Universidad Zaragoza
1542

Trabajo Fin de Grado

Clasificación automática de requisitos de software: Creación del conjunto de datos y entrenamiento de modelos de lenguaje.

Automatic classification of software requirements: Creation of the dataset and training of language models.

Autor

Diego Murcia Martínez

Directores

José Javier Merseguer Hernaiz

Jorge Bernad Lusilla

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2024

AGRADECIMIENTOS

Quiero agradecer a mis tutores José y Jordi, por enseñarme, ayudarme y su esfuerzo a lo largo de todo este proyecto. A mis compañeros de la carrera, sobre todo a aquellos con los que he compartido mil vivencias desde el primer día que entramos en la universidad y que a día de hoy puedo llamar amigos. A mis amigos de toda la vida, que me han empujado a mejorar y especialmente a Alejandro, mi compañero incondicional de biblioteca. Por último y más importante, a mi familia y a mi pareja por apoyarme todos estos años, siendo pilares fundamentales que me sostuvieron para llegar hasta aquí.

Clasificación automática de requisitos de software: Creación del conjunto de datos y entrenamiento de modelos de lenguaje.

RESUMEN

Este trabajo fin de grado aborda el problema de clasificar, de manera automática, los requisitos de los sistemas software. Este es un paso previo, que debemos solucionar, antes de automatizar muchas otras tareas del proceso de desarrollo del software. Entre ellas, por ejemplo, la de generar de manera automática dichos requisitos partiendo de estándares, manuales y/o procedimientos consolidados. No cabe duda que, en el contexto actual, la inteligencia artificial debe jugar un papel fundamental en estos procesos de automatización. Concretamente, las técnicas de procesamiento del lenguaje natural, apoyadas en los modelos de lenguaje, son el pilar que soportará dicha automatización.

Esta memoria documenta cada una de las fases que se han llevado a cabo para medir la efectividad de diversos modelos de lenguaje a la hora de clasificar requisitos de software en funcionales y no funcionales. Estas fases incluyen desde la generación de un conjunto de datos, válido para el entrenamiento, validación y test de los modelos, hasta la estimación de valores estadísticos que midan el desempeño del *dataset* y de los propios modelos.

El *dataset* trata de replicar los múltiples y diferentes contextos que aparecen en el mundo del desarrollo de software. Por ejemplo, los requisitos funcionales superan habitualmente en número a los no funcionales, de manera amplia. Esta característica diferencia a nuestro proyecto, ya que desde nuestro conocimiento, el *dataset* desarrollado es el más extenso, y realista, de los hasta ahora propuestos por la comunidad.

La experimentación se ha diseñado de forma que pueda ser replicada por otros investigadores. Pudiendo además recabarse información del comportamiento de los distintos modelos en diversas situaciones. También se rehacen experimentos ya existentes en la literatura, posibilitando así la comparación con otros trabajos.

El *dataset*, el código y los resultados relativos a la realización de este proyecto pueden encontrarse en el repositorio de GitHub «DaReC[1]».

[1] `https://github.com/diegomurciamart/DaReC`.

ABSTRACT

This final degree project addresses the problem of automatically classifying the requirements of software systems. This is a preliminary step that must be solved before automating many other tasks in the software development process. These include, for example, the automatic generation of these requirements based on standards, manuals and/or consolidated procedures. There is no doubt that, in the current context, artificial intelligence must play a fundamental role in these automation processes. Specifically, natural language processing techniques, supported by language models, are the pillar that will support such automation.

This report documents each of the phases that have been carried out to measure the effectiveness of various language models in classifying software requirements into functional and non-functional. These phases include from the generation of a dataset, a set of data valid for training, validation and testing of the models, to the estimation of statistical values that measure the performance of the dataset and of the models themselves.

The dataset tries to replicate the many different contexts that appear in the software development world. For example, functional requirements usually outnumber non-functional requirements by a wide margin. This characteristic differentiates our project, since to our knowledge, the dataset developed is the most extensive, and realistic, of those so far proposed by the community.

The experimentation has been designed in such a way that it can be replicated by other researchers. It is also possible to gather information on the behaviour of the different models in different situations. Experiments already existing in the literature are also reworked, thus enabling comparison with other works.

The dataset, code and results related to this project can be found in the GitHub repository «DaReC[1]».

Índice

Capítulo 1

Introducción

1.1. Motivación

El despegue de la Inteligencia Artificial (IA) [1], como una de las tecnologías más importantes a nivel global [2], obliga a las empresas a desarrollar sus productos y su modelo de trabajo aprovechando esta herramienta para no quedarse por detrás de sus competidoras.

La necesidad de aceptar a esta relativamente nueva compañera de trabajo se acentúa aún más en sectores que deben ser tecnológicamente punteros, como las telecomunicaciones, la industria, el transporte, las finanzas o el sector energético. La ingeniería del software (IS) [3] es uno de los campos donde la IA también puede ser de gran utilidad. Cada una de las fases que componen el desarrollo de un sistema software [4], desde la elicitación de sus requisitos hasta el propio despliegue y mantenimiento, puede verse beneficiada por la IA. Por ejemplo, en el ámbito de la captura de requisitos y el análisis, la IA podría ayudar recabando las necesidades de los usuarios, y traduciéndolas a un lenguaje formal. En el ámbito del diseño software, podría ayudar aconsejando al desarrollador la arquitectura más adecuada para sus sistemas. En el ámbito de las pruebas, sería interesante que la IA proporcionase diferentes entornos de *test* para validar las soluciones. En el ámbito del mantenimiento, sería útil que la IA fuese capaz de identificar problemas a partir de información externa para mejorar el software [5].

Por lo tanto, el desafío que se plantea es el de llegar a tener una relación simbiótica con la IA, que permita aprovechar al máximo su potencial para facilitar y mejorar el trabajo hecho por personas, pero manteniendo siempre el espíritu crítico humano, que no dé por buena automáticamente cualquier solución propuesta por la IA.

En esa ambiciosa meta de implementar la IA a lo largo de toda la cadena productiva del desarrollo del software, este trabajo de fin de grado se centra en el primer paso, es decir, en la etapa de captura de requisitos [6]. En particular, se pretende la clasificación

automática de los requisitos del software en funcionales y no funcionales. Los requisitos funcionales son aquellos que establecen una funcionalidad o comportamiento del sistema, y por tanto describen una interacción entre el sistema y el entorno que es independiente de la implementación. Los requisitos no funcionales son atributos de calidad y criterios para juzgar cómo opera un sistema. Por ejemplo, el número de usuarios que debe soportar el sistema o el tiempo de respuesta que debe ofrecer un servicio concreto.

1.2. Objetivos

El primer objetivo es medir la efectividad de los modelos de lenguaje [7, 8, 9] a la hora de clasificar los requisitos en funcionales y no funcionales. Este objetivo es previo a la finalidad de la implementación de la IA en el ámbito de los requisitos. Ciertamente la finalidad sería lograr una generación completa de los requisitos a partir de especificaciones existentes en lenguaje natural, como por ejemplo manuales, procedimientos o estándares. Para lograr esa generación automática, debe cimentarse una buena base que garantice que los modelos de lenguaje comprenden los requisitos y es en este punto en el que se realiza este trabajo.

El segundo objetivo, no por ello menos importante, que se trata de alcanzar en este trabajo es el de proporcionar a la comunidad un nuevo *dataset* o conjunto de datos. Queremos que este *dataset* posea unas características diferentes a los existentes y sea válido para el entrenamiento de modelos de lenguaje. Es decir, la propia generación del conjunto de datos es de interés científico por sí misma.

1.3. Fases del proyecto

- Fase 1. Análisis del estado del arte.
- Fase 2. Elaboración del *dataset* o conjunto de datos. Se creará un conjunto de datos ofreciendo un repositorio de requisitos público.
- Fase 3. Entrenamiento de modelos de lenguaje. Se desarrollará el entorno de test y se experimentará con los modelos BERT [8], RoBERTa [10] y DeBERTa [11], realizando para cada uno una serie de pruebas diferenciadas.
- Fase 4. Fase de pruebas. Se recogen resultados estadísticos que miden la tasa de éxito de los modelos y se analizan para estimar la utilidad, comparándolos entre ellos y con otros estudios ya existentes.
- Fase 5. Redacción de la memoria.

Capítulo 2

Estado del arte

2.1. Arquitectura *Transformer*

Los modelos de lenguaje de gran tamaño, también conocidos por sus siglas en ingés LLM (Large Language Models), son modelos de aprendizaje con una gran cantidad de parámetros y preentrenados con abundantes datos. La arquitectura de *transformers* [7] subyacente es un sistema de redes neuronales compuesto por un codificador (o *encoder*) y un decodificador (o *decoder*), diseñado para extraer el significado de una secuencia de texto y comprender las relaciones entre las palabras y frases que la conforman. Los *transformers* son capaces de realizar un aprendizaje auto supervisado y a través de este proceso aprenden a procesar el lenguaje natural humano [12, 13].

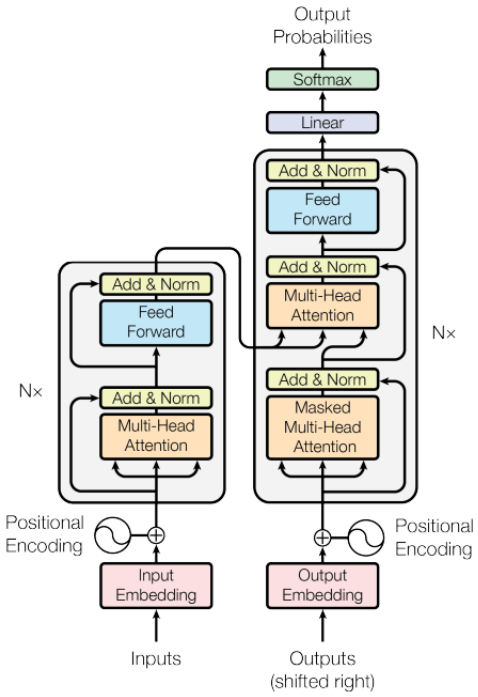

Figura 2.1: Arquitectura de Transformers. Imagen extraída de [7].

El *encoder* es el bloque izquierdo de la arquitectura en la Figura 2.1, que toma la entrada en texto y la transforma en una representación vectorial de las palabras (*embedding*), en la que además, las palabras con un significado semántico similar se encuentran próximas entre sí dentro del espacio vectorial. La siguiente capa (*Multi-Head attention*) permite que el modelo conozca con qué palabras de la secuencia textual está relacionada la palabra que se procesa. Por último, después de que la autoatención proporcione representaciones contextuales de la entrada, la red *Feed-Forward* añade otra capa de complejidad al aplicar funciones no lineales a estas representaciones. Esto permite al modelo aprender patrones más complejos en los datos y refinar las características extraídas por la capa de autoatención.

El *decoder* toma como *input* la salida al final del *encoder* y los *tokens* generados previamente, para predecir el siguiente *token* de salida. El bloque *Masked Multi-Head Attention* permite al decodificador centrarse en varias partes de la secuencia de entrada a medida que genera la salida en cada paso, asegurándose de que no tiene acceso a la información de los tokens futuros en la secuencia de entrada.

2.2. Elección del modelo de lenguaje

El primer punto a tratar en este trabajo ha sido la decisión de qué tipo de modelo de lenguaje utilizar, puesto que no todos operan del mismo modo. Los dos modelos más populares son BERT y GPT [9]:

- BERT (Bidirectional Encoder Representations from Transformers). Forma parte de los modelos de lenguaje que emplean la parte izquierda de la arquitectura *transformer*, el *encoder*. Los modelos similares o basados en este, son capaces de realizar multitud de tareas diferentes del procesado natural del lenguaje. Para ello, BERT se pre-entrena con 3300 millones de palabras provenientes de *Wikipedia* (2500M) y *BooksCorpus* (800M) y se le puede añadir otra capa de entrenamiento específico para que sea capaz de realizar trabajos concretos (*fine-tuning*). Dada su arquitectura bidireccional, observa palabras anteriores y posteriores a una palabra considerada clave en la frase, por lo que es capaz de reconocer el contexto de las oraciones. Tiene un tamaño de 110 millones de parámetros en su versión *base* y 330 millones de parámetros en su versión *large*.

- GPT (Generative Pre-trained Transformer). Es un modelo de lenguaje autorregresivo que utiliza la parte del decodificador del *transformer*. Está diseñado para generar automáticamente textos escritos, buscando para ello la palabra que mejor se adapte a la continuidad de la frase. Para ello observa

las palabras anteriores y genera una salida en función de la probabilidad de aparición de la siguiente palabra, aprendida durante el entrenamiento. Tal y como se alude en [14], cuando este tipo de modelos de lenguaje se entrenan con un conjunto de datos suficientemente grande y diverso, son capaces de cumplir tareas exitosamente en dominios de problemas bastante distintos, como por ejemplo traducción, síntesis o comprensión de escritos. Por ello, GPT-3 fue entrenado con 570 GB de información en lenguaje natural de múltiples sitios web de internet y tiene un tamaño de hasta 175 mil millones de parámetros en su versión más extensa.

En resumen, los modelos GPT tienen una finalidad clara de generación de texto, mientras que los modelos BERT pueden utilizar con mucho éxito el *fine-tuning* para refinar con un conjunto de datos más pequeño el grueso del modelo ya entrenado, obteniendo así salidas más relevantes para problemas muy diversos. Los modelos GPT actuales son gigantescos mientras que los BERT poseen versiones que pueden utilizarse en entornos sin un coste de computación excesivamente elevado, como *Google Colab*, lo cual resulta interesante en el ámbito académico en el que se desarrolla el trabajo, donde no se dispone de mucha capacidad de cálculo. Las redes neuronales se suelen ejecutar usando GPU (Graphic Processing Unit) debido a las capacidades de paralelización de cálculo que tienen estas unidades.

Existe un estudio que hace una revisión sistemática de la literatura acerca de los LLM en la ingeniería de software [15] que revela los casos de éxito o de uso más extendido de los distintos tipos de modelos y que por ello revela factores influyentes a la hora de seleccionar un modelo:

- Para resolver el problema de la ambigüedad anafórica, los modelos GPT han demostrado ser extremadamente precisos, llegando incluso a efectividades del 100 % en algunos experimentos [16]. La ambigüedad anafórica surge cuando un lector puede comprender un requisito de distintas maneras o cuando varios lectores pueden comprender el requisito de distintas maneras y resolver este problema implica dirimir cuál es la interpretación adecuada.

- Para categorizar requisitos en funcionales y no funcionales, se han desarrollado modelos basados en BERT con *fine-tuning* con los que se logra una gran eficacia, especialmente en escenarios *zero-shot*. El *zero-shot learning* es una técnica de aprendizaje automático que permite a un modelo conocer y clasificar objetos o conceptos para los cuales no ha sido explícitamente entrenado, por lo que debe utilizar para ello información aprendida de clases conocidas. Se basa en la capacidad de los modelos de generalización y transferencia del conocimiento [17].

– Para identificar términos concretos dentro de los requisitos utilizados en contextos distintos, el uso de BERT es el más extendido ya que proporciona una comprensión del lenguaje más profunda.

– Existen estudios basados en BERT que resuelven el problema de la detección de la correferencia con una efectividad de hasta el 96,10 % [18]. Este problema surge cuando distintas expresiones lingüísticas se refieren a la misma entidad del mundo real. Esto da lugar a malentendidos sobre terminología técnica y afecta negativamente a la legibilidad y comprensibilidad de los requisitos.

– Para lograr automatizar la trazabilidad del software, nuevamente los estudios existentes utilizan BERT, superando en precisión a las técnicas existentes y siendo capaz adaptarse a distintos ámbitos sin necesidad de un entrenamiento específico para cada proyecto.

En definitiva, por todo lo anteriormente expuesto, la elección final de los modelos de lenguaje que se van a utilizar es aquellos basados en BERT, y más concretamente RoBERTa [10] y DeBERTa [11] además del propio BERT [8].

2.3. BERT, RoBERTa Y DeBERTa

Tanto RoBERTa (*Robustly Optimized BERT Pretraining Approach*) como DeBERTa (*Decoding-enhanced BERT with Disentangled Attention*) siguen una estructura de preentrenamiento y fine-tuning muy similar a la introducida por BERT. Sin embargo, proponen modificaciones para mejorar el rendimiento o la precisión de este modelo de lenguaje en situaciones generales y específicas. Los tres modelos cuentan con versiones *base* y *large*, según se prefiera utilizar modelos más manejables o más potentes.

2.3.1. BERT

La innovación introducida por BERT con respecto a modelos anteriores fue la posibilidad de realizar el preentrenamiento con texto sin etiquetar condicionando para ello conjuntamente el contexto izquierdo y el derecho. El preentrenamiento es una técnica utilizada en el procesado de lenguaje natural (también conocido como PNL o NLP por sus siglas en inglés), que consiste en entrenar el modelo de lenguaje con una gran cantidad de texto sin etiquetar. El objetivo del preentrenamiento es generar un modelo que haya logrado aprender los patrones propios y estructuras del lenguaje real, permitiéndose así conseguir una comprensión profunda del lenguaje a partir de

la cual se obtendrán respuestas relevantes para el contexto y coherentes. Al añadir contexto bidireccional al preentrenamiento, se logra que BERT tenga utilidad en un amplio rango de tareas añadiendo tan solo una capa extra de entrenamiento específico o *fine-tuning*. El *fine-tuning* es una técnica que permite a los modelos de lenguaje preentrenados adaptarse a una tarea o dominio específicos. reentrenando el modelo con datos relevantes para dicha tarea.

Los modelos unidireccionales previos al lanzamiento de BERT (Glove [19], FastText [20, 21] o Word2Vec [22]) no eran capaces de comprender las oraciones como un todo, lo cual según los desarrolladores del modelo era subóptimo especialmente a la hora de realizar el fine-tuning. En contraposición a los modelos anteriores basados en *word-embeddings* (que asumen que todas las palabras tienen la misma representación vectorial independientemente del contexto; por ejemplo la palabra casco estaría representada por el mismo vector tanto si se refiere a la pieza de protección de la cabeza, como si se refiere a un auricular), los autores del artículo de BERT [8] proponen un modelo para afinar la comprensión del contexto denominada *masked language model* (MLM) [23]. Fundamentalmente, el modelo consiste en un *encoder* formado por múltiples capas basadas en *transformers*. Utiliza máscaras para omitir ciertas palabras deliberadamente (el 15 % según [8]), forzando al modelo a rellenar los espacios que quedan en blanco, aumentando de este modo su comprensión de las relaciones entre las palabras y la coherencia general del texto. A diferencia de los *word-embeddings* tradicionales, BERT no dispone de una tabla fija en la que encontrar las representaciones de las palabras, por lo que es necesario el contexto completo para hallar el *embedding* de cada una de las palabras. La secuencia de texto de entrada se

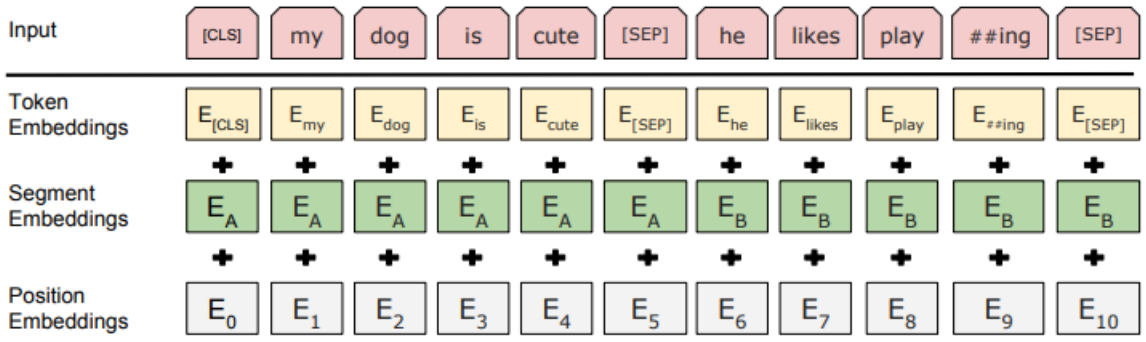

Figura 2.2: Ejemplo de tokenización de dos frases. Imagen extraída de [8]

divide en *tokens* de un vocabulario predefinido. Este vocabulario se define de antemano y se basa en WordPiece [24], un algoritmo de tokenización que genera tokens de las subpalabras más frecuentes. En primer lugar se realiza la partición del texto en palabras o subpalabras y a continuación se agregan tokens para indicar de qué frase proviene

cada uno de las particiones, así como su posición. El término «frase» puede referirse a un trozo de texto contiguo en lugar de a una frase lingüística, y el término «secuencia» se refiere a la secuencia de tokens de entrada del modelo, que puede representar una o dos frases que se empaquetan conjuntamente. Existen tokens especiales que fragmentan el texto para la comprensión por parte del modelo:

- CLS: el primer token de cada secuencia. El estado final de este token es usado como representación de la secuencia agregada en tareas de clasificación.

- SEP: se utiliza para separar y diferenciar frases que se introducen conjuntamente al modelo en una sola secuencia.

La construcción de la secuencia de tokens puede verse en la Figura 2.2.

En lugar de utilizar los métodos tradicionales para el preentrenamiento de izquierda a derecha o de derecha a izquierda, BERT se preentrena llevando a cabo dos tareas sin supervisión (es decir, a partir de datos sin etiquetar):

1. MLM: que ya se ha explicado brevemente con anterioridad. Dada la imposibilidad de realizar un entrenamiento bidireccional debido a que el estándar condicional de los modelos de lenguaje permitirían al modelo «verse a sí mismo», se enmascaran aleatoriamente un 15 % de los tokens para que BERT tenga que predecirlos. En la práctica esto equivale a realizar un entrenamiento bidireccional. Aparece un nuevo token especial (MASK), que provoca un desajuste en el *fine-tuning*, ya que no aparece en esta segunda fase del entrenamiento. Para mitigar esto, no siempre se enmascara el token: el 80 % de las veces sí se utiliza MASK, el 10 % se utiliza un token aleatorio y el 10 % restante no se modifica nada.

2. *Next sentence prediction* (NSP): para entrenar un modelo que comprenda la relación entre las frases, se realiza un preentrenamiento con el fin de predecir cuál es la frase que sigue a otra, en un contexto binario. En concreto, al elegir las frases A y B para cada ejemplo de preentrenamiento, el 50 % de las veces B es la frase siguiente a A (etiquetada como *IsNext*), y el 50 % de las veces es una frase aleatoria del corpus (etiquetada como *NotNext*). Pese a la simplicidad, en [8] se demuestra que este proceso es muy beneficioso para resolver posteriormente problemas de «respuesta a preguntas» o de «inferencia del lenguaje natural».

El fine-tunning es sencillo, ya que el mecanismo de autoatención del Transformer permite a BERT modelar muchas tareas posteriores, ya impliquen un solo texto o pares de textos, intercambiando las entradas y salidas apropiadas. Para cada tarea, simplemente se introducen las entradas y salidas específicas de la tarea en BERT y se

ajustan todos los parámetros de extremo a extremo. A la entrada, un par de frases A y B del preentrenamiento equivalen a:

- Un par de sentencias en tareas de parafraseado.
- Un par hipótesis-premisa en secuencias lógicas.
- Un par pregunta-respuesta.
- Un par texto-vacío en labores de clasificación o etiquetado.

A la salida las representaciones de los tokens se introducen en una capa de salida para tareas a nivel de token, como el etiquetado de secuencias o la respuesta a preguntas. La representación del token especial CLS, se introduce en una capa de salida para la clasificación o el análisis de sentimientos. En comparación con el preentrenamiento, el ajuste fino es relativamente barato y pueden replicarse experimentos partiendo del mismo modelo preentrenado en un lapso de tiempo relativamente corto.

2.3.2. RoBERTa

Además del *dataset* con el que se entrena BERT (*Wikipedia* en inglés + *BooksCorpus*), que suma 16GB de datos, RoBERTa aumenta la cantidad de información hasta ser 10 veces más grande con: 76 GB de datos de *Common Crawl* (CC-News), 38 GB de datos de *OpenWebText*, y 31 GB de datos de un *subset* de *Common Crawl* llamado Stories. Esto se debe a que se estima que el estilo de preentrenamiento de BERT depende en gran medida de la cantidad de datos con los que se realiza, por lo que en el estudio que presenta RoBERTa se opta por aumentarlos. Las diferencias que presenta RoBERTa con respecto a BERT se encuentran en la fase de preentrenamiento:

- *Static* vs *Dynamic Masking*: en BERT se enmascara aleatoriamente una sola vez al procesar los datos de entrada. RoBERTa propone realizar el enmascarado tantas veces como número de instancias de entrenamiento haya para que cada vez se realice de forma distinta, es decir, que cada vez los tokens omitidos sean diferentes. Esto presenta una ligera mejora frente al modo de operación de BERT.
- Abandono del NSP: utilizando cuatro formas de entrenamiento con y sin el mecanismo de mecanismo de predicción NSP, que se pueden consultar en [10], se demuestra que eliminándolo se obtienen rendimientos iguales o superiores a los obtenidos por BERT. Esta es la modificación más significativa introducida por RoBERTa, ya que llevando a cabo una fase menos de entrenamiento, se alcanzan resultados similares.

– Entrenamiento con grandes *batches*: al entrenar un modelo de lenguaje se introducen a la arquitectura *Transformers* particiones de la secuencia de tokens denominadas *batches*, una detrás de otra. En BERT el tamaño del *batch* es de 256 secuencias y se realiza el entrenamiento durante 1 millón de pasos. Con el experimento de RoBERTa se demuestra que hay resultados similares o ligeramente mejores si se disminuye el número de pasos, a cambio de aumentar el tamaño del *batch* de entrada. Si se aumenta mucho el tamaño del *batch*, los resultados empeoran, así que se debe llegar a un compromiso *batch-size*/número de pasos.

– Cambios en la forma de codificar el texto: el diccionario empleado para el entrenamiento de RoBERTa cuenta con 50000 subpalabras, frente a las 30000 que contiene el de BERT. Mientras que el algoritmo para encontrar las subpalabras más frecuentes en BERT es *WordPiece*, para RoBERTa, el algoritmo utilizado es Byte-Pair Encoding (BPE), presentado en [25].

2.3.3. DeBERTa

La primera discrepancia entre DeBERTa y los otros dos modelos tratados en las subsecciones anteriores se encuentra en la cantidad de datos de entrenamiento. DeBERTa se entrena con 78 GB provenientes de las mismas fuentes que RoBERTa pero descartando la parte del *dataset* correspondiente a CC-News y aplicando un mecanismo de eliminación de datos duplicados. A pesar de contar con la mitad de datos de entrenamiento, presenta resultados mejores que RoBERTa, con la desventaja de que requiere utilizar el doble de GPU RAM que RoBERTa y hasta el triple que BERT [26]. Las mayores innovaciones introducidas por DeBERTa se encuentra en el mecanismo de atención empleado y en la mejora del decodificador de máscaras [11, 27]:

– *Disentangled attention*: mientras que en los modelos de lenguaje anteriores se utilizaba un solo vector para representar la suma de los *embeddings* de la palabra y su posición textual, en DeBERTa se utilizan dos vectores. Aparecen por separado el significado y la posición de cada palabra. Los pesos asignados a las palabras se calculan entonces empleando dos matrices, una para el contenido y otra para la posición relativa. Este enfoque se basa en la idea de que el significado real de una palabra depende enormemente de la posición que ocupa en el texto. Por ejemplo la relación de dependencia existente entre las palabras «número» y «complejo» es mucho mayor si se encuentran juntas que si se encuentran en frases distintas.

– *Enhanced mask decoder*: al igual que BERT y RoBERTa, DeBERTa emplea el

mecanismo MLM para su preentrenamiento. Por un lado el MLM general emplea los tokens que rodean al token enmascarado para predecir cuál es la palabra oculta tras la máscara. Por otro, DeBERTa aprovecha tanto la información de contenido como la de posición de las palabras en el contexto. El mecanismo de *disentangled attention* permite considerar los contenidos y las posiciones relativas de las palabras de contexto, pero no las posiciones absolutas de estas palabras, que en muchos casos son cruciales para la predicción. La implicación es que las oraciones con contextos locales similares pero roles sintácticos diferentes dependen en gran medida de la posición absoluta, lo que subraya la importancia de tener en cuenta la posición absoluta en el modelo. DeBERTa incorpora la incrustación de la posición absoluta de las palabras justo antes de la capa *Softmax*, que descodifica las palabras buscadas basándose en la incrustación contextual agregada del contenido y la posición de las palabras, mejorando así el proceso de enmascarado.

Capítulo 3

Conjunto de datos

3.1. Introducción y motivación

Un conjunto de datos, también llamado *dataset*, consiste en una colección de datos, habitualmente tabulados, a partir de los cuales se entrenan los modelos de lenguaje (LLM). Es un elemento fundamental de dicho entrenamiento, puesto que es un factor crítico que determina la capacidad de generalización, la efectividad y el rendimiento del LLM. A día de hoy no existen métricas para evaluar la calidad de un *dataset* de manera fiable, sino que la revisión debe ser efectuada por uno o varios expertos. El conjunto de datos desarrollado en este trabajo parte de otros existentes en la literatura, que en su día fueron evaluados por sus autores. Este ha sido evaluado por su autor.

A la hora de construir el *dataset*, se ha utilizado la información recabada por otros estudios y conjuntos de datos, que se analizan en la Sección 3.2. Si bien es cierto que existen estudios en la materia de clasificación de requisitos, uno de los experimentos que se va a realizar en este trabajo requiere disponer de las descripciones de los proyectos de los que provienen los requisitos y no se han encontrado otros conjuntos de datos con esta característica. Por lo tanto, se han realizado modificaciones en los *datasets* existentes para alinearlos con los objetivos de nuestro proyecto, optimizando su adecuación para el entrenamiento de los modelos de lenguaje que vamos a emplear. Asimismo, trabajar sobre un *dataset* preexistente reduce el tiempo y esfuerzo necesarios, facilitando un proceso de construcción más eficiente y controlado. Esta estrategia ofrece un equilibrio entre aprovechar recursos ya disponibles y adaptarlos para lograr resultados más específicos y efectivos, y es por ello que se elige como la óptima para la realización del trabajo.

3.2. Conjuntos de datos existentes

En la búsqueda del *dataset* más adecuado para la clasificación de requisitos en funcionales y no funcionales, se han revisado multitud de publicaciones [18, 28, 29, 30, 31, 32, 33, 34] que emplean sus propios conjuntos de datos para alcanzar diversos objetivos. A continuación se presentan algunos de los *datasets* provenientes de estos estudios previos que, en algún momento del proyecto, se han tenido en cuenta como opciones para el entrenamiento de los modelos de lenguaje. Se incluyen también la finalidad del estudio en el que se emplean, los motivos por los que se han considerado válidos o por los que se han finalmente descartado y otras características.

3.2.1. *Datasets* de dominio privado

Muchos estudios, dada su finalidad práctica para casos empresariales e industriales concretos o debido a estar desarrollados directamente por alguna corporación, utilizan conjuntos de datos extraídos de proyectos internos de la empresa, por lo que no son accesibles para el público general. Este tipo de conjuntos de datos, dado el carácter académico de este trabajo han sido descartados.

Otro caso similar a este ocurre cuando en algún estudio se emplean proyectos de repositorios de empresas pero no se citan los proyectos concretos cuyos requisitos forman parte del *dataset*, por lo que a efectos prácticos estos conjuntos de datos son nuevamente inaccesibles para el público general, especialmente si el repositorio de la empresa es grande. Un ejemplo de este acontecimiento se encuentra al revisar una publicación que trata de resolver el problema de la correferencia [18], puesto que menciona que extrae los datos de 21 proyectos de *CMB*[1] pero no menciona de cuáles.

3.2.2. PROMISE

Este conjunto de datos tiene un uso muy extendido dado que aparece en multitud de publicaciones acerca de los distintos usos de los modelos de lenguaje y más concretamente aquellos basados en BERT. Se utiliza mayoritariamente para clasificación de requisitos, por ejemplo:

- En un estudio que propone el *prompt-learning* para llevar a cabo la clasificación en funcionales y no funcionales [28].
- En un estudio que propone agregar una capa de interpretabilidad al *machine learning* para facilitar a los analistas de requisitos su clasificación [29].

[1] CMB: China Merchants Bank

– En un estudio que propone utilizar BERT tanto para clasificar requisitos en funcionales y no funcionales, como para clasificar las distintas subdivisiones que existen dentro de los no funcionales [30].

Se trata de un conjunto de datos relativamente pequeño, conformado por tan sólo 625 requisitos provenientes de 15 proyectos y recabado por estudiantes. De los 625, 255 son funcionales y los 370 restantes no funcionales. Los requisitos no funcionales se subdividen en las siguientes categorías: disponibilidad, tolerancia a fallos, legales, aspecto, mantenimiento, operacionales, rendimiento, portabilidad, escalabilidad, seguridad y usabilidad. Aunque este *dataset* fue considerado útil como punto de partida, el número de proyectos y de requisitos que contiene no llega a las aspiraciones de nuestro proyecto y la subdivisión en categorías de los requisitos no funcionales aporta información ajena al dominio del problema tratado. Además, según [28], PROMISE está demasiado obsoleto para las técnicas de procesado del lenguaje natural y las tendencias de mercado actuales, por lo tanto este conjunto de datos se descartó. A raíz de la obsolescencia del *dataset* surgen nuevas publicaciones que tratan de expandir PROMISE, siendo [31] una de las más destacadas. En este estudio se presenta una tasa de aumento de la cantidad de datos del 55,04 % (344 requisitos extras, divididos en 189 funcionales y 155 no funcionales). Sin embargo esta ampliación solo supone una mejora en algunas métricas de validación de clasificación de requisitos y según qué algoritmo de clasificación se esté utilizando (por ejemplo si se emplea árbol de decisión no se mejora ninguna métrica, pero si se utiliza el algoritmo de Naïve Bayes se observa una mejora en la precisión y *f1-score* en la detección de requisitos funcionales y una mejora del *recall* en la detección de no funcionales), por lo que en este trabajo se opta por recabar el conjunto de datos desde otras fuentes ajenas totalmente a PROMISE y sus sucesores.

3.2.3. NFR-SO y NFR-Review

Ambos *datasets* son la fuente de entrenamiento del experimento realizado en [28].

– NFR-SO (*nonfunctional requirements from StackOverflow*) es un conjunto de datos etiquetados recabado del sitio web para programadores *StackOverflow*[2]. Cada muestra de NFR-SO es el contenido de una pregunta del foro, etiquetado en una de las 7 categorías en las que el estudio decide dividir los requisitos no funcionales: disponibilidad, rendimiento, mantenibilidad, portabilidad, escalabilidad, seguridad y tolerancia a fallos. Mayoritariamente las

[2] *https://stackoverflow.com*

etiquetas son añadidas manualmente por los visitantes del sitio web, por lo que se consideran creíbles. Aún así, existen casos en los que aparecen más de una etiqueta para cada muestra, por lo que es necesario un filtrado para que tan solo aparezca una etiqueta para cada requisito. Posteriormente al filtrado se realiza una revisión manual auxiliar que garantiza la fidelidad del etiquetado. NFR-SO contiene 17434 requisitos no funcionales, distribuidos en subclases según la Tabla 3.1

Quality characteristic	**#Sentences**	**Proportion**
Availability	1301	7,46 %
Performance	6794	38,97 %
Maintainability	295	1,69 %
Portability	1393	7,99 %
Scalability	2206	12,66 %
Security	5211	29,89 %
Fault tolerance	234	1,34 %
Total	**17434**	**100 %**

Tabla 3.1: Distribución de NFR-SO. Tabla creada con los datos de [28].

– NFR-Review se publica en [32], y consta de 1278 requisitos, recolectados aleatoriamente de 4000 frases de reseñas de usuarios para dos aplicaciones bastante populares: *iBooks* para iOS y *Whatsapp* para Android (2000 reseñas de cada una). En un estudio previo de los autores de NFR-Review [35] se obtiene un dataset de 1259 requisitos, puesto que se excluyen 19 pertenecientes a la subcategoría de *Seguridad* al considerarse inframuestreados, pero en NFR-Review se consideran válidos los 1278, que finalmente son clasificados por tres anotadores diferentes en 5 categorías de requisitos no funcionales tal y como se muestra en la Tabla 3.2 . El objetivo original de NFR-Review es aportar un dataset para la clasificación en categorías de requisitos no funcionales, ya que la literatura acerca del tema se centra principalmente en la clasificación de funcionales.

Quality characteristic	**#Sentences**	**Proportion**
Reliability	587	45,9 %
Usability	432	33,7 %
Performance efficiency	121	9,5 %
Portability	119	9,3 %
Security	19	1,6 %
Total	**1278**	**100 %**

Tabla 3.2: Distribución de NFR-Review. Tabla extraída de [35].

La razón principal por la que se descartan ambos conjuntos de datos es porque se centran en la clasificación de subcategorías de requisitos no funcionales, en lugar de la

clasificación binaria *funcional / no funcional*. Particularmente en el caso de NFR-SO, el tamaño del *dataset* excede la capacidad de cálculo disponible para manejarlo en el contexto de la realización de este trabajo.

3.2.4. Otros *datasets*

Añadidos a los conjuntos de datos anteriores, se encuentran otros *datasets*, que por sus características son difíciles de encapsular en una categoría concreta o que no cuentan con un nombre propio por constituirse recolectando requisitos de varios proyectos diferentes e incluso mezclando otros conjuntos de datos. Es el caso de los siguientes:

- El *dataset* empleado en el experimento del artículo [29] construye su colección de datos juntando el ya mencionado PROMISE con una selección de requisitos de otros 7 proyectos industriales del modo que muestra la Tabla 3.3. Los requisitos se etiquetan según su aspecto funcional (F) o cualitativo (NF), con la singularidad de que se anota si un requisito puede poseer ambas propiedades (F+NF) o si no posee ninguna (R). El etiquetado es realizado por dos autores que deben determinar si una muestra es funcional o no, entrando en juego un tercer autor para desempatar en el caso de que no se llegue a un acuerdo.

  - *ESA Euclid*: es un proyecto de la *European Space Agency* que aporta requisitos de fiabilidad, seguridad, control de altitud y órbita, propulsión, telemetría, seguimiento y mando.
  - *Helpdesk*: contiene requisitos de los usuarios para implantar un centro de asistencia.
  - *User mgmt*: contiene requisitos para una aplicación de solicitud y gestión de cuentas de usuario.
  - *Dronology*: contiene requisitos en el ámbito del desarrollo de sistemas aéreos no tripulados.
  - *ReqView*: detalla la especificación de requisitos para la herramienta de gestión de requisitos *ReqView*. Los autores modifican el formato de los requisitos, formando frases completas.
  - *Leeds Library*: se trata de un sistema de gestión de una biblioteca.
  - *WASP*: aporta los requisitos de un sistema de una aplicación de arquitecturas web (*Web Architectures for Services Platforms*).

Dataset	Public	Rows	F	NF	Only F	Only NF	F + NF	R
PROMISE	Yes	625	310	382	230	302	80	13
ESA Euclid	No	236	91	211	23	143	68	2
Helpdesk	No	172	143	51	121	29	22	0
User mgmt	No	138	126	25	113	12	13	0
Dronology	Yes	97	94	28	68	2	26	1
ReqView	Yes	87	75	32	54	11	21	1
Leeds Library	Yes	85	44	61	23	40	21	1
WASP	Yes	62	55	19	42	6	13	1
Totals		**1502**	**938**	**809**	**674**	**545**	**264**	**19**

Tabla 3.3: Distribución de requisitos por documento. Tabla extraída de [29].

– El *dataset* empleado en el estudio [33], que combina un conjunto de datos preexistente denominado *ReqEval*, con 98 requisitos, con uno creado específicamente para el desarrollo del experimento denominado *DAMIR*, con 1251. Si bien en el comienzo del trabajo se consideró útil puesto que contiene un buen número de requisitos recabados de 23 proyectos de *software* diferentes, el objetivo de *ReqEval* y *DAMIR* es la detección de anáforas en lugar de la clasificación, por lo que carece del etiquetado *funcional / no funcional*.

El *dataset* que unifica PROMISE con los otros 7 proyectos industriales resultó clave para decidir la dirección adoptada a la hora de construir el *dataset* final: recabar las muestras de diferentes documentos de especificación de requisitos. Sin embargo se descartó por contener pocos proyectos diferentes y por presentar la problemática ya mencionada de obtener los requisitos de repositorios privados en los casos de *ESA Euclid*, *Helpdesk* y *User mgmt*.

3.2.5. PURE

Continuando en la línea de explorar publicaciones relacionadas con el NLP, más allá de la clasificación se encuentra [34], que trata de resolver el problema de la detección de correferencias. En este *paper* se presenta TABASCO, una herramienta para detectar e identificar las ambigüedades presentes en los sustantivos empleados en los requisitos de software mediante el uso de BERT. Uno de los experimentos llevados a cabo para la demostración de la utilidad de la herramienta, consiste en detectar las ambigüedades existentes en un conjunto de requisitos obtenido al fusionar los documentos de requisitos de software multidisciplinares presentados en el *dataset* PURE (PUblic REquirements), presentado en [36].

PURE es un conjunto de datos orientado a permitir la replicación de experimentos de procesado natural de lenguaje y la generalización de resultados en ingeniería de

requisitos. Consta de 79 documentos de especificación de requisitos de software públicos y recabados de la web. Los documentos presentes en PURE [3] cubren múltiples dominios y además de estar disponibles para ser descargados, se encuentran descritos según sus características a distintos niveles:

- Estructura: según la forma en la que se expresan los requisitos dentro del documento, aparecen archivos *structured* (S) cuyos requisitos aparecen descritos de un modo estructurado, como en casos de uso; *unstructured* (U) cuyos requisitos aparecen en lenguaje natural; *one statement* (O) cuyos requisitos aparecen en una sola oración cada uno. Existen documentos que mezclan estructuras.

- Nivel: según el grado de abstracción que de los requisitos. Existen requisitos de alto nivel (H o *high-level*) si se orientan más a la comprensión cognitiva humana o de bajo nivel (L o *low-level*) si son especificaciones más técnicas.

- Fuente: según el origen de los documentos. Aparecen proyectos de índole universitaria (U) e industrial pública o privada (I).

Como nota negativa, la colección de datos de PURE incluye 34268 sentencias, una cantidad enorme que dificulta el procesado en el caso de que se quiera utilizar el *dataset* completo. Además, su utilidad para la categorización de requisitos aún no está empíricamente comprobada y no se ha realizado la anotación *funcional/no funcional*, por lo que para la realización de ese experimento es necesario más trabajo.

3.3. Construcción del conjunto de datos: DaReC

3.3.1. Justificación de la elección y estructura

La elección final como conjunto de datos de partida sobre el que se va a construir el *dataset* propio en este trabajo es PURE. Pese a las desventajas comentadas en la subsección anterior, se estima que los puntos positivos y las posibilidades de PURE superan las presentadas por el resto de literatura revisada:

- Frente a los *datasets* de dominio privado, sus requisitos son fácilmente accesibles y provienen de un repositorio público que cualquiera puede utilizar y revisar.

- Frente a PROMISE, presenta un tamaño mucho mayor, con lo que al aumentar el número de datos de entrenamiento del modelo de lenguaje, el experimento puede resultar más preciso.

[3] Sitio web para la consulta y descarga del *dataset*: `https://fmt.isti.cnr.it/nlreqdataset`

– Frente a NFR-SO y NFR-Review, contiene tanto requisitos funcionales como no funcionales. Característica indispensable si se quiere realizar una clasificación binaria entre estas dos opciones.

– Frente al dataset que combina PROMISE con otros 7 proyectos de software, dispone de una mayor variedad de proyectos con la riqueza multidisciplinar que ello conlleva.

– Frente a ReqEval y DAMIR, resulta más fácil de localizar y presenta unas estadísticas que pueden resultar útiles durante la construcción (estructura, nivel y fuente).

Aunque en su concepción inicial PURE no estaba pensado para labores de clasificación, contiene una gran cantidad de ejemplos en lenguaje natural. Concretamente estos ejemplos son requisitos, por lo que si se logran etiquetar correctamente se puede llegar a obtener un nuevo dataset de referencia para la categorización en ingeniería de requisitos. Por otro lado, al contar con 34268 requisitos, PURE resulta ser un conjunto de datos demasiado grande. A nivel máquina no se dispone de la capacidad de cálculo para entrenar un modelo de lenguaje con tanto *input*. A nivel humano, el preprocesamiento manual necesario para la extracción de todos los requisitos, todas las descripciones y la categorización *funcional/no funcional* excede por mucho el tiempo correspondiente a la realización de un solo trabajo de fin de grado. Además, incluso si se dispusiera de más recursos, los modelos de lenguaje seleccionados no admiten un entrenamiento con tal cantidad de parámetros de entrada. Debido a todo esto, se opta por truncar el número de documentos que se inspeccionan a 50 y a anotar tan solo los requisitos que vienen descritos exactamente como tal.

La estructura objetivo del conjunto de datos es contener todos los requisitos estimados válidos de los 50 documentos revisados, con un identificador individual, indicando el proyecto del que proviene cada uno y etiquetándolo según si es o no funcional. Debe haber una descripción de la problemática que trata de resolver cada uno de los proyectos de software de cuyas documentaciones se extraen los requisitos que finalmente aparecerán en el *dataset*. Adicionalmente se debe realizar una categorización por temas según la finalidad de cada proyecto. El conjunto de datos obtenido finalmente se denomina DaReC (*DAtaset for REquirements Classification*).

3.3.2. Metodología empleada

Para la obtención de DaReC a partir de la modificación de PURE, se va a tratar de seguir, en la medida de lo posible, la forma de preparación de los datos propuesta

en [18]:

1. Pre-procesado: se filtran los *tokens* ruidosos, como por ejemplo las etiquetas HTML o las expresiones SQL mediante el uso de expresiones regulares. Se eliminan expresiones de plantilla como por ejemplo «el código debería...» o «me gustaría que...». Durante la construcción del conjunto de datos no ha sido necesario ningún cambio en los requisitos de PURE debido a que los anotadores de los documentos de requisitos recogidos en PURE ya debieron realizar este proceso, una de las ventajas de partir de un *dataset* existente.

2. Ground-truth labeling: los requisitos, por su carácter técnico, acostumbran a ser difíciles de etiquetar. Para asegurar la precisión del etiquetado esta labor la suelen realizar dos o más anotadores, de tal modo que solo se incluyen en el conjunto de datos final aquellas muestras que logran unanimidad o mayoría en su categorización. En PURE aparecen fundamentalmente requisitos ya etiquetados como funcionales o no, pero en el caso de que no exista dicha diferenciación, solo se dispone de un autor para dirimir la naturaleza del requisito.

3. Balanceo: es importante conocer y comprender las características del *dataset* que se está construyendo, pero no existe una métrica que demuestre que un conjunto de datos con muestras equiprobables de decisión para la clasificación obtenga mejores resultados. En el caso académico [18], se inframuestrea una de las dos posibilidades en la categorización para lograr la equiprobabilidad, pero para la modificación de PURE se opta por no llevar a cabo este procedimiento. De esta manera, se obtienen un número de requisitos funcionales mucho mayor que de no funcionales, pero este escenario es muy similar a la realidad en la anotación de requisitos de software.

Teniendo clara esta guía de buenas prácticas, se procede a recabar los datos existentes en los documentos de PURE y a organizarlos en un archivo Excel. Para la extracción de los requisitos se escoge respetar lo máximo posible la definición dada por la documentación: se introducen modificaciones solo para el caso de explicación de siglas y acrónimos o por la necesidad de contextualizar el inicio y el final de la frase que contiene el requisito, especialmente en los casos que no son *one-statement*. Para la extracción de las descripciones del problema sí que es necesaria una depuración mayor, ya que se tratan de párrafos completos en lenguaje natural, que deben ser autoexplicativos y a partir de los cuales deben poder deducirse los requisitos relativos al proyecto. La estrategia general adoptada, aunque dependiendo del documento puede cambiar debido a la variedad de archivos y formatos, consiste

en: abrir el fichero pertinente; leer el apartado *description* y *purpose* que suelen contener toda la información necesaria para la comprensión del problema; redacción de la descripción; búsqueda de requisitos; redacción de los requisitos; categorización *funcional/no funcional*. En la Tabla 3.4 y la Tabla 3.5 pueden observarse ejemplos del formato y contenido de requisitos y descripciones en el conjunto de datos final.

Con respecto al cronograma adoptado para la elaboración de DaReC. En primer lugar se realizó el estudio de conjuntos de datos ya existentes que pudiesen servir de ayuda y la elección de PURE. En segundo lugar, se realizó una clasificación por temas de los proyectos presentados en los documentos de PURE para el filtrado y selección de los 50 documentos que aparecerían en el conjunto de datos final. En tercer lugar, se realiza la primera labor de descarte de documentos por los siguientes motivos: similitud con otros proyectos, preferencia por documentos cuyos requisitos estuvieran enunciados como *one-statement* o *structured*, sobremuestreo de algún tema, aparición de documentos muy dependientes del contexto (siglas, organizaciones no enunciadas dentro del proyecto, etc.) que además solían ser los más extensos. En cuarto lugar, se selecciona la herramienta en la que se van a recopilar los requisitos y descripciones (Excel), y se precisa el formato y la información que debe contener cada uno. En quinto lugar, se realiza la extracción como tal de requisitos y descripciones, redactando las descripciones y copiando y pegando en la medida de lo posible los requisitos para tratar de modificarlos lo mínimo posible. En sexto lugar, al no haberse excluido aún suficientes documentos para tener tan solo 50, se llevó a cabo una última fase de descarte en la que se desecharon proyectos por su dificultad o extensión. En séptimo y último lugar, ya se dispone del *dataset* DaReC finalizado y preparado para el entrenamiento del modelo de lenguaje, por lo que se realiza un análisis estadístico de las características del mismo.

Referencia	**Requisito**	**Documento**	**F / NF**
OTR-39	The system shall enable user to select the shipping method during payment process.	2007 - e-store.pdf	Funcional
GES-391	The information browsers shall provide tabular information identifying locations with associated data.	2007 - mdot.pdf	Funcional
INT-172	The system shall be implemented in the Java 1.3 programming language	2001 - hats.pdf	No funcional

Tabla 3.4: Ejemplo de requisitos en el *dataset* final.

Documento	Descripción
2004 - ijis.doc	The Integrated Justice Information System (IJIS) is an initiative being undertaken by Tarrant County, Texas. The vision for IJIS is to define and develop enhanced business processes and supporting technology solutions that result in a more effective and efficient administration of justice in Tarrant County. When complete, IJIS will facilitate rapid sharing of information across the Criminal Justice Community while providing each stakeholder with advanced justice management capabilities. The system must submit requests for appointed counsel, process requests for appointed counsel, manage notifications, manage reporting, manage notice of case filing decisions, manage defendant contact, manage attorney list and manage defense attorney compensation
2001 - spaces fractions.pdf	The Space Fractions project is a learning tool created to help improve fraction-solving skills for sixth-grade students. The product will be a web-based, interactive game. At the end of the game, students will be given feedback based on their game scores. We are also providing an umbrella for the past games created. The umbrella will be a web-based menu system allowing the user to choose between the games.

Tabla 3.5: Ejemplo de descripciones en el *dataset* final.

3.4. Resultado

DaReC contiene 2391 requisitos, 2022 de ellos funcionales y 369 no funcionales, recogidos de los distintos documentos de PURE tal y como se observa en los Anexos A. Con él se van a realizar los experimentos de clasificación empleando BERT, RoBERTa y DeBERTa.

Los temas en los que se han dividido los 50 proyectos según su finalidad son 6: interfaz de usuario, juegos, monitorización, control, gestión y otros (aquellos que no corresponden a ninguno de los otros campos). La distribución de los requisitos y del número de documentos por tema se puede observar en la Figura 3.1 y la Figura 3.2 respectivamente. Los documentos menos abundantes son los correspondientes a interfaz de usuario y control, seguidos por aquellos catalogados como de monitorización y juegos, por lo que en una línea futura podría ser conveniente tratar de aumentar su representación en el caso de querer mejorar el *dataset*. Los documentos más abundantes son los de gestión y los denominados «otros», dada la variedad de proyectos que se presentan en DaReC. La distribución de los requisitos por tema va estrechamente

ligada a la cantidad de documentos.

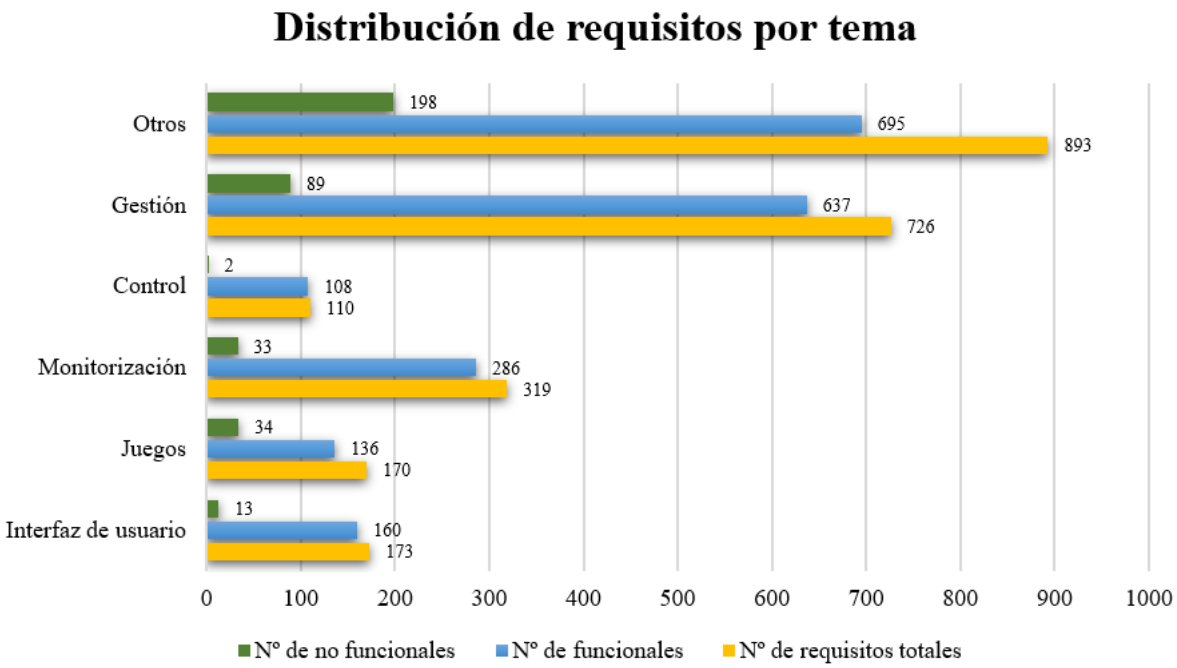

Figura 3.1: Distribución de DaReC. Requisitos funcionales, no funcionales y totales divididos según el tema.

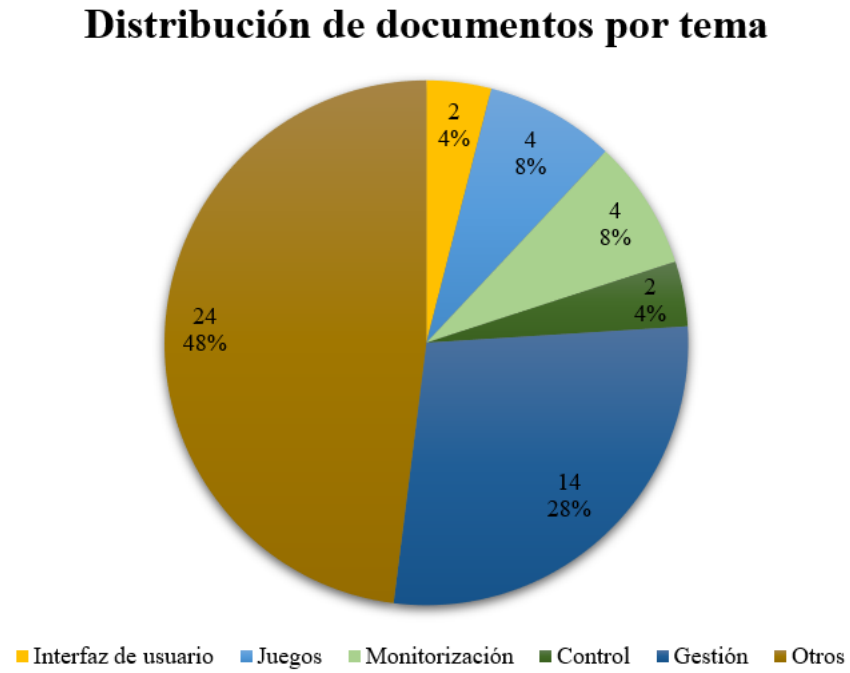

Figura 3.2: Distribución de DaReC. Proporción de documentos pertenecientes a cada uno de los temas.

El resultado final es un Excel de tres hojas, una para el listado de requisitos, otra para el listado de descripciones y la tercera para el análisis, representación gráfica y

otros cálculos realizados con los datos. En la primera hoja, además de enunciar el requisito, se asigna un identificador para el mismo, se etiqueta con el documento del que procede y se señaliza si es funcional o no funcional, como se observa en la Tabla 3.4. En la hoja de descripciones, aparece el nombre de cada documento y su explicación, como se observa en la Tabla 3.5, además de una columna de comentarios. En la última aparecen tablas y gráficas para analizar las características del *dataset*.

Para finalizar el análisis de DaReC, se ha calculado el número promedio de palabras con las que cuenta cada requisito, junto con la desviación estándar. Asimismo se ha calculado estos datos para las descripciones. Los resultados se pueden observar en la Tabla 3.6 y las Figuras 3.3 y 3.4.

	Longitud media	**Desviación estándar**
Requisitos totales	22,38	16,22
Requisitos funcionales	22,91	16,65
Requisitos no funcionales	19,49	13,25
Descripciones	224,10	127,14

Tabla 3.6: Media y desviación estándar de la longitud en palabras de requisitos y descripciones.

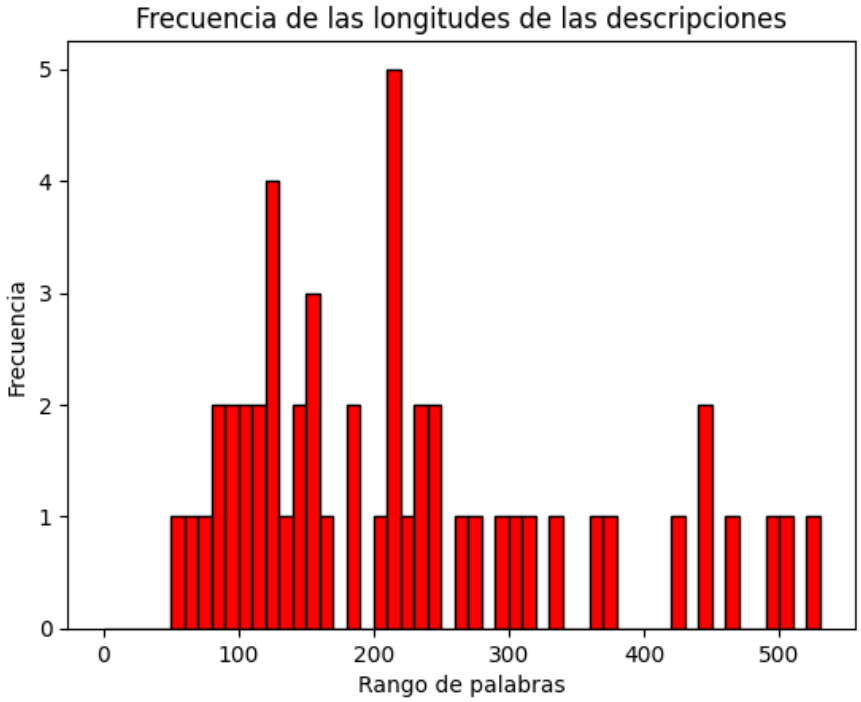

Figura 3.3: Histograma descriptivo de las longitudes de las descripciones en palabras

Las descripciones son mucho más largas que los requisitos como cabría esperar. Los requisitos funcionales parecen ser algo más largos que los no funcionales, pero la elevada desviación estándar sugiere una variabilidad significativa, lo que podría comprometer

la fiabilidad de los datos. Determinar si una longitud de 22 palabras es adecuada para la redacción de un requisito depende del contexto y del dominio del problema. Podrían ser extensos en algunos casos y breves en otros. El alto valor de las desviaciones típicas refleja diferencias en cómo se redactan los requisitos y descripciones, lo cual es esperable al recabar los datos de proyectos en dominios de aplicación muy distintos entre sí y definidos por equipos normalmente heterogéneos.

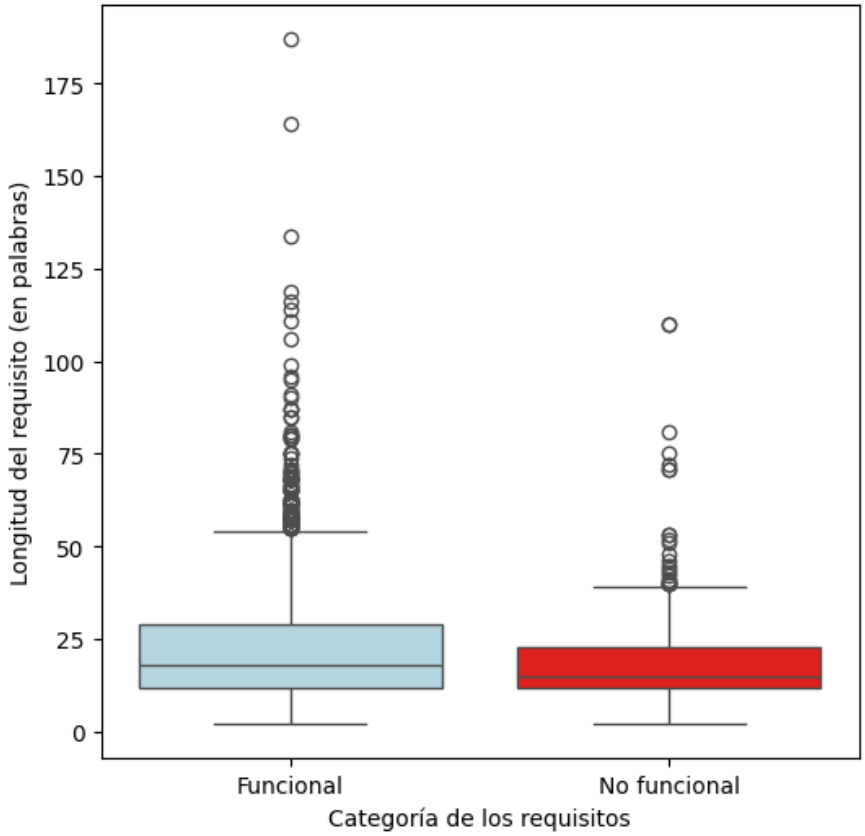

Figura 3.4: Comparativa de la longitud en palabras de requisitos funcionales y no funcionales

Capítulo 4

Experimentación

4.1. Descripción general

Una vez definidos en el Capítulo 2 los modelos de lenguaje y en el Capítulo 3 el conjunto de datos con el que se entrenan dichos modelos (DaReC), se procede a definir las pruebas que se van a realizar para medir la precisión a la hora de clasificar los requisitos en funcionales y no funcionales.

En líneas generales, se va a tratar de aprovechar toda la información que aporta el *dataset*, con respecto a su división por temas y documentos, para realizar distintos experimentos fragmentando los datos de entrada de diversas maneras especificadas en la Sección 4.3. El objetivo es dirimir cuál de las distintas opciones de entrenamiento arroja mejores resultados. Además, cada una de las pruebas se va a replicar con cada uno de los modelos para observar de este modo cuál es más adecuado para clasificar el conjunto de datos.

Todas las pruebas consisten en particionar de algún modo el *dataset* en tres: datos de entrenamiento, datos de validación y datos de test. Los datos de entrenamiento sirven para entrenar el modelo, los de validación para evaluar el rendimiento del modelo durante el entrenamiento y realizar reajustes y los de test para extraer resultados y conclusiones. Las conclusiones consistirán en afirmar o refutar que cada modelo en cada experimento es capaz de realizar una buena clasificación de requisitos en funcionales y no funcionales. Para llegar a estas conclusiones se debe estudiar el comportamiento de los siguientes parámetros que se obtienen a la salida de cada experimento:

- Precisión: mide qué proporción de las predicciones positivas hechas por el modelo son realmente correctas. Es especialmente útil en escenarios donde es importante evitar falsos positivos. Se calcula siguiendo la Fórmula 4.1

$$\text{Precisión} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Positivos}} \tag{4.1}$$

- *Recall*: mide qué proporción de los casos positivos reales fueron identificados

correctamente por el modelo. Es útil en escenarios donde es importante capturar todos los casos positivos, incluso si se generan algunos falsos positivos. Se calcula siguiendo la Fórmula 4.2

$$\text{Recall} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Negativos}} \tag{4.2}$$

– *F1-Score*: es la media armónica de la precisión y el *recall*. Es una métrica que busca un balance entre ambas, especialmente útil cuando los datos están desbalanceados y se necesita tanto alta precisión como alto *recall*. Se calcula siguiendo la Fórmula 4.3

$$\text{F1-Score} = 2 \cdot \frac{\text{Precisión} \cdot \text{Recall}}{\text{Precisión} + \text{Recall}} \tag{4.3}$$

Las tres métricas son evaluadas tanto para requisitos clasificados como funcionales como para requisitos clasificados como no funcionales. Esta es la estrategia seguida en [30] para evaluar si el modelo es capaz de categorizar los requisitos correctamente.

4.2. Versiones de los modelos de lenguaje

Si bien ya se ha justificado la elección de los modelos, al llegar a la fase de experimentación es primordial seleccionar también con qué versión se van a efectuar las pruebas. Por ejemplo, no es lo mismo tratar de traducir dos líneas de texto del francés al español con GPT-2 (1500 millones de parámetros) que con GPT-4 (100 billones de parámetros): ambos son capaces de conseguirlo, pero si se trata de entrenar todo el modelo para realizar esta tarea, se necesita mucha más capacidad de computación para el segundo caso. Para seleccionar un modelo de lenguaje adecuado se estudia el número de parámetros, el vocabulario de tokens y otras características inherentes a cada modelo.

Siguiendo este enfoque, se ha estimado que para la clasificación binaria *funcional / no funcional* de DaReC, que cuenta con 2391 requisitos, es suficiente con las versiones base de los modelos BERT, RoBERTa y DeBERTa. Todas las versiones se tratan de modelos preentrenados en inglés, idioma en el que se encuentran escritos los requisitos, lo cual ahorra tokens a la hora de realizar el entrenamiento. Estos modelos están disponibles para el público general en Hugging Face [37], la plataforma colaborativa para la comunidad de *machine-learning*.

4.2.1. Versión de BERT

En el caso de BERT, el modelo empleado es *bert-base-cased*[1]. Tiene la particularidad de distinguir mayúsculas y minúsculas a la hora de tokenizar palabras. Dado que en los requisitos aparecen palabras en mayúscula y minúscula además de un número importante de siglas, se prefiere la versión *cased* a la *uncased*. Tiene las siguientes características:

- Número de parámetros: 108 millones.
- Tamaño del vocabulario de tokens: 29 mil.
- Número de tokens que permite a la entrada el modelo: 512.
- Número de capas: 12.
- Dimensión de los *embeddings*: 768.

4.2.2. Versión de RoBERTa

El modelo empleado es *roberta-base*[2]. Tiene las siguientes características:

- Número de parámetros: 125 millones.
- Tamaño del vocabulario de tokens: 50 mil.
- Número de tokens que permite a la entrada el modelo: 514.
- Número de capas: 12.
- Dimensión de los *embeddings*: 768.

4.2.3. Versión de DeBERTa

En el caso de DeBERTa, el modelo empleado es *deberta-v3-base*[3]. Se trata de la versión «pequeña» de DeBERTa más reciente. DeBERTaV3 [38] reemplaza el mecanismo MLM por *Replaced Token Detection* (RTD), una tarea de preentrenamiento más eficiente y que genera mejores resultados. Tiene las siguientes características:

- Número de parámetros: 184 millones.

[1]Para la consulta y descarga de BERT *base cased*: `https://huggingface.co/google-bert/bert-base-cased`

[2]Para la consulta y descarga de RoBERTa *base*: `https://huggingface.co/FacebookAI/roberta-base`

[3]Para la consulta y descarga de DeBERTa base: `https://huggingface.co/microsoft/deberta-v3-base`

– Tamaño del vocabulario de tokens: 128 mil.

– Número de tokens que permite a la entrada el modelo: 512.

– Número de capas: 12.

– Dimensión de los *embeddings*: 768.

4.3. Experimentos

Todos los experimentos se han realizado en Python, ya que gracias a su ecosistema de bibliotecas diseñadas para el aprendizaje automático y el NLP, se simplifica en gran medida la tarea. Principalmente estas librerías son:

– *datasets*: facilita el acceso, preprocesamiento y manejo de grandes conjuntos de datos para entrenamiento de modelos de lenguaje.

– *transformers*: proporciona los modelos de lenguaje, además de simplificar su uso y personalización.

– *evaluate*: permite calcular métricas de rendimiento

– *accelerate*: facilita el entrenamiento de modelos proporcionando una interfaz sencilla para distribuir tareas y optimizar el rendimiento, gestionando automáticamente la paralelización de procesos.

– *sklearn*: ofrece herramientas de clasificación, regresión y clustering, entre otras, para aprendizaje automático. Se utilizará para hacer validación cruzada[4].

En cuanto al entorno donde se han ejecutado las pruebas, se han utilizado Google Colab y Kaggle. Durante el desarrollo del código se empleó Google Colab ya que presentaba la ventaja de poder acceder fácilmente al *dataset* y otros archivos de entrada al poderse conectar directamente a Google Drive. Del mismo modo resultó muy útil para organizar los *outputs* generados. Para la ejecución final se empleó Kaggle, un entorno de ejecución en línea con más potencia de computación que Google Colab, lo cual aceleró la obtención de resultados.

A la hora de entrenar los modelos es necesario seleccionar el valor de ciertos parámetros que pueden influir en la respuesta final. La estrategia adoptada es fijar dichos parámetros para todas las ejecuciones, dado que hay bastantes pruebas diferentes

[4]La validación cruzada o cross-validation es una técnica empleada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y prueba

y de este modo se pueden comparar los resultados. Se replica en muchos aspectos el escenario planteado en [30]: *weight-decay* = 0,01 (valor para prevenir el sobreajuste que ayuda a mantener los pesos más pequeños para mejorar la generalización en datos nuevos); *epochs* = 10 (número de veces que el modelo pasa por todo el conjunto de datos de entrenamiento durante el aprendizaje); *batch-size* = 32 (número de muestras que se procesan simultáneamente antes de actualizar los pesos durante el entrenamiento). Se prueban para cada experimento dos valores de *learning-rate* (2e-5 y 5e-5) para comprobar si existe alguna mejora. El *learning-rate* es un valor que define la magnitud de los ajustes que se hacen a los pesos del modelo en cada paso de la optimización, si es muy alto el modelo puede no converger y si es muy bajo el entrenamiento será lento.

La estructura general del código para todas las pruebas es la siguiente:

1. Descarga o importación de las bibliotecas y paquetes necesarios para la ejecución.
2. Preparación del dataset: se trata de un *.csv*, por lo que es necesario hacer un pre-procesamiento para codificar las columnas en un formato adecuado para su uso.
3. Descarga del modelo de lenguaje y tokenización de los datos a clasificar.
4. Partición de los datos en entrenamiento, validación y test.
5. Declaración del entrenador con todos los parámetros inicializados a los valores que se han estimado oportunos.
6. Entrenamiento con los datos de entrenamiento y validación. Durante el proceso de validación se escoge el *learning-rate* que proporciona mejores resultados.
7. Con el modelo ya entrenado y el *learning-rate* seleccionado, realizar predicción con los datos de test que se habían reservado.
8. Cálculo de precisión, *recall* y *f1-score*.

4.3.1. Experimento 1: *Stratified 10-fold*

El primero de los experimentos consiste en realizar una partición de los datos estratificada para realizar una validación cruzada de 10 iteraciones. Así todos los datos del *dataset* aparecen tanto en el conjunto de entrenamiento como en el de test, con lo que se puede comprobar si los resultados obtenidos varían mucho según la partición con la que se lleva a cabo el aprendizaje. Otro motivo por el que se realiza *cross-validation* es que es metodológicamente incorrecto entrenar el modelo con los datos que se van a utilizar para predecir resultados una vez ha concluido el aprendizaje. Un modelo

entrenado de esta manera repetiría las etiquetas ya vistas y tendría una precisión perfecta, pero no sería útil para realizar predicciones sobre un conjunto de datos nuevo [39, 40].

Hacer una validación cruzada de 10 iteraciones significa que los datos van a ser divididos en 10 subconjuntos, de los cuales 9 van a ser utilizados para entrenamiento y el restante para test. Dentro de los datos de entrenamiento, se vuelve a realizar la misma división para obtener el subconjunto con el que se va a realizar el entrenamiento real y el subconjunto de validación para observar cómo va el aprendizaje durante la ejecución y ajustar el *learning-rate*. Que la división sea estratificada asegura que la distribución de las clases (*funcional / no funcional*) se mantenga en el set de datos de entrenamiento y el de test.

En resumen, se está dividiendo el *dataset* en trozos de unos 240 requisitos, teniendo alrededor de 2100 para entrenar y validar el modelo y alrededor de 240 para tratar de clasificar en funcional y no funcional y medir los resultados obtenidos. En ambas particiones la proporción *funcional / no funcional* se mantiene similar a la que se observa en el *dataset* original. Se habla de valores aproximados porque la división se realiza de manera automática y para que se cumplan las especificaciones en cada ejecución puede variar.

Para corroborar la validez de las predicciones y comprobar la independencia del funcionamiento de los modelos con respecto a los datos de entrenamiento, se calculan la precisión, recall y f1-score tomando como datos de test cada una de las 10 particiones y almacenando para cada partición el valor de *learning-rate* más beneficioso. Este experimento se replica con BERT, RoBERTa y DeBERTa para poder comparar el desempeño de los distintos modelos.

4.3.2. Experimento 2: *10-fold* dividiendo por documentos

El siguiente experimento replica la estrategia del experimento descrito en la Subsección 4.3.1 con algunas modificaciones.

En este caso se busca medir si la precisión, *recall* y *f1-score* obtenidos mejoran con respecto al experimento anterior al entrenar el modelo con un conjunto de requisitos que pertenezcan a un conjunto de documentos y se dejan para test los requisitos que pertenecen al conjunto de documentos restante. Que la partición sea *10-fold* implica que se dividen en 10 los 50 documentos, utilizando 1 partición de 5 para la predicción y las otras 9 particiones de 5 para entrenamiento.

La escisión del conjunto de datos de test no puede ser estratificada, ya que se está dividiendo por nombres de documentos, no según si los requisitos contenidos son funcionales o no funcionales. Sin embargo, la división en datos de entrenamiento y

validación sí que se realiza de forma estratificada para que el aprendizaje sea lo más completo posible.

De nuevo se corrobora la validez de los resultados y se comprueba la independencia del funcionamiento de los modelos con respecto a los datos de entrenamiento tomando como datos de test cada una de las 10 particiones y replicando el experimento para los tres modelos.

4.3.3. Experimento 3: *6-fold* dividiendo por temas

Este experimento es prácticamente igual que el descrito en la Subsección 4.3.2.

Se divide el *dataset* en datos de test y de entrenamiento. Este caso tiene la particularidad de que no puede dividirse en 10 dado que el número de temas disponibles es de tan solo 6. Por ello, a diferencia de los experimentos ya vistos, las particiones siempre serán iguales: interfaz de usuario, juegos, monitorización, control, gestión y otros.

Se calcula el *learning-rate* que produce mejores resultados en validación y con este parámetro fijado, se mide la precisión, *recall*, *f1-score* tomando como conjunto de test cada una de las 6 particiones en temas. Se replica para los tres modelos de lenguaje.

4.3.4. Experimento 4: Añadiendo las descripciones

Como último experimento que se va a realizar utilizando los requisitos recabados del *dataset* de PURE, se replica el experimento de la Subsección 4.3.1, pero añadiendo a cada requisito la descripción del proyecto del que subyace. El fundamento en el que se basa este experimento es que los modelos basados en BERT son capaces de reconocer el contexto de las oraciones. Por lo tanto, es posible que exista una mejora en la clasificación si se introduce información del proyecto en el que se desarrolla cada uno de los requisitos.

La división sigue siendo de *10-fold* estratificada y las métricas que se estudian siguen siendo mejor *learning-rate*, precisión, *recall* y *f1-score*. Se calculan los resultados para los tres modelos y tomando como conjunto de datos de test cada una de las 10 particiones.

La peculiaridad que aparece en este experimento es que es necesario introducir en el modelo dos frases, relacionadas pero diferentes. Para ello se utiliza el token especial SEP cuyo funcionamiento ya se ha detallado en la Subsección 2.3.1 y con el que se construye una estructura de entrada al modelo como la que se observa en la Figura 4.1. La longitud de la estructura en algunos casos puede exceder el límite del modelo, por lo que es necesario truncar el número de *tokens* al máximo que se admita para poder

realizar el entrenamiento.

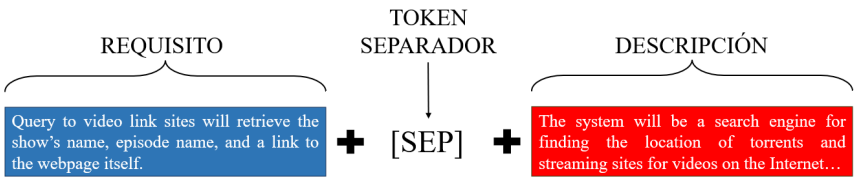

Figura 4.1: Ejemplo de estructura *Requisito + SEP + Enunciado*

4.3.5. Experimento 5: Utilizando PROMISE

Como experimento final, se busca comparar el desempeño del *dataset* recabado para el proyecto con el ampliamente extendido PROMISE. Se elige PROMISE porque es el utilizado en el artículo [30], estudio del que se extrae el diseño de las pruebas de este proyecto.

Las pruebas consisten en reproducir el experimento que realiza una partición en 10 estratificada de todos los requisitos y una partición en 10 según el documento de origen, descritos en las Subsecciones 4.3.1 y 4.3.2 respectivamente.

PROMISE, al igual que DaReC, identifica los documentos de los que obtiene sus requisitos, sin embargo no clasifica por temas. Por ello puede realizarse la réplica del segundo experimento, pero no la del tercero. Tampoco se puede replicar la prueba en la que se añade la descripción del problema dado que el *dataset* PROMISE no especifica ningúna descripción de los proyectos de los que recaba sus requisitos.

Como particularidad, la versión de PROMISE utilizada para la clasificación de requisitos en [30], diferencia distintas subclases de requisitos no funcionales. En este trabajo la clasificación buscada es binaria *funcional / no funcional*, así que es necesario un preprocesamiento previo al inicio del entrenamiento de los modelos de lenguaje que transforme las subclases en la etiqueta «No funcional».

Se emplean BERT, RoBERTa y DeBERTa y las métricas estudiadas son las mismas que en el resto de pruebas: mejor *learning-rate*, precisión, *recall* y *f1-score*.

Capítulo 5

Resultados

La estrategia adopatada a la hora de evaluar los resultados obtenidos consiste en:

- Analizar los valores de precisión, *recall* y *f1-score* para cada uno de los experimentos realizados.
- Comprobar si dentro de cada uno de los experimentos existe algún modelo que proporcione una respuesta superior al resto.
- Comprobar si se obtienen mejores resultados para alguno de los dos valores de *learning-rate* con los que se replican cada uno de los experimentos.
- Cotejar los valores obtenidos en experimentos distintos para examinar si alguna de las formas de entrenamiento aventaja a las demás.

Para realizar la comparación entre los valores de *learning-rate*, se calcula en cuántos casos se obtienen mejores resultados para cada uno. Se toma como mejor aquella cifra que provoque mejores resultados un número mayor de veces.

Para realizar la comparación entre modelos dentro de un mismo experimento y la comparación entre experimentos, se realiza una prueba T de Student o *T-Test*. Este análisis permite determinar si la diferencia en las medias entre dos grupos es estadísticamente significativa. Previamente se debe comprobar que los datos siguen una distribución normal[1] y que la diferencia de varianzas es nula[2].

Es importante conocer que el *T-Test* solo permite realizar la comparativa entre pares, es decir, no se pueden comparar los tres modelos a la vez. Por ello la metodología a seguir es seleccionar aquella métrica con un resultado superior y realizar un *T-Test* con los otros dos modelos en dos ejecuciones diferentes.

[1]Para comprobar la distribución normal de cada una de las métricas a comparar, se realiza una prueba de Shapiro-Wilk.

[2]Para comprobar que la diferencia de varianzas es cero se realiza una prueba de Levene

5.1. Resultados del Experimento 1: *Stratified 10-fold*

Con respecto al primer experimento que se realiza, las métricas de precisión, *recall* y *f1-score* toman los valores especificados en las Tablas 5.1, 5.2 y 5.3 respectivamente.

Modelos	**Precisión 'F'**		**Precisión 'NF'**	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT	0,9569	0,0131	0,7817	0,0944
RoBERTa	0,9516	0,0152	0,7999	0,0774
DeBERTa	0,9571	0,0172	0,7784	0,0625

Tabla 5.1: Valores obtenidos de precisión para el Experimento 1.

Modelos	***Recall* 'F'**		***Recall* 'NF'**	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT	0,9575	0,0271	0,7611	0,0792
RoBERTa	0,9644	0,0195	0,7285	0,0942
DeBERTa	0,9580	0,0187	0,7616	0,1009

Tabla 5.2: Valores obtenidos de *recall* para el Experimento 1.

Modelos	***F1-Score* 'F'**		***F1-Score* 'NF'**	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT	0,9569	0,0128	0,7648	0,0538
RoBERTa	0,9577	0,0086	0,7560	0,0545
DeBERTa	0,9572	0,0049	0,7625	0,0342

Tabla 5.3: Valores obtenidos de *f1-score* para el Experimento 1.

Los resultados son muy similares para los tres modelos. Los valores de precisión indican que alrededor del 95 % de requisitos catalogados por los modelos como funcionales son realmente funcionales y alrededor del 78 % de los catalogados como no funcionales son realmente no funcionales. Los valores de *recall* indican que alrededor del 95 % de requisitos funcionales han sido correctamente clasificados como funcionales por los modelos y en torno al 75 % de no funcionales han sido correctamente clasificados como tal. Si se tiene en cuenta el *f1-score* como métrica para valorar la actuación de BERT, RoBERTa y DeBERTa a la hora de clasificar, la tasa de acierto es similar a la obtenida en precisión y *recall*.

La respuesta obtenida en el caso de los requisitos funcionales es algo mejor que para el caso de los no funcionales. Esto se debe a que la cantidad de muestras en el entrenamiento categorizadas como «F» excede con creces la cantidad de «NF».

El *T-Test* revela que pese a las ligeras diferencias, ningún modelo es superior al resto en ninguna de las métricas medidas.

En cuanto a la elección del *learning-rate* más favorable:

- En BERT, un *learning-rate* de 2e-5 es mejor en 4 ocasiones, frente a las 6 de 5e-5.
- En RoBERTa, un *learning-rate* de 2e-5 es mejor en 7 ocasiones, frente a las 3 de 5e-5.
- En DeBERTa, un *learning-rate* de 2e-5 es mejor en 7 ocasiones, frente a las 3 de 5e-5.

5.2. Resultados del Experimento 2: *10-fold* por documentos

Con respecto al segundo experimento que se realiza, las métricas de precisión, *recall* y *f1-score* toman los valores especificados en las Tablas 5.4, 5.5 y 5.6 respectivamente.

Modelos	**Precisión 'F'**		**Precisión 'NF'**	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT	0,9188	0,0403	0,6389	0,1353
RoBERTa	0,9357	0,0261	0,6857	0,1732
DeBERTa	0,9436	0,0413	0,6624	0,1566

Tabla 5.4: Valores obtenidos de precisión para el Experimento 2.

Modelos	***Recall* 'F'**		***Recall* 'NF'**	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT	0,9594	0,0163	0,5955	0,1254
RoBERTa	0,9424	0,0362	0,6420	0,1234
DeBERTa	0,9292	0,0522	0,7214	0,1429

Tabla 5.5: Valores obtenidos de *recall* para el Experimento 2.

En este caso se observa que las diferencias en las métricas entre requisitos funcionales y no funcionales son muy grandes. Esto se debe a que al dividir el *dataset* por documentos, la probabilidad de que durante el aprendizaje el modelo no haya tenido apenas ejemplos de requisitos no funcionales es grande, al existir documentos que no

Modelos	*F1-Score* 'F'		*F1-Score* 'NF'	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT	0,9230	0,0333	0,6071	0,1026
RoBERTa	0,9384	0,0211	0,6441	0,1058
DeBERTa	0,9352	0,0350	0,6729	0,1127

Tabla 5.6: Valores obtenidos de *f1-score* para el Experimento 2.

contienen ninguna muestra de esta categoría. Si se toman como datos de entrenamiento documentos que no incluyen requisitos no funcionales, a la hora de realizar la clasificación se obtienen métricas muy buenas para precisión, *recall* y *f1-score* de «F» y muy malas para precisión, *recall* y *f1-score* de «NF».

Por otra parte y avalado por la prueba del *T-Test*, se puede observar que los 3 modelos arrojan resultados muy similares, sin que ninguno suponga una mejora sobre los demás. Los indicadores de precisión, *recall* y *f1-score* toman valores de entre el 91 % y el 96 % dependiendo de la métrica para requisitos funcionales, mientras que apenas alcanzan valores de entre el 60 % y el 72 % para no funcionales. Además, la categoría no funcional presenta una gran desviación estándar.

En cuanto a la elección del *learning-rate* más favorable:

- En BERT, un *learning-rate* de 2e-5 es mejor en 2 ocasiones, frente a las 8 de 5e-5.
- En RoBERTa, un *learning-rate* de 2e-5 es mejor en 4 ocasiones, frente a las 6 de 5e-5.
- En DeBERTa, un *learning-rate* de 2e-5 es mejor en 6 ocasiones, frente a las 4 de 5e-5.

5.3. Resultados del Experimento 3: *6-fold* por temas

Con respecto al tercer experimento que se realiza, las métricas de precisión, *recall* y *f1-score* toman los valores especificados en las Tablas 5.7, 5.8 y 5.9 respectivamente.

Los resultados de este experimento son análogos a los descritos en la Sección 5.2. La división por temas, del mismo modo que la división por documentos, provoca un submuestreo de requisitos no funcionales en la partición de datos de entrenamiento. Esto a su vez provoca que aunque los valores obtenidos de precisión, *recall* y *f1-score* para la clasificación funcional ronden el 95 %, las métricas para la clasificación no funcional apenas alcancen el 57 % o el 67 % dependiendo del indicador que se observe.

Nuevamente la desviación estándar es muy grande para los requisitos no funcionales y suficientemente pequeña para los funcionales.

Modelos	**Precisión 'F'**		**Precisión 'NF'**	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT	0,9548	0,0289	0,5394	0,1709
RoBERTa	0,9540	0,0430	0,5777	0,2297
DeBERTa	0,9553	0,0396	0,5697	0,2127

Tabla 5.7: Valores obtenidos de precisión para el Experimento 3.

Modelos	***Recall* 'F'**		***Recall* 'NF'**	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT	0,9291	0,0362	0,6550	0,1225
RoBERTa	0,9394	0,0278	0,6734	0,1600
DeBERTa	0,9404	0,0218	0,6804	0,1362

Tabla 5.8: Valores obtenidos de *recall* para el Experimento 3.

Modelos	***F1-Score* 'F'**		***F1-Score* 'NF'**	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT	0,9413	0,0270	0,5765	0,1400
RoBERTa	0,9456	0,0178	0,5854	0,1795
DeBERTa	0,9471	0,0191	0,5945	0,1756

Tabla 5.9: Valores obtenidos de *f1-score* para el Experimento 3.

En cuanto a la elección del *learning-rate* más favorable:

- En BERT, un *learning-rate* de 2e-5 es mejor en 2 ocasiones, frente a las 4 de 5e-5.
- En RoBERTa, el número de ocasiones en las que 2e-5 arroja mejores resultados es el mismo que el número de ocasiones en las que 5e-5 arroja mejores resultados.
- En DeBERTa, un *learning-rate* de 2e-5 es mejor en 4 ocasiones, frente a las 2 de 5e-5.

5.4. Resultados del Experimento 4: Añadiendo las descripciones

Con respecto al cuarto experimento que se realiza, las métricas de precisión, *recall* y *f1-score* toman los valores especificados en las Tablas 5.10, 5.11 y 5.12 respectivamente.

Al añadir las descripciones, se vuelve a efectuar un *T-Test* que compare la actuación de BERT, RoBERTa y DeBERTa, pero ninguno de los tres modelos presenta una respuesta mejor que los demás. Los valores de precisión, *recall* y *f1-score* para la clasificación funcional se encuentran en torno al 96 % de acierto, mientras que los valores para la clasificación no funcional no superan el 80 % en el mejor de los casos.

Modelos	**Precisión 'F'**		**Precisión 'NF'**	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT	0,9589	0,0163	0,7831	0,0560
RoBERTa	0,9595	0,0149	0,8043	0,0587
DeBERTa	0,9599	0,0150	0,7923	0,0709

Tabla 5.10: Valores obtenidos de precisión para el Experimento 4.

Modelos	***Recall* 'F'**		***Recall* 'NF'**	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT	0,9594	0,0163	0,7723	0,0970
RoBERTa	0,9639	0,0148	0,7754	0,0885
DeBERTa	0,9609	0,0195	0,7779	0,0885

Tabla 5.11: Valores obtenidos de *recall* para el Experimento 4.

Modelos	***F1-Score* 'F'**		***F1-Score* 'NF'**	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT	0,9590	0,0074	0,7723	0,0515
RoBERTa	0,9615	0,0065	0,7840	0,0420
DeBERTa	0,9602	0,0101	0,7801	0,0562

Tabla 5.12: Valores obtenidos de *f1-score* para el Experimento 4.

En cuanto a la elección del *learning-rate* más favorable:

- En BERT, un *learning-rate* de 2e-5 es mejor en 8 ocasiones, frente a las 2 de 5e-5.
- En RoBERTa, un *learning-rate* de 2e-5 es mejor en 8 ocasiones, frente a las 2 de 5e-5.
- En DeBERTa, el número de ocasiones en las que 2e-5 arroja mejores resultados es el mismo que el número de ocasiones en las que 5e-5 arroja mejores resultados.

5.5. Resultados del Experimento 5: Utilizando PROMISE

Con respecto al último experimento que se realiza, las métricas de precisión, *recall* y *f1-score* toman los valores especificados en las Tablas 5.13, 5.14 y 5.15 respectivamente. Se diferencia entre el caso que hace una división estratificada del *dataset* de PROMISE y el caso que hace una división por documentos en cada una de las tablas.

Modelos	**Precisión 'F'**		**Precisión 'NF'**	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT$_{10fold}$	0,8960	0,0502	0,9192	0,0374
RoBERTa$_{10fold}$	0,8868	0,0530	0,9382	0,0304
DeBERTa$_{10fold}$	0,9129	0,0597	0,9266	0,0489
BERT$_{doc}$	0,7688	0,2736	0,7514	0,1640
RoBERTa$_{doc}$	0,7801	0,2746	0,8290	0,1098
DeBERTa$_{doc}$	0,7955	0,2746	0,8496	0,1098

Tabla 5.13: Valores obtenidos de precisión para el Experimento 5.

Modelos	***Recall* 'F'**		***Recall* 'NF'**	
	Media	**Desv. típica**	**Media**	**Desv. típica**
BERT$_{10fold}$	0,8791	0,0602	0,9270	0,0402
RoBERTa$_{10fold}$	0,9095	0,0466	0,9162	0,0475
DeBERTa$_{10fold}$	0,8863	0,0843	0,9378	0,0469
BERT$_{doc}$	0,8620	0,1542	0,5348	0,2722
RoBERTa$_{doc}$	0,9353	0,0915	0,5004	0,2926
DeBERTa$_{doc}$	0,6728	0,2926	0,8966	0,0915

Tabla 5.14: Valores obtenidos de *recall* para el Experimento 5.

En la prueba estratificada particionando PROMISE en 10, se obtienen métricas bastante aceptables en precisión, *recall* y *f1-score*. Tanto para requisitos funcionales como para no funcionales existe un acierto en la clasificación de cerca del 90 %, con desviaciones típicas pequeñas. Estos datos se asemejan bastante a los obtenidos en [30], lo cual sugiere que el experimento que se ha tratado de replicar se ha llevado a cabo correctamente.

Sin embargo, en la prueba que particiona PROMISE por documentos, ni BERT, ni RoBERTa, ni DeBERTa logran unos resultados consistentes. La elevadísima desviación típica de la precisión y el *f1-score* en la clasificación «F»constata que la clasificación siguiendo este método no es fiable. La diferencia entre los *recalls* de la clasificación

Modelos	*F1-Score* 'F'		*F1-Score* 'NF'	
	Media	**Desv. típica**	**Media**	**Desv. típica**
$\mathbf{BERT}_{10fold}$	0,8851	0,0330	0,9220	0,0227
$\mathbf{RoBERTa}_{10fold}$	0,8960	0,0273	0,9258	0,0220
$\mathbf{DeBERTa}_{10fold}$	0,8953	0,0453	0,9304	0,0275
$\mathbf{BERT}_{doc}$	0,6092	0,2541	0,7802	0,1093
$\mathbf{RoBERTa}_{doc}$	0,5893	0,2769	0,8641	0,0678
$\mathbf{DeBERTa}_{doc}$	0,7148	0,2769	0,8648	0,0678

Tabla 5.15: Valores obtenidos de *f1-score* para el Experimento 5.

funcional y no funcional indica que los modelos pueden identificar correctamente los casos positivos de una categoría pero tienen dificultades para identificar los de la otra. Los resultados obtenidos con BERT y RoBERTa sugieren que estos dos modelos son capaces de identificar correctamente los requisitos funcionales, pero no son buenos para identificar los no funcionales. Por el contrario, DeBERTa es capaz de identificar correctamente no funcionales y no realiza bien la identificación de funcionales.

Al realizar el *T-Test* se llega a la conclusión de que para los dos escenarios analizados, la diferencia en las medidas de precisión, *recall* y *f1-score*, dada la alta variabilidad de los datos, no es suficientemente significativa. Por lo tanto, ninguno de los tres modelos es considerado mejor que los demás.

En cuanto al *learning-rate* más favorable, no se aprecia una diferencia muy grande. Un valor de 2e-5 proporciona mejores resultados en 33 ocasiones, frente a las 27 en las que los mejores resultados se obtienen tomando un valor de 5e-5. Tampoco se aprecia ninguna diferencia entre el comportamiento de los tres modelos en este aspecto.

5.6. Comparaciones de experimentos

Dada su arquitectura similar, ya se ha comprobado que no hay ningún modelo de los estudiados que sobresalga por encima del resto. A continuación se explora si las diferencias que caracterizan a los experimentos realizados provocan resultados significativamente diferentes. Para ello se utiliza nuevamente la prueba T de Student. Como la comparación solo se puede realizar por pares, se contrastan las métricas obtenidas en dos experimentos diferentes para el mismo modelo de lenguaje. Por ejemplo para comparar el Experimento 1 con el Experimento 2, se examinan las medidas obtenidas al emplear BERT en el primer caso con las obtenidas al emplear BERT en el segundo caso.

– En primer lugar se analiza si los experimentos con menor acierto a la hora de

estimar precisión, *recall* y *f1-score* presentan diferencias entre ellos. Se llega a la conclusión de que el Experimento 2 y el Experimento 3 no presentan diferencias significativas, por lo que se pueden usar ambos indistintamente y los resultados serán similares.

- En segundo lugar se analiza si el Experimento 1 presenta alguna mejora sobre la partición en documentos. En el caso de DeBERTa y RoBERTa se observan mejores resultados para las *f1-scores*. En el caso de BERT la mejora es significativa para todas las métricas excepto para el *recall* de los requisitos no funcionales.

- En tercer lugar se compara el Experimento 1 con el Experimento 4, para determinar si añadir el enunciado del problema provoca una tasa de acierto de los modelos mayor. Aunque si se observan tan solo las medias, el Experimento 4 parece arrojar resultados ligeramente mejores, al realizar el *T-Test* se llega a la conclusión de que la variación no es estadísticamente significativa. Por lo tanto el Experimento 1 y el Experimento 4 tienen una respuesta similar para los tres modelos de lenguaje.

- Por último se comparan los resultados del Experimento 1 con el Experimento 5, para tratar de determinar si el uso de *datasets* diferentes provoca resultados diferentes. Al examinar las Tablas 5.1, 5.2, 5.3, 5.13, 5.14 y 5.15, se observa que el Experimento 1 alcanza métricas 7 puntos más altas para los requisitos funcionales, mientras que el Experimento 5 presenta mejores resultados para los no funcionales. Esta diferencia es significativa para BERT, RoBERTa y DeBERTa e indica que el *dataset* de PROMISE es más exacto en la categorización *no funcional*, debido a que está balanceado y el número de requisitos no funcionales es similar al de funcionales. Por otro lado, DaReC clasifica mejor requisitos funcionales. Por lo tanto, resultaría más adecuado uno u otro dependiendo de si es más crítica la exactitud de «F» o de «NF» en un escenario de clasificación de requisitos.

Las variaciones de *learning-rates* a lo largo de todo el proceso de experimentación no parecen muy determinantes a la hora de lograr mejoras, al menos con los valores probados. Si se estudian los resultados globalmente, de un total de 168 ejecuciones distribuidas entre cada uno de los experimentos para cada uno de los modelos, 93 veces se obtiene una mejor respuesta para 2e-5 y 75 veces para 5e-5. La conclusión es que para los casos vistos puede parecer mejor 2e-5, pero la diferencia es pequeña y depende de cada experimento y modelo de lenguaje, puesto que como ya se ha analizado, existen ocasiones donde 5e-5 resulta superior.

Capítulo 6

Conclusión y líneas futuras

6.1. Conclusión

En el presente trabajo de fin de grado se ha medido el alcance de los modelos de lenguaje actuales para realizar labores de clasificación de requisitos.

Entre los logros alcanzados figura la creación de DaReC, un nuevo conjunto de datos amplio y representativo de la realidad en el ámbito de los requisitos en Ingeniería de Software, donde los funcionales suelen superar en número a los no funcionales. Este *dataset* ha permitido la réplica de experimentos previos, además de la realización de nuevas pruebas para comprobar la efectividad de BERT, RoBERTa y DeBERTa en labores de clasificación de requisitos.

Durante la experimentación se ha llegado a la conclusión de que los tres modelos de lenguaje estudiados son capaces de realizar una actuación competitiva, con métricas de precisión, *recall* y *f1-score* adecuadas para la tarea. También se ha visto cómo la diferencia entre conjuntos de datos, entre modelos y entre parámetros de entrenamiento como el *learning-rate* pueden afectar a la consecución de respuestas por parte de los modelos más o menos acertadas. La información extra en materia de división por documentos, temas o descripciones de la problemática a la que aluden los requisitos ha permitido explorar estas diferencias en la calidad de la clasificación.

Además, la diversidad de orígenes, longitudes y temáticas de los requisitos permiten medir el desempeño de BERT, RoBERTa y DeBERTa en escenarios con alta variabilidad.

En resumen, a lo largo de este proyecto se han analizado los resultados obtenidos al entrenar los modelos de lenguaje BERT, RoBERTa y DeBERTa con el objetivo de realizar una clasificación binaria de requisitos, y se ha recabado un nuevo conjunto de datos para el entrenamiento que podría resultar útil a la comunidad.

6.2. Líneas futuras

Los desarrollos posteriores a este trabajo pueden ir enfocados en tres vías: la modificación del conjunto de datos de entrenamiento, el uso de otros modelos de lenguaje distintos y la modificación de los experimentos realizados.

Así pues, una forma de aumentar la exactitud de los modelos en la clasificación de requisitos podría ser aumentar el grueso de los mismos, documentos y temas que se recogen en DaReC. El incremento del volumen de datos podría significar:

- Mejoras en la precisión de las predicciones.
- Reducción de la dependencia entre resultados y datos.
- Mayor diversidad, que ayudaría a los modelos a generalizar sus resultados y mejorar su rendimiento en un escenario real.

Por otro lado sería interesante comprobar el funcionamiento de los experimentos utilizando modelos de lenguaje más grandes. Siguiendo la estrategia adoptada durante el proyecto, podrían utilizarse las versiones *large* de BERT, RoBERTa y DeBERTa. También se podrían comprobar las actuaciones de modelos que utilizan el *decoder* de la arquitectura *transformer*. Estos modelos (como Llama [41] o Mistral [42]) tienen billones de parámetros, por lo que la cantidad de cálculos necesarios para el entrenamiento es exponencialmente superior a los casos estudiados. Serían necesarios por lo tanto entornos de ejecución con una capacidad de computación mayor, que suelen ser de pago. Este problema se acentúa si se quisiera utilizar el modelo más famoso en la actualidad, GPT [9] y más concretamente sus versiones más recientes (GPT4 o GPT4o). Para entrenar los modelos basados en *decoder* sería necesario el uso de *adapters* [43], que permiten el entrenamiento de tan solo unos pocos parámetros situados entre las capas en lugar de modificar todo el modelo, lo cual reduce mucho los recursos requeridos.

Por último, podrían emplearse pesos que balanceasen el error proporcionado por los modelos a la hora de clasificar como funcional y no funcional. Dado que el contenido de DaReC es por definición desbalanceado para tratar de replicar el mundo real, esta práctica podría ayudar a equilibrar el entrenamiento a la hora de aprender a clasificar «F» o «NF».

Capítulo 7

Bibliografía

[1] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

[2] IEEE.org. Ieee-org news. `https://www.ieee.org/about/news/2023/news-release-2023-survey-results.html`, 2023. Accessed on 11/22/2024.

[3] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of software engineering*. Prentice-Hall, Inc., USA, 1991.

[4] ISO/IEC. 12207 - Information Technology / Software Life Cycle Processes, 1995.

[5] V. Terragni, P. Roop, and K. Blincoe. The Future of Software Engineering in an AI-Driven World. *arXiv preprint arXiv:2406.07737.*, 2024.

[6] A. Fox and D.A. Patterson. *Engineering Software as a Service: An Agile Approach Using Cloud Computing*. Strawberry Canyon LLC, 2013.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[10] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach, 2019.

[11] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. DeBERTa: Decoding-enhanced BERT with Disentangled Attention. *CoRR*, 2020.

[12] Amazon Web Services. ¿Qué son los modelos de lenguaje de gran tamaño (LLM)? `https://aws.amazon.com/es/what-is/large-language-model/`, 2024. Accessed on 10/10/2024.

[13] Ignacio López Montaner. Asistente virtual para soporte técnico basado en inteligencia artificial. Trabajo fin de grado, Universidad de Zaragoza, 2024.

[14] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI*, 2019. Accessed: 2024-11-15.

[15] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol. X, Y, Article 1*, 2024.

[16] Giriprasad Sridhara, Ranjani H. G., and Sourav Mazumdar. Chatgpt: A study on its utility for ubiquitous software engineering tasks, 2023.

[17] Hugo Larochelle, Dumitru Erhan, and Y. Bengio. Zero-data learning of new tasks. In *Proceedings of the National Conference on Artificial Intelligence*, volume 2, pages 646–651, 01 2008.

[18] Yawen Wang, Lin Shi, Mingyang Li, Qing Wang, and Yun Yang. A deep context-wise method for coreference detection in natural language requirements. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 180–191, 2020.

[19] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 01 2014.

[20] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[21] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[22] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*, 2013.

[23] Debora Nozza, Federico Bianchi, and Dirk Hovy. What the [MASK]? Making Sense of Language-Specific BERT Models. *CoRR*, abs/2003.02912, 2020.

[24] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

[25] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909, 2015.

[26] Joan C. Timoneda and Sebastian Vallejo Vera. Bert, roberta or deberta? comparing performance across transformer models in political science text. *The Journal of Politics*, 0(ja):null, 0.

[27] Seong-Min Gong. Text classification of cyberbullying comments: a study on the applicability of various BERT models. No Publicado, Junio 2023.

[28] Xianchang Luo, Yinxing Xue, Zhenchang Xing, and Jiamou Sun. PRCBERT: Prompt Learning for Requirement Classification using BERT-based Pretrained Language Models. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA, 2023. Association for Computing Machinery.

[29] Fabiano Dalpiaz, Davide Dell'Anna, Fatma Basak Aydemir, and Sercan Çevikol. Requirements classification with interpretable machine learning and dependency parsing. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 142–152, 2019.

[30] Tobias Hey, Jan Keim, Anne Koziolek, and Walter F. Tichy. NoRBERT: Transfer Learning for Requirements Classification. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 169–179, 2020.

[31] Márcia Lima, Victor Valle, Estevão Costa, Fylype Lira, and Bruno Gadelha. Software engineering repositories: Expanding the PROMISE database. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, SBES '19, page 427–436, New York, NY, USA, 2019. Association for Computing Machinery.

[32] Tianlu Wang, Peng Liang, and Mengmeng Lu. What aspects do non-functional requirements in app user reviews describe? An exploratory and comparative study. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 494–503, 2018.

[33] Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. Automated handling of anaphoric ambiguity in requirements: A multi-solution study. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pages 187–199, 2022.

[34] Ambarish Moharil and Arpit Sharma. TABASCO: A transformer based contextualization toolkit. *Science of Computer Programming*, 230:102994, 2023.

[35] Mengmeng Lu and Peng Liang. Automatic classification of non-functional requirements from augmented app user reviews. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE '17, page 344–353, New York, NY, USA, 2017. Association for Computing Machinery.

[36] Alessio Ferrari, Giorgio Oronzo Spagnolo, and Stefania Gnesi. PURE: A dataset of public requirements documents. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 502–505, 2017.

[37] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

[38] Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTaV3: Improving DeBERTa using ELECTRA-Style pre-training with gradient-disentangled embedding sharing, 2023.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[40] SciKit Learn. 3.1. cross-validation: evaluating estimator performance. `https://scikit-learn.org/1.5/modules/cross_validation.html`, 2024. Accesed on 19/11/2024.

[41] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[42] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[43] Clifton Poth, Hannah Sterz, Indraneil Paul, Sukannya Purkayastha, Leon Engländer, Timo Imhof, Ivan Vulić, Sebastian Ruder, Iryna Gurevych, and Jonas Pfeiffer. Adapters: A unified library for parameter-efficient and modular transfer learning, 2023.

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Información adicional del conjunto de datos

A.1. Desglose de los temas en los que se divide el *dataset*

Temas	Documentos	Nº de requisitos totales	Nº de requisitos funcionales	Nº de requisitos no funcionales
Interfaz de usuario	2	173	160	13
Juegos	4	170	136	34
Monitorización	4	319	286	33
Control	2	110	108	2
Gestión	14	726	637	89
Otros	24	893	695	198

A.2. Desglose de los documentos que componen el *dataset*

Nombre del documento	Nº de requisitos funcionales	Nº de requisitos no funcionales	Tema
2001 - beyond.pdf	29	1	Interfaz de usuario
2001 - hats.pdf	131	12	Interfaz de usuario
1999 - multi-mahjong.html	63	17	Juegos
2001 - space fractions.pdf	12	6	Juegos
2003 - qheadache.pdf	45	4	Juegos
2005 - triangle.pdf	16	7	Juegos
2003 - tachonet.pdf	18	6	Monitorización
2005 - clarus low.pdf	85	25	Monitorización
2005 - phin.pdf	115	0	Monitorización
2010 - mashboot.pdf	68	2	Monitorización

Nombre del documento	Nº de requisitos funcionales	Nº de requisitos no funcionales	Tema
2004 - rlcs.pdf	39	2	Control
2007 - water use.pdf	69	0	Control
0000 - inventory.pdf	24	7	Gestión
1998 - themas.pdf	36	0	Gestión
2001 - ctc network.pdf	79	7	Gestión
2001 - esa.pdf	21	1	Gestión
2002 - evla back.pdf	45	7	Gestión
2002 - evla corr.pdf	28	13	Gestión
2004 - sprat.pdf	54	0	Gestión
2005 - microcare.pdf	35	0	Gestión
2005 - pontis.pdf	122	13	Gestión
2007 - mdot.pdf	66	19	Gestión
2008 - keepass.pdf	26	5	Gestión
2008 - vub.pdf	31	12	Gestión
2009 - inventory 2.0.pdf	37	5	Gestión
2009 - model manager.pdf	33	0	Gestión
0000 - gamma j.pdf	14	26	Otros
2001 - libra.doc	12	10	Otros
2003 - agentmom.pdf	24	1	Otros
2003 - pnnl.pdf	26	9	Otros
2004 - colorcast.pdf	12	4	Otros
2004 - ijis.doc	26	7	Otros
2004 - phillips.doc	24	1	Otros
2005 - grid 3D.pdf	13	7	Otros
2005 - nenios.html	38	8	Otros
2005 - znix.pdf	8	8	Otros
2006 - stewards.pdf	14	17	Otros
2007 - eirene fun 7.pdf	94	1	Otros
2007 - e-store.pdf	73	22	Otros
2007 - puget sound.pdf	21	6	Otros
2008 - caiso.pdf	32	0	Otros
2008 - virtual ed.doc	27	12	Otros
2009 - email.pdf	51	9	Otros
2009 - gaia.pdf	25	8	Otros
2009 - library.pdf	33	4	Otros
2009 - peazip.pdf	11	2	Otros
2009 - video search.pdf	17	9	Otros
2009 - warc III.pdf	38	10	Otros
2010 - digital home 1.3.pdf	36	11	Otros
2010 - split merge.pdf	26	6	Otros