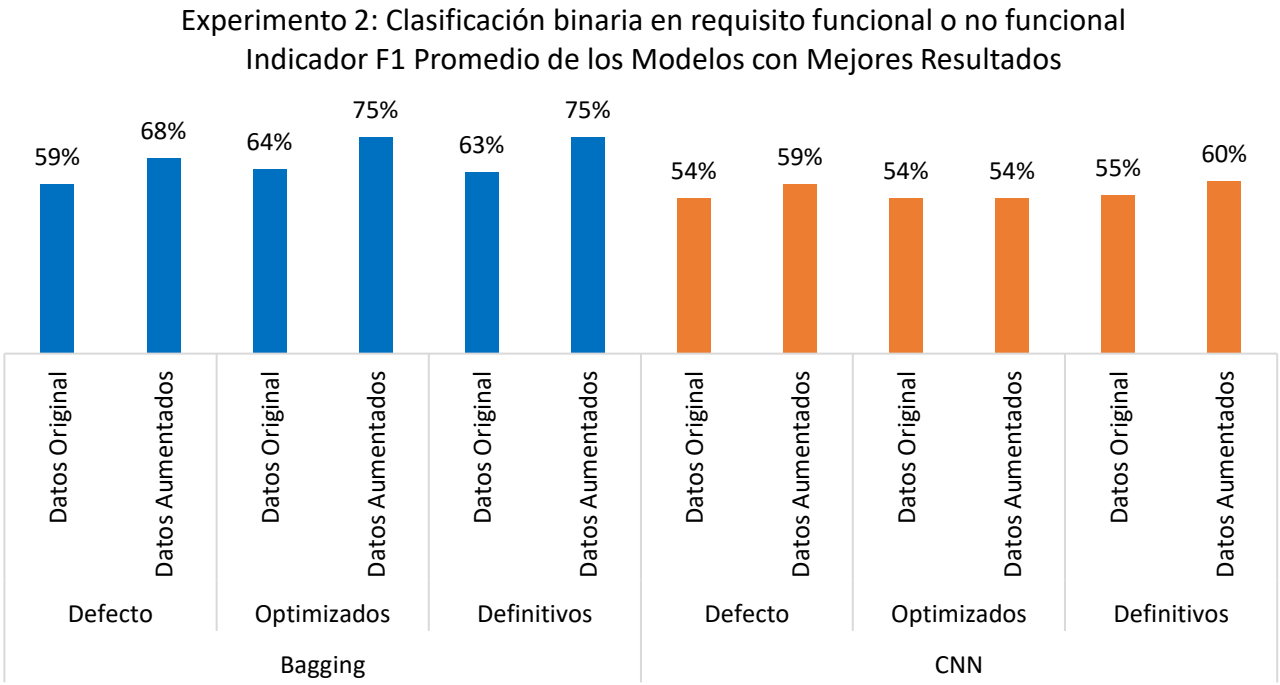

Figura 28. Resultados Indicador F1 Promedio de los Modelos con Mejores Resultados en los Tres Escenarios: Modelos por Defecto - Optimizados – Definitivos, Experimento 2

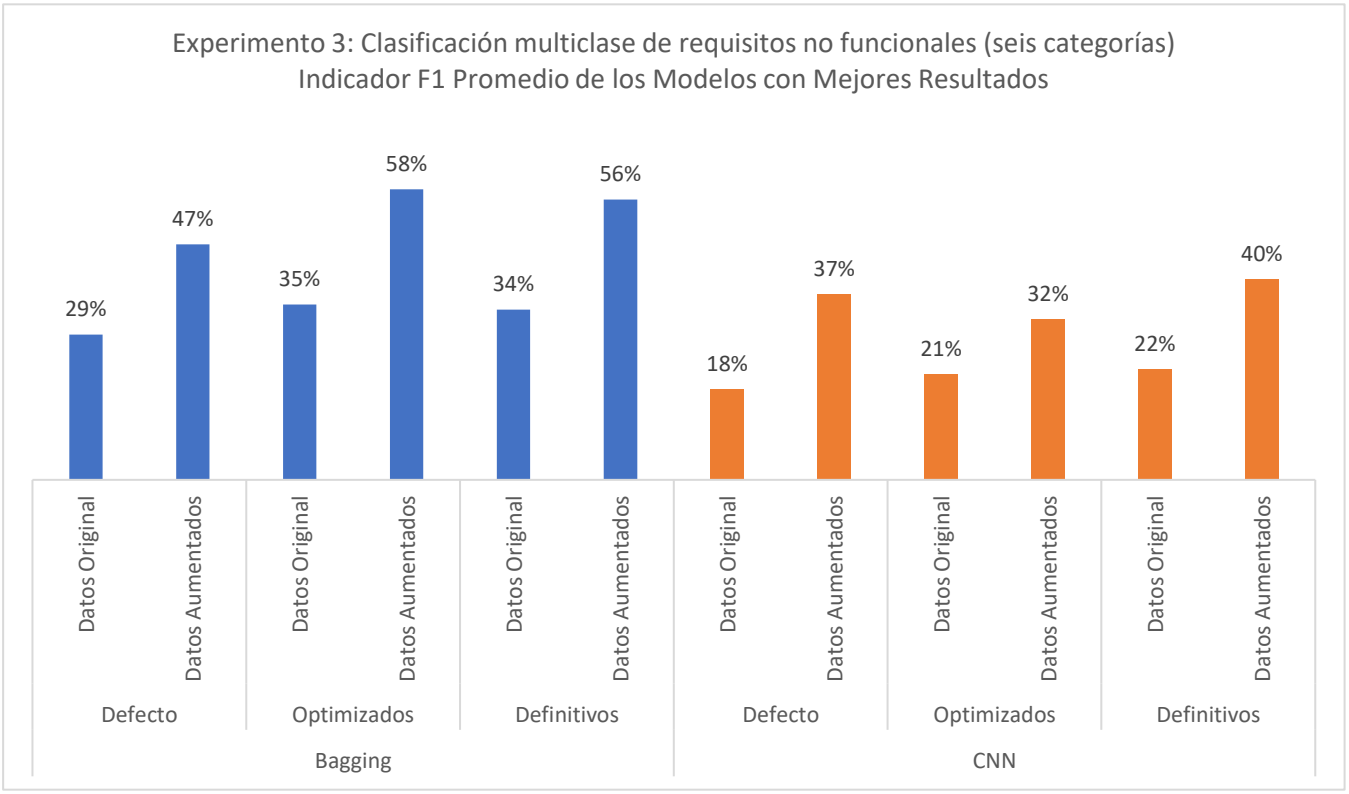

Figura 29. Resultados Indicador F1 Promedio de los Modelos con Mejores Resultados en los Tres Escenarios: Modelos por Defecto - Optimizados – Definitivos, Experimento 3

La incidencia relevante del aumento de datos en el desempeño de los modelos al clasificar cada clase frente a las muestras originales del conjunto de datos, se observa en la Figura *30* y Figura *31*, las cuales presentan el comportamiento en el escenario general de los tres experimentos y en particular para la técnica de Bagging:

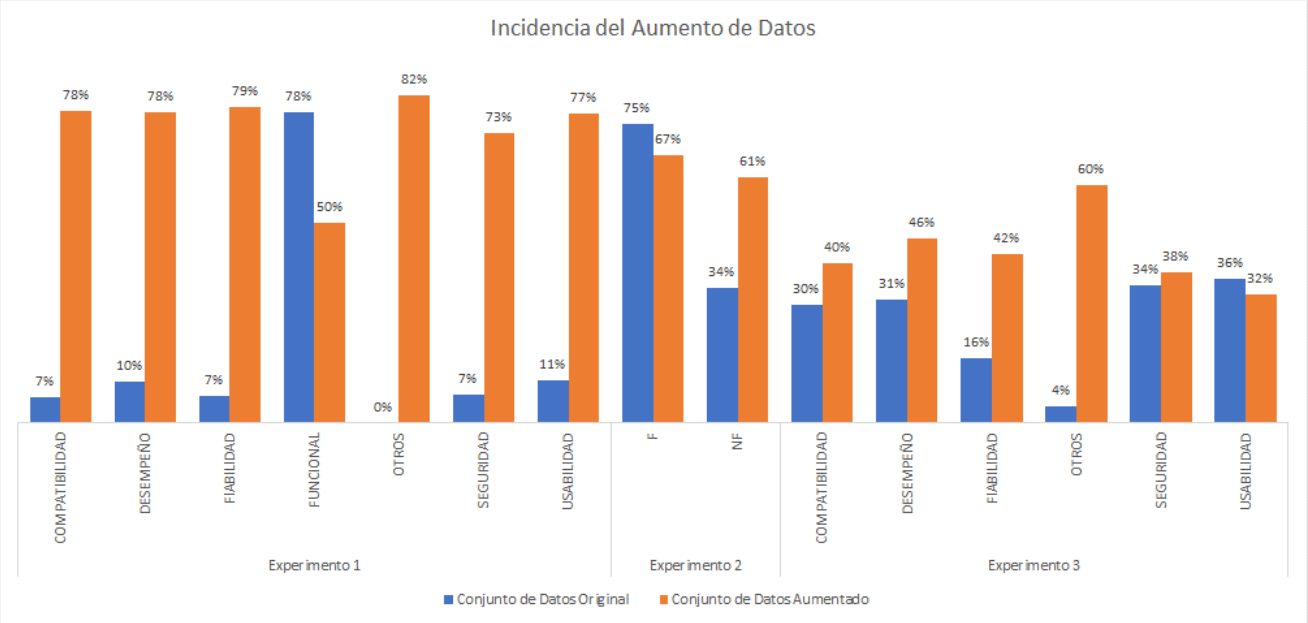

Figura 30. Comparación de la incidencia del aumento de datos en el indicador f1 general de los experimentos 1, 2 y 3.

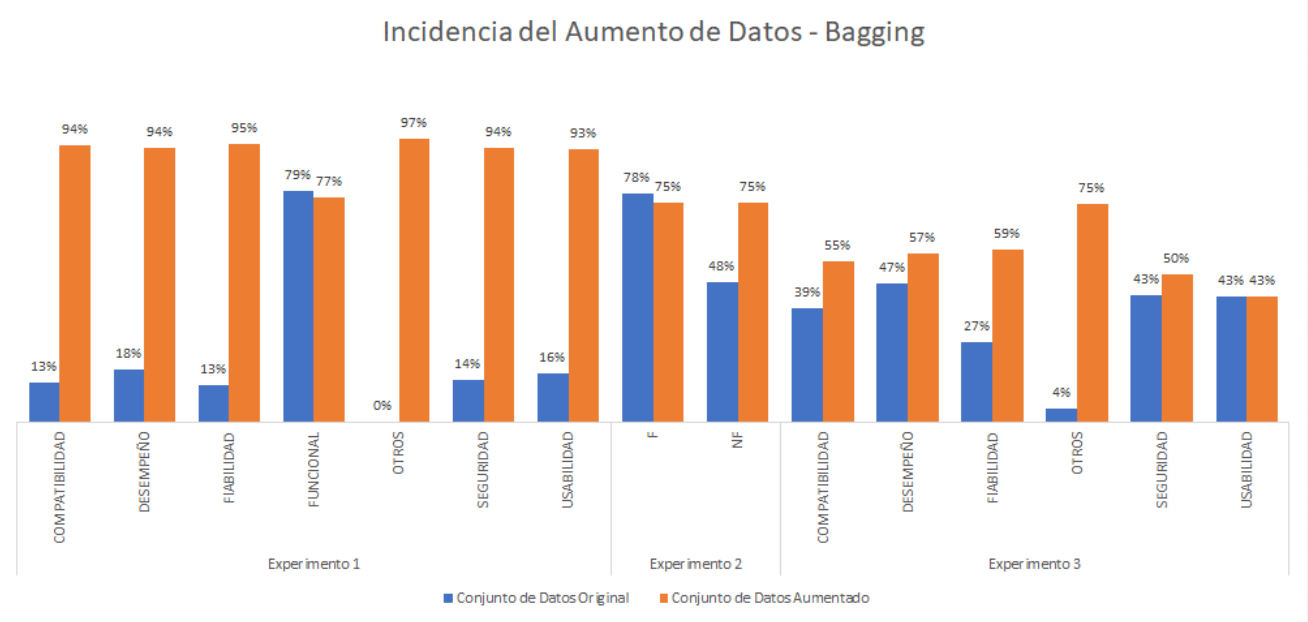

Figura 31. Comparación de la incidencia del aumento de datos en el indicador f1 general del modelo con mejores resultados Bagging.

Al realizar una comparación de los resultados obtenidos en los tres primeros experimentos, se observó que el modelo *bagging* presenta un desempeño superior que el promedio general de todos los modelos, como se muestra en la Figura *32*. Además que los mejores resultados del modelo bagging se obtuvieron con el conjunto de datos aumentado, por tanto, se recomienda el uso del modelo bagging para la clasificación en 7 categorías (funcional y 6 no funcionales) con datos aumentados y parámetros del Cuadro 10 donde tuvo un desempeño de 92% frente a un promedio general de 74%, también para la clasificación binaria de los requisitos con los datos aumentados y parámetros del Cuadro 19 donde tuvo un promedio de 75% frente a un promedio general de 64% y para la clasificación de las 6 categorías de los requisitos no funcionales con los datos aumentados y parámetros del Cuadro 28 donde tuvo un promedio general de 56% frente a un promedio general de 43%:

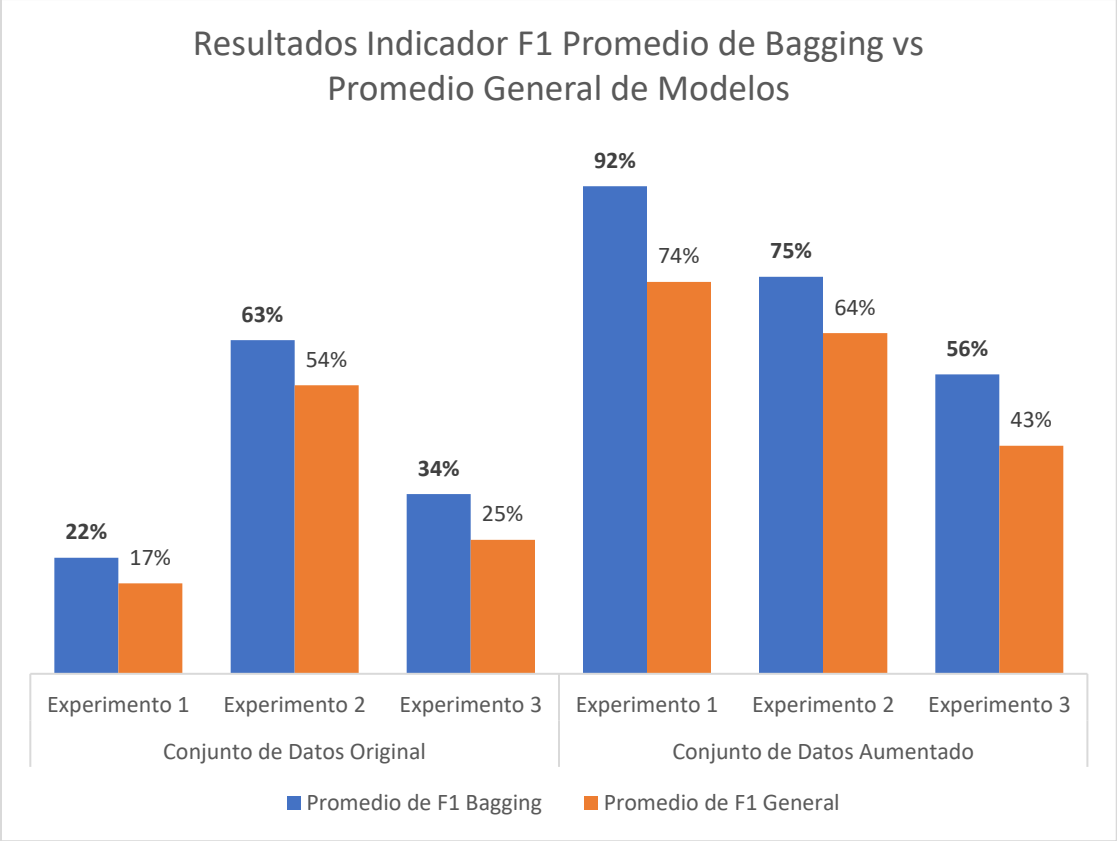

Figura 32. Comparación de indicador f1 mejor modelo vs promedio general de modelos de los tres primeros experimentos.

Asimismo, Bagging presenta el mejor tiempo de procesamiento que los demás modelos definitivos, como se observa en la Figura *33*, en el experimento 1, tuvo tiempo de 139,74 segundos con datos aumentados frente a 5156,45 segundos de la red neuronal convolucional (CNN); en el experimento 2, tuvo tiempo de 35,65 segundos con datos aumentados frente a 1729,90 segundos de la red neuronal convolucional (CNN); y en el experimento 3, tuvo tiempo de 35,18 segundos con datos aumentados frente a 203,68 segundos de la red neuronal convolucional (CNN):

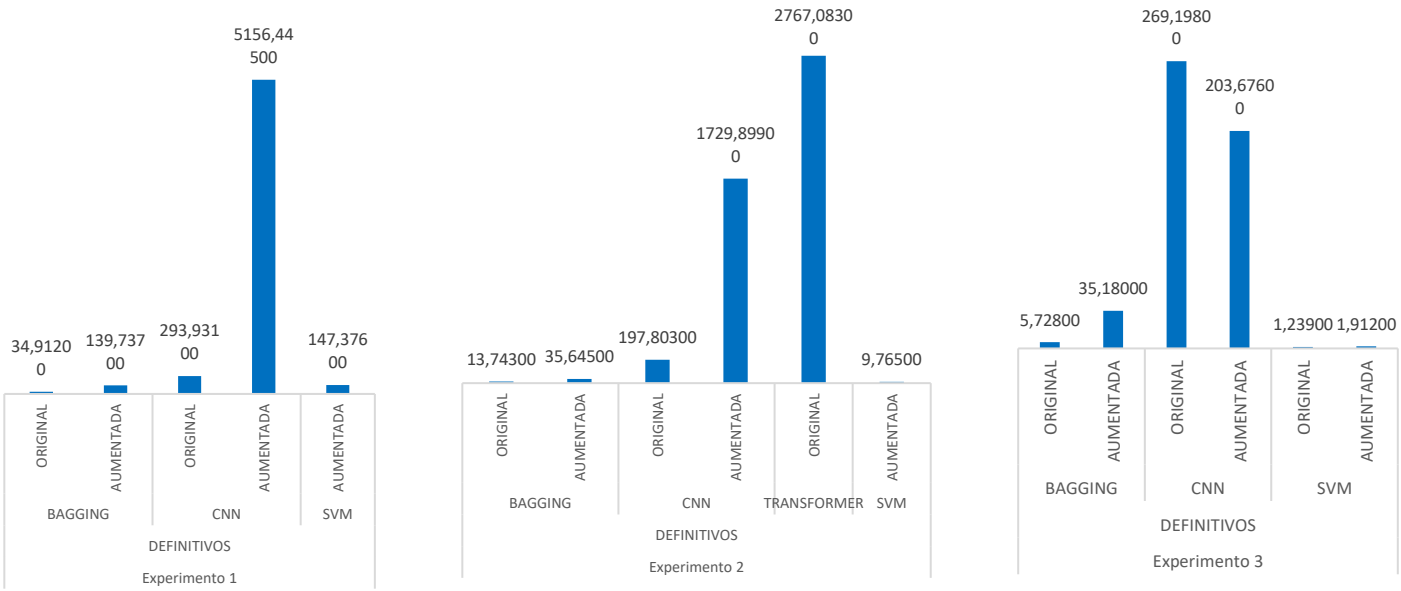

Figura 33. Tiempos de procesamiento en segundos de cada modelo definitivo.

Finalmente, el experimento con mejores resultados es el experimento 1 con un promedio en el indicador f1 de 92%, superando en un 17% al experimento 2, en un 36% al experimento 3, en un 38% al experimento 4 y en un 49% al experimento 5, como se observa en la Figura *34*.

Se evidencia una diferencia amplia respecto a los experimentos 3, 4 y 5, donde probablemente se deba a que los modelos tenían menor número de muestras para su entrenamiento (siendo más evidente en el experimento 5 que tiene limitante de 200 muestras) o, por otra parte, la aplicación de solo una técnica de representación numérica de los requisitos pudo afectar el desempeño de los modelos, de manera que en futuras investigaciones se podría fortalecer el número de muestras originales del conjunto de datos y/o se podrían utilizar distintas técnicas como n-gramas, embeddings pre entrenados y/o combinaciones de técnicas.

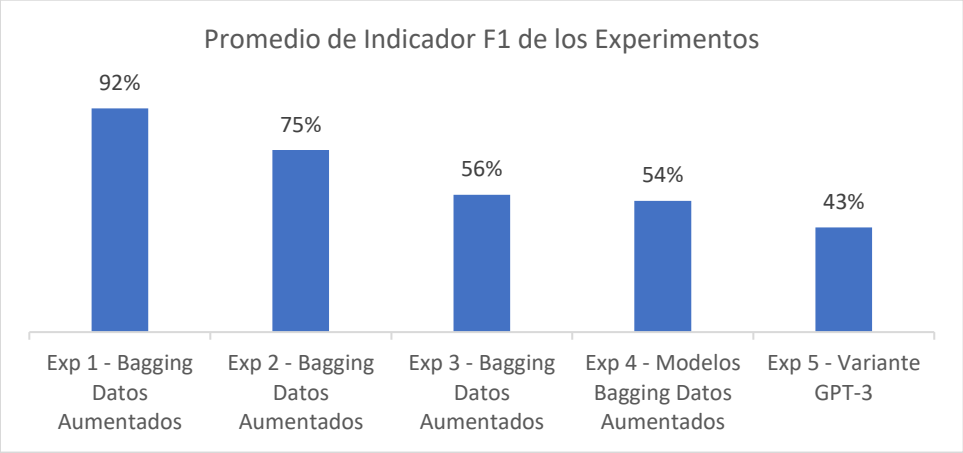

Figura 34. Comparación promedio indicador f1 de los cinco experimentos.

7. CONCLUSIONES

Como se observó en la Sección 1 de definición del problema, el proceso de clasificación de requisitos de software es una actividad fundamental para el éxito de proyectos de software y gracias a su optimización, se pueden mejorar los tiempos de implementación porque ahorra el análisis manual de requisitos a los analistas, arquitectos y desarrolladores, reduce el esfuerzo cognitivo requerido para dar sentido a las descripciones de los requisitos, aporta a que el proceso de desarrollo de software sea más eficiente, por consiguiente, aporta a disminuir el riesgo de extensión de los presupuestos en los proyectos de software. Retomando la Sección 5 de revisión de literatura, recientemente han aumentado los estudios enfocados a la automatización de la clasificación de requisitos de software, lo que confirma la importancia de la temática y la existencia de un área con altas opciones de exploración y aportes.

La automatización de la clasificación de requisitos de software se realiza mediante el aprendizaje automático con metodologías de excelente sustento técnico que permiten automatizar el proceso con dos enfoques: el aprendizaje supervisado y el aprendizaje profundo. Son más numerosos los estudios basados en aprendizaje supervisado, por tanto, fue el enfoque que se seleccionó.

En este trabajo se presenta la propuesta de generar un conjunto de datos en idioma español por la escasez de conjuntos de datos disponibles públicamente. El nuevo conjunto de datos puede ser utilizado en investigaciones de aprendizaje automático para apoyar la tarea de clasificación automática de requisitos de software descritos en español, al igual que en cualquier investigación que requiera un conjunto de datos de requisitos de software previamente etiquetado. El nuevo conjunto de datos en español contiene 2858 requisitos de software de diferentes dominios de software traducidos a partir de los requisitos descritos en ingles de Dalpiaz et al. (2019), Ferrari, Spagnolo y Gnesi (2017) y Lima et al. (2019), el cual se ajustó en algunas de las etiquetas para generar la analogía con las categorías consideradas en el estándar ISO/IEC 25010.

Este trabajo incluyó tareas de clasificación automática en 5 tipos de experimentos, clasificación de siete categorías (requisitos funcionales y las 6 categorías de no funcionales), clasificación entre dos categorías (requisitos funcionales y requisitos no funcionales), clasificación entre 6 categorías de no funcionales (se unificó en la categoría OTROS la MANTENIBILIDAD y la PORTABILIDAD que fueron las categorías con menor número de muestras), clasificación simultánea del requisito en funcional o no funcional y sus respectivas categorías, y una variante del primer experimento 1 utilizando GPT-3.

En el análisis exploratorio se observó que el conjunto de datos contiene una diferencia relevante en la cantidad de muestras de requisitos funcionales en comparación con los requisitos no funcionales, factor que influyó en el desempeño de todos los experimentos realizados, por lo que se procedió a aplicar la técnica de sobremuestreo SMOTE para balancear el conjunto de datos, mejorando los resultados sobre las clases minoritarias.

Los modelos de Bagging y Redes Neuronales Convolucionales (CNN) son los más recomendados para agilizar el proceso de definición de requisitos (ahorrar el análisis manual, reducir el esfuerzo cognitivo, aportar para desarrollo de software más eficiente y disminuir el riesgo de extensión de presupuestos.

El modelo pre-entrenado de GPT-3 en la versión completa es una opción prometedora para obtener mejores resultados, puesto que los resultados obtenidos con la versión beta y con apenas 200 muestras fueron similares a los a los obtenidos en los modelos del experimento 1, por lo que tener acceso completo a este modelo permitiría una mejor solución a la problemática.

Se recomienda como trabajo futuro, fortalecer el número de muestras originales del conjunto de datos para que los modelos cuenten con más datos en su entrenamiento y puedan generalizar mejor, probar con otras técnicas de representación de los textos, así como probar diferentes conjuntos de valores en los parámetros (otras arquitecturas, más iteraciones y demás).

Finalmente, el conjunto de datos generado disponible para la comunidad científica se encuentra en https://github.com/Sara-Abadia-Sarria/Software-requirements-data-set

BIBLIOGRAFÍA

AL-AZANI, S. y EL-ALFY, E.S.M., 2017. Using Word Embedding and Ensemble Learning for Highly Imbalanced Data Sentiment Analysis in Short Arabic Text. En: Paper que balancea una base de datos para clasificación de texto, sirve en caso que se necesite balancear el dataset., *Procedia Computer Science* [en línea], vol. 109, pp. 359-366. ISSN 18770509. DOI 10.1016/j.procs.2017.05.365. Disponible en: http://dx.doi.org/10.1016/j.procs.2017.05.365.

ALSHEMALI, B. y KALITA, J., 2020. Improving the Reliability of Deep Neural Networks in NLP: A Review. *Knowledge-Based Systems* [en línea], vol. 191. ISSN 09507051. DOI 10.1016/j.knosys.2019.105210. Disponible en: https://doi.org/10.1016/j.knosys.2019.105210.

ARORA, C., SABETZADEH, M., BRIAND, L. y ZIMMER, F., 2017. Automated Extraction and Clustering of Requirements Glossary Terms. *IEEE Transactions on Software Engineering*, vol. 43, no. 10, pp. 918-945. ISSN 00985589. DOI 10.1109/TSE.2016.2635134.

BAKER, C. y AL., E., 2019. Automatic Multi-class Non-Functional Software Requirements Classification Using Neural Networks. *IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 610-615.

BERRAR, D., 2018. Cross-validation. *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, vol. 1-3, no. April, pp. 542-545. DOI 10.1016/B978-0-12-809633-8.20349-X.

BINKHONAIN, M. y ZHAO, L., 2019. *A review of machine learning algorithms for identification and classification of non-functional requirements*. 1 abril 2019. S.l.: Elsevier Ltd.

BIRD, S. y TAN, L., 2021. Natural Language Toolkit. [en línea]. Disponible en: https://www.nltk.org/.

BROWN, T.B., MANN, B., RYDER, N., SUBBIAH, M., KAPLAN, J., DHARIWAL, P., NEELAKANTAN, A., SHYAM, P., SASTRY, G., ASKELL, A., AGARWAL, S., HERBERT-VOSS, A., KRUEGER, G., HENIGHAN, T., CHILD, R., RAMESH, A., ZIEGLER, D.M., WU, J., WINTER, C., HESSE, C., CHEN, M., SIGLER, E., LITWIN, M., GRAY, S., CHESS, B., CLARK, J., BERNER, C., MCCANDLISH, S., RADFORD, A., SUTSKEVER, I. y AMODEI, D., 2020. Language Models are Few-Shot Learners. [en línea], Disponible en: http://arxiv.org/abs/2005.14165.

CHAWLA, N. V., BOWYER, K.W., HALL, L.O. y KEGELMEYER, P., 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial INtelligence Research*, vol. 16, no. 2, pp. 321-357. ISSN 19395582. DOI 10.1002/eap.2043.

CONTRIBUTIONS, L.A., SAITO, D., DISCLOSURES, I., CO, T.P., OZAKI, D., SYSTEMS,

M.N., PHARMACIES, A., CHRONICLE, W., PHARMACIES, A., CHRONICLE, W., CONTRIBUTIONS, A., KAMI, M., DJ, R., WJ, M., CD, B., KR, T., TK, M., JD, M., JA, H., OCTOBER, A., JULY, A., SERVICES, M., PAYMENTS, O. y DECEMBER, A., 2019. Assessing the Use of Google Translate for Spanish and Chinese Translations of Emergency Department Discharge Instructions. , vol. 179, no. 4, pp. 2019-2021. DOI 10.1001/jama.2009.407.

DALPIAZ, F., DELL'ANNA, D., AYDEMIR, F.B. y ÇEVIKOL, S., 2019. Requirements classification with interpretable machine learning and dependency parsing. *Proceedings of the IEEE International Conference on Requirements Engineering*, vol. 2019-Septe, pp. 142-152. ISSN 23326441. DOI 10.1109/RE.2019.00025.

FERRARI, A., SPAGNOLO, G.O. y GNESI, S., 2017. PURE: A Dataset of Public Requirements Documents. *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*, pp. 502-505. DOI 10.1109/RE.2017.29.

FONG, V., 2018. Software Requirements Classification Using Word Embeddings and Convolutional Neural Networks. , no. June.

GLINZ, M., 2011. A glossary of requirements engineering terminology. *Standard Glossary of the Certified Professional for Requirements Engineering (CPRE) Studies and Exam*, vol. 1, pp. 56.

GRAMAJO, M.G., BALLEJOS, L. y ALE, M., 2019. Software Requirements Engineering through Machine Learning Techniques: A Literature Review. En: Se puede utilizar este artículo para justificar el uso del ML en la ingeniería de requisitos como gancho para el problema de clasificación. Así como un abrebocas de los métodos más utilizados., *2018 IEEE Biennial Congress of Argentina, ARGENCON 2018*, DOI 10.1109/ARGENCON.2018.8646010.

GULATI, K., KUMAR, S.S., SARATH, R., BODDU, K., SARVAKAR, K., KUMAR, D. y NOMANI, M.Z.M., 2021. Materials Today : Proceedings Comparative analysis of machine learning-based classification models using sentiment classification of tweets related to COVID-19 pandemic. En: Este paper sirve para justificar las técnicas de limpieza de texto utilizadas en el paso 1., pp. 1-4.

HEY, T., KEIM, J., KOZIOLEK, A. y TICHY, W.F., 2020. NoRBERT : Transfer Learning for Requirements Classification. En: ESTE ES, pp. 169-179. DOI 10.1109/RE48521.2020.00028.

IQBAL, T., ELAHIDOOST, P. y LUCIO, L., 2018. A Bird's Eye View on Requirements Engineering and Machine learning. , pp. 11-20. DOI 10.1109/APSEC.2018.00015.

ISO, 2011. *ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. S.l.: s.n.

KAMATH, U., LIU, J. y WHITAKER, J., 2019. *Deep Learning for NLP and Speech Recognition*. S.l.: s.n. ISBN 9783030145958.

KNAUSS, E. y OTT, D., 2014. (Semi-) automatic Categorization of Natural Language Requirements. , no. C, pp. 39-54.

KORDE, V., 2012. Text Classification and Classifiers:A Survey. *International Journal of Artificial Intelligence & Applications*, vol. 3, no. 2, pp. 85-99. ISSN 09762191. DOI 10.5121/ijaia.2012.3208.

KOWSARI, K., MEIMANDI, K.J., HEIDARYSAFA, M., MENDU, S., BARNES, L. y BROWN, D., 2019. *Text classification algorithms: A survey*. 2019. S.l.: s.n.

KRASNER, H., 2018. The cost of poor quality software in the us: A 2018 report. *Consortium for IT Software Quality, Tech*, vol. 10.

KUMAR, A. y JAIN, M., 2020. Ensemble Learning for AI Developers. *Apress*, DOI https://link.springer.com/book/10.1007%2F978-1-4842-5940-5.

LEE, J.S. y HSIANG, J., 2020. Patent claim generation by fine-tuning OpenAI GPT-2. En: Clasificación de texto usando GPT-2, *World Patent Information*, vol. 62, no. August. ISSN 01722190. DOI 10.1016/j.wpi.2020.101983.

LIAO, W., WANG, Y., YIN, Y., ZHANG, X. y MA, P., 2020. Improved sequence generation model for multi-label classification via CNN and initialized fully connection. *Neurocomputing* [en línea], vol. 382, pp. 188-195. ISSN 18728286. DOI 10.1016/j.neucom.2019.11.074. Disponible en: https://doi.org/10.1016/j.neucom.2019.11.074.

LIMA, M., VALLE, V., COSTA, E., LIRA, F. y GADELHA, B., 2019. Software engineering repositories: Expanding the PROMISE database. En: ESTE ES, *ACM International Conference Proceeding Series*, pp. 427-436. DOI 10.1145/3350768.3350776.

MAHMOUD, A. y WILLIAMS, G., 2016. Detecting, classifying, and tracing non-functional software requirements. *Requirements Engineering*, vol. 21, no. 3, pp. 357-381. ISSN 1432010X. DOI 10.1007/s00766-016-0252-8.

MUKHOPADHYAY, S., 2018. *Advanced data analytics using Python: with machine learning, deep learning and NLP examples* [en línea]. S.l.: s.n. ISBN 9781484234495. Disponible en: https://www.springer.com/it/book/9781484234495.

NASUKAWA, T. y YI, J., 2003. Sentiment analysis: Capturing favorability using natural language processing. En: Este paper se utilizó para decir que el análisis de datos del paso 1 es bueno-, *Proceedings of the 2nd International Conference on Knowledge Capture, K-CAP 2003*, no. March, pp. 70-77. DOI 10.1145/945645.945658.

NAVARRO-ALMANZA, R., JUUREZ-RAMIREZ, R. y LICEA, G., 2018. Towards Supporting

Software Engineering Using Deep Learning: A Case of Software Requirements Classification. En: Ejemplo de un paper que hace clasificación de requisitos funcionales y no funcionales, *Proceedings - 2017 5th International Conference in Software Engineering Research and Innovation, CONISOFT 2017*, vol. 2018-Janua, pp. 116-120. DOI 10.1109/CONISOFT.2017.00021.

ORNAT, S.M., 2015. Trabajo Fin de Grado Google Translate versus Human Translator : A Comparative Analysis of the A ccuracy in Google ' s. ,

POHL, K., 2016. Requirements engineering fundamentals: a study guide for the certified professional for requirements engineering exam-foundation level-IREB compliant. *Rocky Nook, Inc.*,

QUINTO, B., 2020. *Next-Generation Machine Learning with Spark*. S.l.: s.n. ISBN 9781484256688.

RADFORD, A. y col, 2018. Improving language understanding by generative pre-training. ,

RAHMAN, M.A., HAQUE, M.A., TAWHID, M.N.A. y SIDDIK, M.S., 2019. Classifying non-functional requirements using RNN variants for quality software development. *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, co-located with ESEC/FSE 2019*, pp. 25-30. DOI 10.1145/3340482.3342745.

RAJENDER KUMAR SURANA, C.S., SHRIYA, GUPTA, Di.B. y SHANKAR, S.P., 2019. Intelligent Chatbot for Requirements Elicitation and Classification. *2019 4th IEEE International Conference on Recent Trends on Electronics, Information, Communication and Technology, RTEICT 2019 - Proceedings*, pp. 866-870. DOI 10.1109/RTEICT46194.2019.9016907.

RESTREPO KLINGE, S., 2019. *Google Translate vs Traducción Humana : percepciones de ocho traductores en torno al papel de este traductor automático en su labor* [en línea]. S.l.: s.n. Disponible en: http://hdl.handle.net/10554/43243.

RICHARD BERNTSSON, B.S., TONY, G. y REGNELL, 2009. Quality requirements in practice: An interview study in requirements engineering for embedded systems. *International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer*, pp. 218-232.

SALUR, M.U. y AYDIN, I., 2020. A Novel Hybrid Deep Learning Model for Sentiment Classification. *IEEE Access*, vol. 8, pp. 58080-58093. ISSN 21693536. DOI 10.1109/ACCESS.2020.2982538.

SHIRABAD, S., 2005. The PROMISE Repository of Software Engineering Databases. En: Paper para referenciar la metodología de sequence to sequence en la clasificación de textos, *School of Information Technology and Engineering* [en línea], Disponible en: http://promise.site.uottawa.ca/SERepository/.

SITES, G., 2020. 3. Técnicas para identificar requisitos funcionales y no funcionales. [en línea], Disponible en: https://sites.google.com/site/metodologiareq/capitulo-ii/tecnicas-para-identificar-requisitos-funcionales-y-no-funcionales.

SKLEARN, 2021a. Sklearn - CountVectorizer. [en línea], Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.

SKLEARN, 2021b. Sklearn - TfidTransformer. [en línea], Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html.

TAIRA, B.R., KREGER, V., ORUE, A. y DIAMOND, L.C., 2014. A Pragmatic Assessment of Google Translate for Emergency Department Instructions. , DOI 10.1007/s11606-021-06666-z.

VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A.N., KAISER, Ł. y POLOSUKHIN, I., 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. Nips, pp. 5999-6009. ISSN 10495258.

VOGELSANG, A. y BORG, M., 2019. Requirements engineering for machine learning: Perspectives from data scientists. *Proceedings - 2019 IEEE 27th International Requirements Engineering Conference Workshops, REW 2019*, pp. 245-251. DOI 10.1109/REW.2019.00050.

WOLF, T., DEBUT, L., SANH, V., CHAUMOND, J., DELANGUE, C., MOI, A., CISTAC, P., RAULT, T., LOUF, R., FUNTOWICZ, M., DAVISON, J., SHLEIFER, S., VON PLATEN, P., MA, C., JERNITE, Y., PLU, J., XU, C., LE SCAO, T., GUGGER, S., DRAME, M., LHOEST, Q. y RUSH, A., 2020. Transformers: State-of-the-Art Natural Language Processing. , pp. 38-45. DOI 10.18653/v1/2020.emnlp-demos.6.