

Universidad Tecnológica de la Habana

“José Antonio Echeverría”

CUJAE

|  |
| --- |
| Filtrado de opiniones para soporte de software con conocimiento del dominio |

Informe de Prácticas Profesionales 2

Autor:

Ray Maestre Peña (raymaestrex@gmail.com)

Tutores:

Dr. C. Alfredo Simón Cuevas (asimon@ceis.cujae.edu.cu)

Dr. C. Orlando Gabriel Toledano López

La Habana, enero de 2025

Resumen

El éxito de un sistema de software se puede medir por cuán útil es para sus usuarios Por ello se hace necesario para los equipos de desarrollo conseguir una buena retroalimentación con sus usuarios que les indique sus intereses y necesidades. Con la popularización de plataformas como las redes sociales o las tiendas de aplicaciones han surgido muchas formas para obtener opiniones de usuarios respecto a un determinado software. Sin embargo, la cantidad es tal que su análisis manual se vuelve incosteable e incluso, la gran mayoría de estas opiniones no aportan información útil, son irrelevantes. Este trabajo propone el desarrollo de un algoritmo de filtrado y clasificación de opiniones de usuarios para un equipo de software, con el objetivo de filtrar aquellas opiniones que no son de utilidad y facilitar el posterior análisis de aquellas informativas dividiéndoles en distintas categorías. Esta solución está basada en el uso de modelos de lenguajes de arquitectura transformer y la incorporación de conocimiento de dominio a partir de un glosario de términos importantes en el desarrollo de software. Se evaluaron los modelos RoBERTa, BERTweet, ALBERT, XLNet y GPT-2 mediante experimentos utilizando una validación cruzada de 5 pliegues en 4 sets de datos que contienen miles de opiniones recopiladas sobre las aplicaciones Facebook, SwiftKey, Tap Fish y Temple Run 2 respectivamente. Los resultados obtenidos fueron favorables, destacándose las soluciones utilizando RoBERTa y BERTweet como las mejores entre las evaluadas y superando en la mayoría de escenarios al resto de soluciones del estado del arte.

**Palabras Clave:** opiniones de usuarios, clasificación de texto, modelos de lenguaje, conocimiento de dominio.

Abstract

The success of a software system can be measured by how useful it is to its users. This makes it necessary for development teams to get good feedback from their users to indicate their interests and needs. With the popularization of platforms such as social networks or app stores, many ways have emerged to get user reviews regarding a given software. However, the quantity is such that their manual analysis becomes unaffordable and even the vast majority of these reviews do not provide useful information, they are irrelevant. This work proposes the development of an algorithm for filtering and classifying user reviews for a software team, with the objective of filtering those reviews that are not useful and facilitating the subsequent analysis of informative reviews by dividing them into different categories. This solution is based on the use of transformer architecture language models and the incorporation of domain knowledge from a glossary of important terms in software development. The RoBERTa, BERTweet, ALBERT, XLNet and GPT-2 models were evaluated through experiments using a 5-fold cross-validation on 4 datasets containing thousands of reviews collected on the Facebook, SwiftKey, Tap Fish and Temple Run 2 applications respectively. The results obtained were favorable, highlighting the solutions using RoBERTa and BERTweet as the best among those evaluated and outperforming in most scenarios the rest of the state-of-the-art solutions.

**Keywords:** user reviews, text classification, language models, domain knowledge.

Índice

[Capítulo 1: Fundamentación teórica 6](#_Toc188009145)

[1.1 Ingeniería de requisitos basada en datos 6](#_Toc188009146)

[1.2 Clasificación de textos 7](#_Toc188009147)

[1.2.1 Pre-procesamiento 8](#_Toc188009148)

[1.2.2 Representaciones de texto 9](#_Toc188009149)

[1.2.3 Aprendizaje de máquina 10](#_Toc188009150)

[1.2.4 Enfoques tradicionales de solución 11](#_Toc188009151)

[1.3 Clasificación de textos con aprendizaje profundo 12](#_Toc188009152)

[1.3.1 Redes Neuronales 12](#_Toc188009153)

[1.3.1.1 Arquitecturas más utilizadas 15](#_Toc188009154)

[1.3.2 Representaciones distribuidas 16](#_Toc188009155)

[1.3.3 Transferencia de aprendizaje 17](#_Toc188009156)

[1.3.3.1 Fine-tuning de modelos Transformers para clasificación 17](#_Toc188009157)

[1.4 Incorporación de conocimiento del dominio 19](#_Toc188009158)

[1.4.1 Granularidad y fuentes de conocimiento 20](#_Toc188009159)

[1.4.2 Métodos de inyección de conocimiento 22](#_Toc188009160)

[1.4.2.1 Inyección directa de texto 22](#_Toc188009161)

[1.4.2.2 Vectores de características 23](#_Toc188009162)

[1.4.2.3 Modificación de *Embeddings* 24](#_Toc188009163)

[1.4.2.4 Recuperación de bases de datos 25](#_Toc188009164)

[1.4.2.5 Guiados por reglas 26](#_Toc188009165)

[1.4.2.6 Pre-entrenamiento adicional 27](#_Toc188009166)

[1.5 Antecedentes 28](#_Toc188009167)

[1.6 Métodos de evaluación 30](#_Toc188009168)

[1.6.1 0Métricas 30](#_Toc188009169)

[1.7 Herramientas utilizadas 32](#_Toc188009170)

[1.7.1 Lenguaje de programación: Python 32](#_Toc188009171)

[1.7.2 Bibliotecas principales 33](#_Toc188009172)

[1.8 Conclusiones parciales 36](#_Toc188009173)

[Capítulo 2: Método de solución planteado 38](#_Toc188009174)

[2.1 Descripción de la solución propuesta 38](#_Toc188009175)

[2.1.1 Descripción general 38](#_Toc188009176)

[2.1.2 Pre-procesamiento 39](#_Toc188009177)

[2.1.3 Construcción del vector de características 40](#_Toc188009178)

[2.1.3.1 Vector RC 40](#_Toc188009179)

[2.1.3.2 Vector RP 41](#_Toc188009180)

[2.1.4 Modelo de clasificación sin conocimiento 43](#_Toc188009181)

[2.1.5 Introducción de conocimiento al modelo. 44](#_Toc188009182)

[2.2 Desarrollo del clasificador de opiniones según su relevancia para el desarrollo de software 46](#_Toc188009183)

[2.2.1 Modelo de dominio 46](#_Toc188009184)

[2.2.2 Captura de requisitos 47](#_Toc188009185)

[2.2.2.1 Diagrama de casos de uso del sistema 47](#_Toc188009186)

[2.2.2.2 Descripción de los casos de uso 48](#_Toc188009187)

[2.3 Conclusiones parciales 50](#_Toc188009188)

[Capítulo 3: Validación y análisis de los resultados 51](#_Toc188009189)

[3.1 Marco de evaluación 51](#_Toc188009190)

[3.1.1 Modelos pre-entrenados seleccionados 52](#_Toc188009191)

[3.2 Descripción de los sets de datos 54](#_Toc188009192)

[3.3 Resultados experimentales 55](#_Toc188009193)

[3.3.1 Resultados experimentales sin uso de conocimiento del dominio 55](#_Toc188009194)

[3.3.2 Resultados experimentales utilizando conocimiento del dominio 58](#_Toc188009195)

[3.3.3 Comparación con otros resultados en la literatura 61](#_Toc188009196)

[3.4 Conclusiones parciales 63](#_Toc188009197)

[Conclusiones 64](#_Toc188009198)

[Recomendaciones 66](#_Toc188009199)

[Bibliografía 67](#_Toc188009200)

Índice de Figuras

[*Figura 1.1: Estructura general una red neuronal [*25*]* 13](#_Toc188009201)

[*Figura 1.2: Esquema del proceso de entrenamiento de una red neuronal [*31*]* 14](#_Toc188009202)

[*Figura 1.3: Estructura común de un modelo fine-tuned para una tarea de clasificación* 19](#_Toc188009203)

[*Figura 1.4: Ejemplo de una aplicación de la inyección directa de texto* 23](#_Toc188009204)

[*Figura 1.5: Integración de un vector de características en un MPT* 24](#_Toc188009205)

[*Figura 1.6: Integración del conocimiento en los embeddings del MPT* 25](#_Toc188009206)

[*Figura 1.7: Estructura general de un modelo integrado con una base de datos vectorial para recuperación* 26](#_Toc188009207)

[*Figura 1.8: Uso de reglas en MPT* 27](#_Toc188009208)

[*Figura 1.9: Esquema del pre-entrenamiento adicional en datos textuales del dominio* 28](#_Toc188009209)

[*Figura 2.1: Esquema del flujo de trabajo de la solución* 38](#_Toc188009210)

[*Figura 2.2: Ejemplo de vector RC* 41](#_Toc188009211)

[*Figura 2.3: Ejemplo de vector RP* 42](#_Toc188009212)

[*Figura 2.2: Arquitectura del modelo de clasificación sin conocimiento* 43](#_Toc188009213)

[*Figura 2.3: Arquitectura del modelo de clasificación con conocimiento* 45](#_Toc188009214)

[*Figura 2.4 Modelo de dominio de la solución* 46](#_Toc188009215)

[*Figura 2.4: Diagrama de casos de uso del sistema* 47](#_Toc188009216)

Índice de Tablas

[*Tabla 1.1 Matriz de confusión para una clasificación específica* 31](#_Toc188009217)

[*Tabla 1.2: Comparación de las principales bibliotecas para redes neuronales* 34](#_Toc188009218)

[*Tabla 2.1: Descripción del caso de uso clasificación de comentarios* 48](#_Toc188009219)

[*Tabla 2.2: Descripción del caso de uso preprocesar comentario* 49](#_Toc188009220)

[*Tabla 2.3: Descripción del caso de uso predecir relevancia* 49](#_Toc188009221)

[*Tabla 3.1: Composición de los sets de datos utilizados* 55](#_Toc188009222)

[*Tabla 3.2: Resultados sin conocimiento en el dataset de Facebook* 55](#_Toc188009223)

[*Tabla 3.3: Resultados sin conocimiento en el dataset de SwiftKey* 56](#_Toc188009224)

[*Tabla 3.4: Resultados sin conocimiento en el dataset de TapFish* 56](#_Toc188009225)

[*Tabla 3.5: Resultados sin conocimiento en el dataset de TempleRun2* 56](#_Toc188009226)

[*Tabla 3.6: Resultados generales sin conocimiento* 58](#_Toc188009227)

[*Tabla 3.7: Resultados comparativos con conocimiento de RoBERTa en Facebook* 59](#_Toc188009228)

[*Tabla 3.8: Resultados comparativos con conocimiento de BERTweet en Facebook* 59](#_Toc188009229)

[*Tabla 3.9: Resultados generales con conocimiento en Facebook* 61](#_Toc188009230)

[*Tabla 3.10: Comparación con otras soluciones de la literatura* 62](#_Toc188009231)

Introducción

El éxito de un sistema de software se puede medir por cuán bien cumple con su propósito, que se define a través de requisitos. Los requisitos mal definidos pueden llevar a un software que no satisface las demandas de los usuarios, resultando en un proyecto de software fallido [[1](#_ENREF_1)]. Por tanto, la Ingeniería de Requisitos juega un papel crucial en el éxito o fracaso de un proyecto de software.

El proceso de obtención de requisitos busca que las partes interesadas expresen sus necesidades de manera específica, guiándose muchas veces por la experiencia e intuición de los ingenieros de software. Este proceso sin embargo puede ser particularmente difícil de escalar con el aumento del alcance de los productos y un mercado cada vez más competitivo [[2](#_ENREF_2)].

Un problema reconocido en el proceso de extracción de requisitos es la falta de inclusión de los usuarios finales, lo que puede resultar en requisitos que no representan adecuadamente las necesidades de la mayoría de los usuarios. Esto puede dar lugar a requisitos incorrectos que eventualmente requieren una cantidad significativa de tiempo y dinero para corregir [[2](#_ENREF_2)]. Por lo tanto, hay una tendencia creciente hacia la participación activa de las partes interesadas, particularmente los usuarios, dicho enfoque se le conoce por algunos autores como Ingeniería de requisitos basada en datos (Data-Driven Requeriments Engeneering) [[3](#_ENREF_3)].

La Ingeniería de Requisitos basada en datos puede ayudar a identificar, priorizar y gestionar los requisitos que provienen de un gran grupo de usuarios [[4](#_ENREF_4)]. Esta opción permite la resolución de preguntas muy útiles para el desarrollo como: ¿Cuáles son los problemas que los usuarios y el software enfrentan? ¿De qué manera se utilizan realmente los sistemas de software desarrollados? ¿Qué nivel de satisfacción experimentan los usuarios? ¿Qué características desean que se mejoren o añadan?

Con la popularización de plataformas como las redes sociales o las tiendas de aplicaciones han surgido muchas fuentes para obtener dichas opiniones. Sin embargo, esto da lugar a nuevos problemas, la cantidad de opiniones es demasiado grande a medida que escala en cantidad de usuarios y se vuelve completamente imposible analizarlas todas a mano [[4](#_ENREF_4), [5](#_ENREF_5)]. Además, estudios como [[6](#_ENREF_6)] han demostrado que entre el 50% y el 70% de dichas opiniones no son relevantes para un equipo de desarrollo. Esto da lugar a una situación en la que una fuente de información importante es desperdiciada, conduciendo a características no óptimas y oportunidades perdidas.

Con los recientes avances en el campo de la inteligencia artificial, especialmente en el campo del procesamiento del lenguaje natural se han abierto nuevas posibilidades para solucionar este problema. Muchos autores en los últimos años como [[7-11](#_ENREF_7)] han enfrentado la clasificación de opiniones de usuarios mediante la utilización de algoritmos de aprendizaje automático para analizar estos grandes volúmenes de opiniones a base de poder computacional obteniendo resultados prometedores. De ellos, [[10-13](#_ENREF_10)] han hecho un enfoque especial en la detección de opiniones informativas para reducir considerablemente el número de opiniones a analizar mediante un filtro. Entre estos trabajos destacan los resultados de [[14-16](#_ENREF_14)] que emplean modelos de lenguaje de arquitectura Transformer mediante técnicas de ajuste fino.

Una práctica muy recomendada en tareas de análisis de datos es aprovechar información externa de dominio que pueda apoyar la técnica empleada [[17](#_ENREF_17), [18](#_ENREF_18)]. Sin embargo, este proceso de filtrado y clasificación de opiniones mediante el uso de modelos de lenguaje no ha sido probado en la literatura en conjunto con el empleo de conocimiento del dominio. El empleo de estas técnicas de soporte en conjunto con los modelos podría representar una mejora significativa a este proceso de detección de relevancia.

Ante las cuestiones analizadas se llega a la siguiente **situación** **problemática**:

* Las opiniones de los usuarios constituyen una fuente de información valiosa para los desarrolladores de software, pero su gran volumen hace incosteable su análisis manual completo.
* Es necesario un sistema que pueda procesar automáticamente las opiniones de los usuarios para lo cual primero hay que deshacerse de la gran cantidad de opiniones no informativas.
* Existe una posibilidad de mejora no explotada en la literatura con la incorporación de conocimiento del dominio a modelos de lenguaje de arquitectura Transformer.

Esta situación nos permitió identificar como **problema** **científico** en esta investigación:

¿Puede el uso la incorporación de conocimiento del dominio a modelos de lenguaje de arquitectura transformer mejorar la eficacia del proceso de filtrado y clasificación de opiniones de usuarios para el soporte de software?

El problema definido se encuentra enmarcado en el **objeto de estudio** de la ingeniería de requisitos basada en datos y tiene como **campo de acción** la clasificación de opiniones de usuarios.

Para dar respuesta al problema planteado se han definido los siguientes objetivos:

**Objetivo General:** Desarrollar un algoritmo de filtrado y clasificación de opiniones de usuarios para un equipo de desarrollo de software que incorpore el conocimiento del dominio a modelos de lenguaje de arquitectura Transformer.

Para lograr el cumplimiento del objetivo propuesto se determinaron los siguientes **objetivos específicos** y **tareas de investigación:**

1. Elaborar un marco teórico sobre la clasificación de texto y los diferentes enfoques utilizados en la literatura**.**

Tareas asociadas:

* 1. Investigar las técnicas de clasificación de texto utilizando aprendizaje profundo.
  2. Estudiar el funcionamiento de la arquitectura Transformer.
  3. Estudiar el proceso de ajuste fino de modelos de lenguaje y las formas de incorporarle conocimiento de dominio.
  4. Realizar un estudio de las soluciones anteriores presentes en la literatura sobre la clasificación de opiniones de usuarios.

1. Desarrollar un método de filtrado de opiniones según relevancia mediante el ajuste fino de modelos de lenguaje

Tareas asociadas:

* 1. Diseñar un método de filtrado de opiniones de usuarios para soporte de software utilizando modelos de lenguaje
  2. Establecer los parámetros y condiciones necesarias para el proceso de ajuste fino de los modelos de lenguaje.
  3. Incorporar el uso del conocimiento del dominio al procesamiento de los modelos de lenguaje.
  4. Implementar el método planteado.

1. Desarrollar un método de clasificación de opiniones mediante el ajuste fino de modelos de lenguaje

Tareas asociadas:

* 1. Diseñar un método de clasificación de opiniones de usuarios para soporte de software utilizando modelos de lenguaje.
  2. Establecer los parámetros y condiciones necesarias para el proceso de ajuste fino de los modelos de lenguaje.
  3. Incorporar el uso del conocimiento del dominio al procesamiento de los modelos de lenguaje.
  4. Implementar el método planteado.

1. Evaluar mediante experimentos los resultados de la solución propuesta

Tareas asociadas:

* 1. Seleccionar los sets de datos en los que se evaluará la solución.
  2. Seleccionar los modelos pre-entrenados de arquitectura Transformer con los que se evaluará la solución.
  3. Realizar experimentos con cada uno de los modelos en cada set de datos.
  4. Evaluar el impacto de la incorporación o no de conocimiento del dominio en la solución.
  5. Comparar los resultados con otras soluciones en la literatura.

El presente informe que contiene los resultados de la investigación se encuentra estructurado de la siguiente forma:

**Capítulo 1. Fundamentación teórica:** Contiene los elementos que conforman el marco teórico del objeto y campo de estudio correspondientes. Se presentan los principales aspectos de la ingeniería de requisitos que involucran la investigación. Se explican además los principales elementos de la clasificación de textos y diferentes enfoques utilizados en la literatura.

**Capítulo 2: Método de solución planteado:** Se fundamenta y describe el método propuesto para el filtrado y clasificación de opiniones, así como la forma en que se incorporó el conocimiento de dominio a este proceso. Se plantean además elementos metodológicos sobre la solución.

**Capítulo 3: Validación y análisis de los resultados:** Se exponen los resultados de evaluación de los modelos entrenados a partir de la realización de experimentos con un enfoque comparativo.

Finalmente se presentan las **Conclusiones**, se emiten las **Recomendaciones** derivadas de la investigación y se listan las **Referencias Bibliográficas** consultadas.

Capítulo 1: Fundamentación teórica

En este capítulo se abordarán las principales cuestiones teóricas referentes al problema de investigación y técnicas del estado del arte al respecto.

Primeramente, se introduce la ingeniería de requisitos basada en datos y las necesidades que aborda. Luego se presentan las bases de la clasificación de textos con procesamiento del lenguaje natural, incluyendo factores claves como el pre-procesamiento, la representación de los textos y los diferentes métodos de aprendizaje automático tradicional que se utilizan. Después se analizan las principales representaciones y arquitecturas utilizadas en el aprendizaje profundo, así como la transferencia de aprendizaje a través de modelos pre-entrenados. Finalmente se exponen las métricas y herramientas de desarrollo utilizadas en la solución propuesta.

# Ingeniería de requisitos basada en datos

Los requisitos de un sistema son las descripciones de los servicios que debe prestar un sistema y las limitaciones de su funcionamiento. Estos requisitos reflejan las necesidades de los clientes de un sistema que responde a un determinado propósito, como controlar un dispositivo, hacer un pedido o encontrar información. El proceso de averiguar, analizar, documentar y comprobar estos servicios y restricciones se denomina ingeniería de requisitos según [[19](#_ENREF_19)].

Es común que muchos softwares no tengan la aceptación o desempeño deseados debido a un mal manejo e identificación de los requisitos del software. Por ejemplo los autores de [[2](#_ENREF_2)]condujeron un estudio sobre los principales problemas en diversos aspectos de la ingeniería de requisitos sobre una muestra de 228 compañías de desarrollo de software. En el estudio anterior se destacaron 3 causas principales de problemas durante el desarrollo de software relacionadas con los requisitos:

* Requisitos incompletos y/o ocultos
* Cambio de objetivos, procesos de negocio y/o requisitos
* Requisitos no especificados que son demasiado abstractos

Estos problemas suelen estar relacionados con la dificultad que representa involucrar a los usuarios y sus necesidades en los procesos de ingeniería de requisitos [[5](#_ENREF_5)]**.** Además, esto es aún más notorio en entornos de desarrollo que cambian rápidamente, como por ejemplo en metodologías de desarrollo ágil de software, donde se van replanteando muchos aspectos del desarrollo a medida que este avanza [[20](#_ENREF_20)]**.**

La ingeniería de requisitos convencional suele implicar a los usuarios a través de entrevistas, talleres y grupos de discusión. Recientemente, los proveedores de software también han empezado a recoger las opiniones de los usuarios a través de canales de redes sociales, foros de usuarios y sistemas de revisión. En particular, con la aparición de las tiendas de aplicaciones como un mercado de software e infraestructura de despliegue, los usuarios pueden enviar fácilmente sus comentarios, revisar nuevas versiones, informar de errores, calificar aplicaciones y sus características, o solicitar nuevas características [[3](#_ENREF_3), [5](#_ENREF_5)]. El aprovechamiento de este tipo de datos por los desarrolladores para identificar, priorizar y gestionar los requerimientos de sus productos de software es conocido como ingeniería de requisitos basada en datos (data-driven requeriments engeneering) [[3](#_ENREF_3)].

# Clasificación de textos

La clasificación de textos es una tarea de importancia fundamental, que ha ido ganando terreno gracias a los recientes avances en los campos de la minería de textos y el procesamiento del lenguaje natural (PLN). Los métodos de clasificación de textos comparten el objetivo común de designar una etiqueta predefinida para un texto de entrada dado, aunque esta denominación puede referirse a una variedad de métodos especializados aplicados a distintos dominios. Ejemplos clásicos de clasificación de textos son el etiquetado de temas, el análisis de sentimientos, la clasificación de noticias y el filtrado de spam. [[21](#_ENREF_21)].

En general, el sistema de clasificación de textos contiene cuatro niveles diferentes de alcance que pueden aplicarse [[22](#_ENREF_22)]:

1. Nivel de documento: En el nivel de documento, el algoritmo obtiene las categorías relevantes de un documento completo.
2. Nivel de párrafo: En el nivel de párrafo, el algoritmo obtiene las categorías relevantes de un único párrafo (una porción de un documento).
3. Nivel de frase: En el nivel de oración, obtiene las categorías relevantes de una sola oración (una porción de un párrafo).
4. Nivel de subsentencia: En el nivel de subsentencia, el algoritmo obtiene las categorías relevantes de las subexpresiones dentro de una oración (una parte de una oración)).

## Pre-procesamiento

Antes de que un modelo procese texto para una tarea específica, a menudo es necesario preprocesar el texto para mejorar el rendimiento y desempeño del modelo. El pre-procesamiento consiste en transformar el texto antes de analizarlo, identificando las unidades (por ejemplo, palabras y frases) que se van a utilizar (tokenización), eliminando el contenido irrelevante para algunas tareas (por ejemplo, eliminar los caracteres no alfabéticos y las palabras vacías), aglomerar los términos relacionados semánticamente para reducir la dispersión de datos y aumentar la capacidad de predicción (conversión a minúsculas, corrección de faltas de ortografía, expansión de contracciones/abreviaturas y stem/lemmatización) y aumentar la cantidad de información semántica que se captura (tratamiento de la negación) [[23](#_ENREF_23), [24](#_ENREF_24)].

Debido al carácter ruidoso que tienen por naturaleza las opiniones de los usuarios este proceso cobra aún más importancia para hacerle una limpieza al texto que lo libre de cualquier información no necesaria que pueda perturbar al análisis. Esto fue comprobado de primera mano por el autor de [[9](#_ENREF_9)] quién realizó pruebas sobre su modelo después de eliminar el paso de pre-procesamiento y experimentó una caída en el desempeño en todas las métricas evaluadas. Sin embargo, esto significa que el pre-procesamiento también puede eliminar información útil (por ejemplo, suprimir las palabras vacías cuando son relevantes para la investigación), introducir errores en el análisis (por ejemplo, cuando el stemming confunde palabras semánticamente distintas) y alterar drásticamente los resultados posteriores [[23](#_ENREF_23)]. Por ello es necesario estudiar cada caso concreto y en función de ello evaluar qué técnicas de pre-procesamiento deberían ser utilizadas.

## Representaciones de texto

Un paso importante que requiere cualquier procedimiento de PLN es la proyección de las características del texto en un espacio de características elegido. Debido a su falta de estructura (desde el punto de vista computacional), es necesario aplicar una serie de operaciones para transformarlo gradualmente en una forma digerible para un ordenador. Esta forma es a través de modelos vectoriales conocidos como Vector Space Models (VSM) [[21](#_ENREF_21)]. En esta sección se tratarán algunas de las formas de representación más comunes.

**Bag of Word (BoW):** El texto es representado como una bolsa (colección) de palabras, ignorando el orden y el contexto. La intuición básica en que se basa es que supone que el texto perteneciente a una clase determinada en el conjunto de datos se caracteriza por un conjunto único de palabras. Si dos textos tienen casi las mismas palabras, pertenecen a la misma bolsa (clase) [[24](#_ENREF_24)].

**Bag of N-Grams(BoN):** funciona dividiendo el texto en trozos de n palabras contiguas (o tokens). Cada trozo se denomina n-grama. El vocabulario del corpus, V, no es más que una colección de todos los n-gramas únicos del corpus de texto. Cada documento del corpus está representado por un vector de longitud |V|. Este vector contiene simplemente los conteos de frecuencia de los n-gramas presentes en el documento y cero para los n-gramas que no están presentes [[25](#_ENREF_25)].

**TF-IDF:** Su objetivo es cuantificar la importancia de una palabra determinada en relación con otras palabras del documento y del corpus. Si una palabra w aparece muchas veces en un documento di, pero no aparece mucho en el resto de documentos dj del corpus, entonces la palabra w debe tener una gran importancia para el documento di. La importancia de w debe aumentar en proporción a su frecuencia en di, pero al mismo tiempo, su importancia debe disminuir en proporción a la frecuencia de la palabra en otros documentos dj del corpus [[24](#_ENREF_24)].

## Aprendizaje de máquina

El objetivo del aprendizaje de máquina o aprendizaje automático es "aprender" a realizar tareas basándose en ejemplos (llamados "datos de entrenamiento") sin instrucciones explícitas. Para ello, se crea una representación numérica (denominada "características") de los datos de entrenamiento y se utiliza esta representación para aprender los patrones de los ejemplos. [[26](#_ENREF_26)]. Los algoritmos de aprendizaje automático pueden agruparse en tres paradigmas principales: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

**Aprendizaje supervisado:** En esta categoría, se proporcionan a la máquina datos de muestra etiquetados para entrenarla, a partir de los cuales predecirá posteriormente los resultados. A continuación, se comprueba si los resultados son correctos y exactos con datos aleatorios. [[24](#_ENREF_24)]. El objetivo es ajustar un modelo que relacione la respuesta con los predictores, con el fin de predecir con exactitud la respuesta para futuras observaciones (predicción) o comprender mejor la relación entre la respuesta y los predictores (inferencia).[[26](#_ENREF_26)]

En otras palabras, consiste en la extracción de una muestra de una población, aprender de ella, y luego aplicar el modelo construido a nuevos ejemplos no etiquetados extraídos de la misma población. Tal como señalan [[27](#_ENREF_27)] este método requiere, sin duda, una gran cantidad de aplicación humana para construir el modelo, pero a la larga permite realizar más rápidamente una tarea que, de otro modo, sería tediosa.

**Aprendizaje no supervisado:** Permite a la máquina aprender sin supervisión. En el aprendizaje no supervisado, se proporciona a la máquina un conjunto de datos no segregados ni etiquetados, y se supone que el algoritmo actúa sobre los datos sin ningún tipo de supervisión. El objetivo de esta teoría es reagrupar los elementos de datos de entrada que presentan patrones similares. En este contexto, en cierto modo trabajamos a ciegas; la situación se denomina no supervisada porque carecemos de una variable de respuesta que pueda supervisar nuestro análisis [[26](#_ENREF_26)]. Este tipo de entrenamiento es explotado sobretodo en situaciones en las que es muy difícil conseguir un set de datos de entrenamiento etiquetado que permita un aprendizaje supervisado.

**Aprendizaje reforzado:** Trata de métodos para aprender tareas mediante ensayo y error y se caracteriza por la ausencia de datos etiquetados o no etiquetados en grandes cantidades. El aprendizaje se realiza en un entorno autónomo y mejora a través de la retroalimentación (recompensa o castigo) facilitada por el entorno [[24](#_ENREF_24)].

## Enfoques tradicionales de solución

Existen muchos métodos diferentes para abordar un problema de clasificación, en esta sección serán comentadas algunas de las más comunes.

**Heurísticas:** Este tipo de soluciones no conllevan ningún tipo de aprendizaje, sino que consisten en una serie de reglas que determinan el resultado del proceso. Se utilizan diccionarios de palabras que apuntan a ciertas clasificaciones generalmente y se evalúa la frecuencia y peso de las apariciones de cada posible clasificación del texto, este enfoque es comúnmente conocido como análisis de los lexicones [[24](#_ENREF_24)]. En los últimos años se ha utilizado mayormente de forma complementaria junto a algoritmos de aprendizaje de máquina como es el caso de [[28](#_ENREF_28)]

**Naive Bayes:** Se basa principalmente en el teorema de Bayes. Calcula la probabilidad de que el texto analizado pertenezca a una etiqueta de clase dado el vector de características de los datos de entrada. Una característica de este algoritmo es que asume que cada característica es independiente de todas las demás.[[24](#_ENREF_24)]. Este enfoque se puede ver en [[29](#_ENREF_29)]

**Support Vector Machine:** Cada punto del conjunto de datos puede considerarse un vector en un plano de N dimensiones, en el que cada dimensión representa una característica de los datos. Este algoritmo identifica los puntos de datos fronterizos (o puntos más cercanos a la clase opuesta), también conocidos como vectores de soporte, y luego intenta encontrar el límite (también conocido como el hiperplano en el espacio N-dimensional) que está más alejado del vector de soporte de cada clase [[7](#_ENREF_7), [25](#_ENREF_25)]. Método utilizado en [[27](#_ENREF_27)]

**K-nearest neighbours:** Este método trabaja encontrando las k instancias etiquetadas más similares y, en su iteración más sencilla, asignando la categoría más común a la instancia no etiquetada que se está clasificando. Este método no paramétrico puede ser bastante rápido, ya que sólo necesita calcular las distancias entre los puntos de datos. Sin embargo, su rendimiento depende en gran medida de la función de distancia elegida, y pueden ser necesarias diferentes funciones o incluso aproximaciones para tratar con grandes conjuntos de datos, en los que el rendimiento de estos métodos puede verse afectado [[21](#_ENREF_21)]. Ejemplo de uso en [[30](#_ENREF_30)]

# Clasificación de textos con aprendizaje profundo

El aprendizaje profundo (Deep Learning) es un campo específico dentro del aprendizaje automático: una nueva forma de aprender representaciones a partir de datos que hace hincapié en el aprendizaje de capas sucesivas de representaciones cada vez más significativas [[31](#_ENREF_31)].

## Redes Neuronales

La base de este tipo de aprendizaje son las redes neuronales, estructuras flexibles que pueden ser adaptadas a una gran variedad de contextos [[32](#_ENREF_32)]. Inspiradas en el principio del procesamiento de la información en los sistemas biológicos, las redes neuronales consisten en representaciones matemáticas de unidades de procesamiento conectadas llamadas neuronas artificiales. Al igual que las sinapsis en un cerebro, cada conexión entre neuronas transmite señales cuya fuerza puede amplificarse o atenuarse mediante un peso que se ajusta continuamente durante el proceso de aprendizaje [[33](#_ENREF_33)]. Una red neuronal se considera profunda generalmente cuando tiene más de 1 capa oculta. Conforme aumentan las capas de la red, su capacidad de abstracción aumenta y se pueden establecer relaciones más complejas y analizar datos de mayor dimensionalidad [[26](#_ENREF_26)]. Esto resulta de especial utilidad para el análisis del complejo lenguaje natural con nuevas posibilidades de representación.

Las redes neuronales están compuestas de forma general por 3 tipos de capas de neuronas: capas de entrada, capas ocultas y capas de salida.

Capa de entrada (Input layer): Esta es la primera capa en una red neuronal, y el número de nodos o neuronas en esta capa es igual al número de características que se alimentarían a la red.

Capa oculta (Hidden layer): Una red neuronal puede tener una o más capas ocultas. Son las capas intermedias de una red. En estas capas se derivan las relaciones y los patrones de los datos.

Capa de salida (Output layer): Es la última capa de una RNA que proporciona la salida para una entrada determinada. Recibe los resultados de las capas ocultas y los transmite. El número de nodos de la capa de salida depende del tipo de problema que se esté resolviendo. La capa de salida sólo tiene un nodo si se trata de un problema de clasificación binaria, ya que este nodo puede emitir 0 o 1 dependiendo de la clase en la que se clasifique el punto de datos. Para los problemas de clasificación multiclase, el número de nodos de la capa de salida es igual al número de clases, de modo que cada nodo representa la probabilidad de que el punto de datos pertenezca a una clase determinada [[25](#_ENREF_25)].

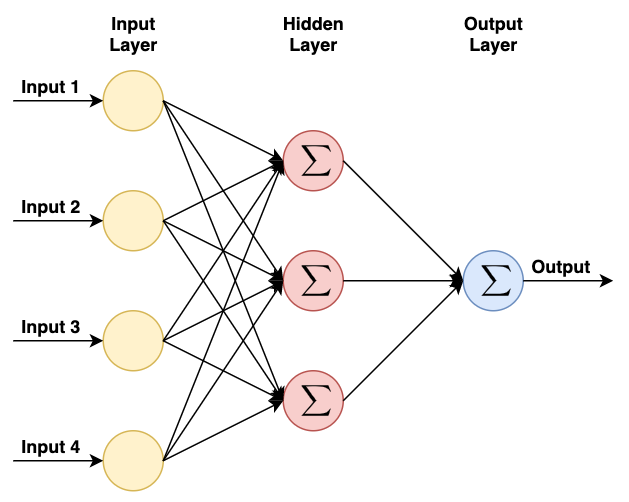


Figura 1.1: Estructura general una red neuronal [[25](#_ENREF_25)]

Con esta estructura la información es introducida a la red por la capa de entrada con un formato específico la cual es posteriormente procesada por las capas ocultas hasta que llega a la capa de salida que lo convierte en un resultado.

Este resultado predicho como es común en el aprendizaje automático se compara con el resultado esperado a través de una función de coste que puntúa que tan buena fue la predicción y penaliza al modelo por sus errores según el tipo o magnitud del error [[34](#_ENREF_34)]. Es fundamental en el aprendizaje profundo utilizar esta puntuación como señal de retroalimentación para ajustar un poco el valor de los pesos en una dirección que reduzca la puntuación de pérdida para el ejemplo actual. Este ajuste es tarea del optimizador, que implementa lo que se denomina algoritmo de retro-propagación (back-propagation). [[31](#_ENREF_31)].

Inicialmente, a los pesos de la red se les asignan valores aleatorios, por lo que la red se limita a aplicar una serie de transformaciones aleatorias. Naturalmente, su resultado dista mucho de lo que debería ser idealmente, y el valor de pérdida es, en consecuencia, muy alto. Pero con cada ejemplo que procesa la red, los pesos se ajustan un poco en la dirección correcta y la pérdida disminuye. [[31](#_ENREF_31)]. Este es el bucle de entrenamiento, que, repetido un número suficiente de veces produce valores de peso que minimizan la función de pérdida. Una buena representación gráfica de este proceso se puede apreciar en la figura 1.2

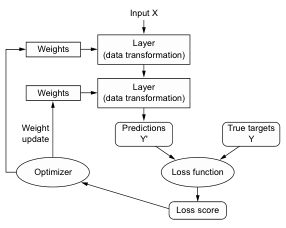


Figura 1.2: Esquema del proceso de entrenamiento de una red neuronal [[31](#_ENREF_31)]

### Arquitecturas más utilizadas

Con los años han surgido varias arquitecturas de redes neuronales con diferentes enfoques y capacidades. A continuación, se comentarán algunas de las arquitecturas más comunes en la literatura para la resolución de problemas orientados al procesamiento del lenguaje natural:

**Recurrent neural networks (RNN):** Las redes neuronales recurrentes están especialmente diseñadas para tener en cuenta el procesamiento secuencial y el aprendizaje. Las RNN tienen unidades neuronales capaces de recordar lo que han procesado hasta el momento. Esta memoria es temporal, y la información se almacena y actualiza con cada paso temporal a medida que la RNN lee la siguiente palabra de la entrada [[24](#_ENREF_24)]. A pesar de su capacidad y versatilidad, las RNN sufren el problema de la memoria olvidadiza: no pueden recordar contextos más largos y, por tanto, no funcionan bien cuando el texto de entrada es largo. Este problema fue abordado creando una variante conocida como **Long short-term memory(LSTM)** la cual sortea este problema dejando de lado el contexto irrelevante y recordando sólo la parte del contexto que se necesita para resolver la tarea en cuestión [[35](#_ENREF_35)]. Esta arquitectura fue abordada para el problema concreto de requerimientos de software en [[36](#_ENREF_36)].

**Convolutional neural networks (CNN):** Las redes neuronales convolucionales son usadas mayormente en visión por computadora, sin embargo, al tratar con representaciones vectoriales de palabras de igual tamaño es posible representar los textos como matrices de dimensión n x d, donde n es el número de palabras de la frase y d es el tamaño de los vectores de palabras [[24](#_ENREF_24)]. De esta forma se puede tratar al texto como si fuera una imagen. Son capaces de aprender una jerarquía de características y patrones que abstraen información espacial de dichas matrices, esto es de gran utilidad en tareas de clasificación [[37](#_ENREF_37)]. Las CNN se componen principalmente de capas convolucionales, seguidas de capas de activación y, a veces, de capas de agrupamiento (pooling), que permiten reducir la dimensionalidad de los datos mientras conservan las características más importantes [[22](#_ENREF_22)]. También se han desarrollado soluciones de clasificación de requerimientos con esta arquitectura [[9](#_ENREF_9)].

**Transformers:** arquitectura presentada en [[38](#_ENREF_38)] y que ha alcanzado el estado del arte en los últimos años en muchas tareas de PLN. Modelan el contexto textual, pero no de forma secuencial. Dada una palabra en la entrada, prefiere mirar todas las palabras a su alrededor (lo que se conoce como auto-atención) y representar cada palabra con respecto a su contexto [[31](#_ENREF_31)]. Este mecanismo de atención funciona calculando para cada palabra puntuaciones de relevancia respecto al resto de palabras en la oración. Se hará especial énfasis en esta arquitectura durante el trabajo pues es la arquitectura utilizada en la solución propuesta

## Representaciones distribuidas

Mientras que los métodos de representación vistos en el epígrafe 1.2.2 se centraban en capturar las representaciones sintácticas de las palabras y, en algunos casos, un pequeño subconjunto de las relaciones sintácticas que las unen en las frases, siguen careciendo de la capacidad de capturar su significado semántico. Un ejemplo clásico de este problema lo representan los sinónimos de palabras: aunque son semánticamente iguales, estos modelos no pueden captar su similitud. [[21](#_ENREF_21)].

Para resolver el problema anterior se desarrollaron los **words embeddings,** esta técnica de aprendizaje tiene como objetivo aprender una correspondencia entre cada fragmento de texto (normalmente palabras, de ahí su nombre) y un vector n-dimensional de números reales. Un embedding es, por tanto, una representación vectorial, digerible por una máquina, pero que también codifica parte del significado subyacente de las palabras [[24](#_ENREF_24)]. Estos enfoques se basan en redes neuronales, que aprenden estos mapeados mediante distintos procedimientos de aprendizaje; en general, se basan en el supuesto de que el significado de una palabra puede extraerse de las palabras que la rodean en una frase [[21](#_ENREF_21)].

Afortunadamente, en muchos casos no es necesario entrenar nuestros propios embeddings, y a menudo basta con utilizar words embeddings ya entrenados. Estas pueden verse como una gran colección de pares clave-valor, donde las claves son las palabras del vocabulario y los valores son los vectores de palabras correspondientes [[24](#_ENREF_24)]. Algunos de los embeddings pre-entrenados más utilizados en el estado del arte son Word2vec de Google [[39](#_ENREF_39)], GloVe de Stanford [[40](#_ENREF_40)], y fasttext embeddings de Facebook [[41](#_ENREF_41)].

## Transferencia de aprendizaje

Recientemente, se han utilizado grandes transformers para el aprendizaje por transferencia de tareas derivadas más pequeñas. El aprendizaje por transferencia es una técnica de IA en la que los conocimientos adquiridos al resolver un problema se aplican a otro distinto pero relacionado [[21](#_ENREF_21)]. Con los transformers, la idea es entrenar un modelo muy grande de forma no supervisada (lo que se conoce como pre-entrenamiento) para predecir una parte de una frase dado el resto del contenido, de modo que pueda codificar los matices de alto nivel del lenguaje en ella [[24](#_ENREF_24)]. Estos modelos se entrenan con enormes cantidades de datos textuales, extraídos de todo Internet.

Estos modelos pre-entrenados de lenguaje son utilizados como base para re-entrenarlos en tareas específicas como la clasificación o traducción o incluso para mejorar su precisión de predicción en un dominio específico de una situación particular [[31](#_ENREF_31)]. Este proceso en la literatura es conocido como “fine-tunning”. Entre los modelos más usados en la literatura para tareas de clasificación se encontraron: BERT [[14](#_ENREF_14)], roBERTa [[15](#_ENREF_15), [42](#_ENREF_42)], alBERT [[15](#_ENREF_15)], BERTweet [[42](#_ENREF_42)], entre otros.

### Fine-tuning de modelos Transformers para clasificación

Tal como se mencionó anteriormente los modelos pre-entrenados de lenguaje son entrenados con una gran cantidad de datos de texto sin etiquetas, como libros, artículos y sitios web. El objetivo es capturar los patrones, estructuras y conocimiento semántico subyacente en el corpus de texto. Este proceso es un aprendizaje no supervisado, donde los modelos aprenden de los datos de texto sin orientación explícita o etiquetas [[24](#_ENREF_24), [25](#_ENREF_25)]. Debido a esto uno de estos modelos no puede ser utilizado directamente para realizar una tarea de clasificación, primero deben sufrir algunas modificaciones en su estructura.

Primero que nada, debe ser adaptada la entrada del modelo al tamaño que se estime necesario para cada situación particular, pues este tamaño de entrada determinará la capacidad máxima de tokens que pueden ser procesadas por cada petición, esto en una tarea de clasificación de texto se corresponde generalmente a la cantidad de palabras (en algunas modelaciones se pueden incluir tokens especiales) que puede contener el texto a clasificar.

Cuando el tamaño del texto entrado no coincide con la entrada del modelo se deben tomar medidas para ajustar dicho tamaño al esperado. En caso de ser menor la entrada la mayoría de autores [[14](#_ENREF_14), [15](#_ENREF_15), [42](#_ENREF_42)] deciden aplicar un “padding” que consiste con rellenar la entrada con tokens especiales ignorados por el modelo para completar el tamaño. De lo contrario, si es mayor, por lo general la entrada se trunca para forzar el tamaño requerido. Como es de suponer este último método puede llevar a la pérdida de información que podría ser significativa para definir la clasificación, por ello es muy importante elegir bien este tamaño de entrada para cada dominio específico de forma que la gran mayoría de ejemplares a clasificar se encuentren dentro del tamaño de entrada seleccionado [[31](#_ENREF_31)].

Una vez definida la entrada el interior del modelo se mantiene estructuralmente igual y se debe decidir qué parte de su salida será utilizada posteriormente en las capas adicionales de clasificación. Por ejemplo autores como [[14](#_ENREF_14)] utilizan el primer token de la salida como representación del texto completo, mientras otros como [[42](#_ENREF_42)] utilizan operaciones como la media o suma de los tokens de salida para obtener un valor más representativo.

La ya mencionada agregación de capas de neuronas adicionales a la salida del modelo de lenguaje es imprescindible para convertir su salida en un valor de decisión. Uno de los enfoques más comunes utilizados por [[14](#_ENREF_14), [15](#_ENREF_15), [42](#_ENREF_42)] es utilizar una capa lineal que agrupe la información contenida en la salida en la cantidad de clases posibles y finalmente a través de una función de activación obtener una predicción.

En el caso de una clasificación binaria la salida se agruparía en una sola neurona que representaría la probabilidad de pertenencia a la clase positiva, la función de activación (generalmente sigmoide) a partir de un umbral de decisión clasificaría el resultado en una de las clases opuestas, en la figura 1.3 (Extraida de [[43](#_ENREF_43)]) se puede ver una representación de esta estructura. Si las 2 clases a clasificar no son exactamente opuestas se puede utilizar un enfoque multi-label de 2 clases con una función de activación argmax que elija la clase de mayor probabilidad entre los resultados.

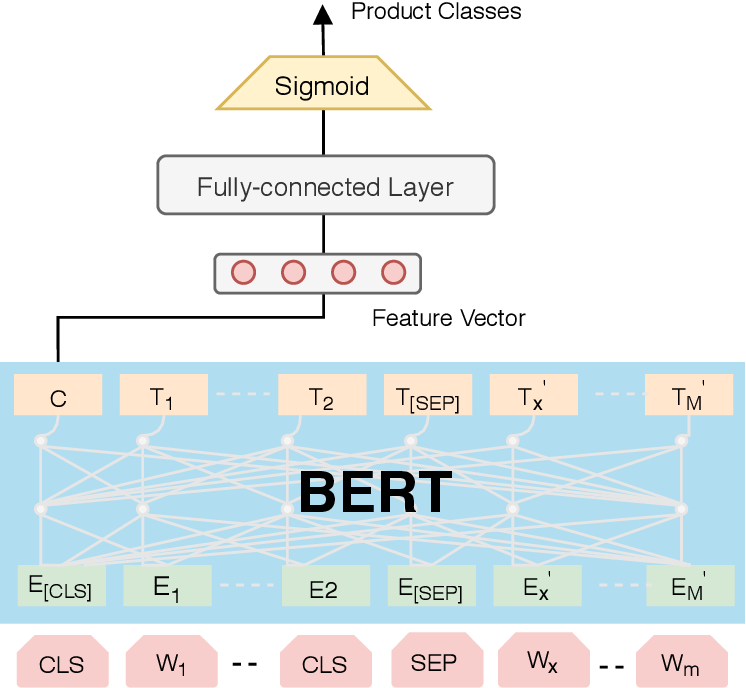


Figura 1.3: Estructura común de un modelo fine-tuned para una tarea de clasificación

# Incorporación de conocimiento del dominio

A pesar de los avances de los modelos pre-entrenados Transformers (MPT), estos presentan varias limitaciones. En primer lugar, aunque aprenden bien la semántica de las palabras comunes, su desempeño es deficiente con palabras poco frecuentes, debido a la distribución desigual de los datos disponibles. En segundo lugar, como son modelos estadísticos que se basan en señales de co-ocurrencia, carecen de habilidades para el razonamiento lógico, lo que afecta su capacidad para realizar tareas que requieren pensamiento más estructurado. Además, aunque los MPT pueden capturar una gran cantidad de conocimiento lingüístico, semántico y factual, su desempeño en la extracción de conocimiento y el razonamiento depende en gran medida de indicaciones sesgadas. Por último, pueden generar oraciones que carecen de sentido común, lo que plantea problemas éticos y de responsabilidad, especialmente en tareas donde pueden superar el desempeño humano [[18](#_ENREF_18)].

Para abordar estas limitaciones, una solución prometedora es combinar redes neuronales con conocimiento simbólico o específico del dominio concreto en que se piensa utilizar [[17](#_ENREF_17)]. Por un lado, el uso de conocimiento simbólico puede mejorar el manejo de palabras poco frecuentes, ya que estos recursos suelen tener una mayor cobertura de términos raros, complementando la supervisión textual limitada de los MPT. Por otro lado, el conocimiento simbólico también proporciona información relacional explícita y reglas que fortalecen las capacidades de razonamiento de los modelos. Además, esta integración puede mejorar la interpretabilidad de los MPT, permitiendo entender mejor cómo utilizan el conocimiento en tareas específicas [[44](#_ENREF_44)]. Incluso enfoques más simples enfoques con menor cantidad de información orientados a potenciar la atención de los MPT en términos concretos de mayor interés pueden tener un resultado positivo. Por cuestiones de comodidad a partir de aquí se referirá a los MPT que incorporan conocimiento como MPTC (Modelo Pre-entrenado Transformer con Conocimiento).

También se han hecho investigaciones incorporando conocimiento no solo específico de dominio sino también de uso general del lenguaje o conocimiento causal, tal es el caso de [[45](#_ENREF_45), [46](#_ENREF_46)].

## Granularidad y fuentes de conocimiento

Los MPTC integran diferentes granularidades de conocimiento para su uso en escenarios que requieren información a diferentes niveles de detalle. Por ejemplo, en el análisis de sentimientos se basan principalmente en características de palabras y, por tanto, requiere más información sobre entidades individuales. En cambio, en la tarea de generación de texto se basan en el conocimiento del sentido común, y la tarea de respuesta a preguntas se basan en reglas y grafos de conocimiento para la inferencia. En función de la granularidad los MPTC se podrían dividir en MPTC fusionados con texto, fusionados con árboles sintácticos, fusionados con grafos de conocimiento y fusionados con reglas.

Los MPTC fusionados con texto hacen uso de textos que suelen contener descripciones detalladas de entidades, relaciones y acontecimientos. Estos modelos utilizan los textos como referencias externas, extrayendo información crítica que beneficia principalmente a las tareas de respuesta a preguntas. Estos modelos codifican tanto las preguntas como los textos en *embeddings* y extraen los detalles más pertinentes de vastas referencias externas mediante cálculos de similitud semántica. A partir de una entrada enriquecida con conocimientos que concatena la información pertinente con la pregunta, pronostican las respuestas más probables [[18](#_ENREF_18)].

En el caso de los MPTC fusionados con árboles sintácticos transforman estos árboles en representaciones semánticas utilizando métodos adecuados de aprendizaje de representaciones y los combinan con representaciones de palabras, o los utilizan para seleccionar constituyentes críticos de secuencias de entrada para enmascararlos [[17](#_ENREF_17), [47](#_ENREF_47)]. Esta forma de conocimiento resulta beneficiosa para tareas que tienen en cuenta la estructura, como el análisis sintáctico, el etiquetado semántico de roles y la extracción de relaciones.

Con el avance de las técnicas de extracción de información, han surgido una multitud de grafos de conocimiento generales y específicos de dominio. Los grafos de conocimiento proporcionan una forma estructurada de representar información rica en forma de entidades y relaciones entre ellas [[48](#_ENREF_48)]. Un enfoque implica el aprendizaje de representaciones para los grafos y la posterior fusión de sus representaciones con representaciones de palabras alineadas mediante un módulo de infusión como es el caso de *ERNIE* [[49](#_ENREF_49)]. Otro enfoque transforma los grafos y las secuencias en una estructura de datos unificada, codificándolos con el mismo modelo. Con un conocimiento rico y detallado de las entidades, los MPTC fusionados con grafos de conocimiento son versátiles y cubren una amplia gama de tareas de comprensión y generación del lenguaje.

También, las reglas pueden caracterizarse como directrices formales o formulaciones lógicas rigurosas. Su principal ventaja reside en la interpretabilidad y responsabilidad que ofrecen las estructuras matemáticas estrictas y un proceso de inferencia transparente [[50](#_ENREF_50)]. Los MPTC pueden emplear este razonamiento simbólico, a partir de sus resultados de predicción, lo que resulta útil para las tareas de razonamiento que hacen hincapié en la transparencia y la responsabilidad.

## Métodos de inyección de conocimiento

El método de inyección de conocimiento desempeña un papel importante en la eficacia y la eficiencia de la integración entre los MPT y el conocimiento, así como en la gestión y el almacenamiento del conocimiento. De hecho, dicta el tipo de conocimiento que puede integrarse y su forma [[18](#_ENREF_18)]. A continuación, se tratarán algunas de las formas de inyección utilizadas en la literatura haciendo énfasis en tareas de clasificación que es el objetivo de este trabajo.

### Inyección directa de texto

Una primera estrategia sencilla para la integración de conocimientos consiste en ampliar directamente los textos de entrada con conocimientos adicionales, aprovechando el principio de extensión de *prompts*. Este texto enriquecido se emplea tanto en la fase de ajuste fino como en el posterior procedimiento de clasificación. La aplicación más sencilla de esta solución consiste en convertir toda la información pertinente del grafo de conocimiento en una cadena que se añade al final del texto original. Un ejemplo de esto es la estructura utilizada en [[51](#_ENREF_51)] que se puede ver en la figura 1.4.

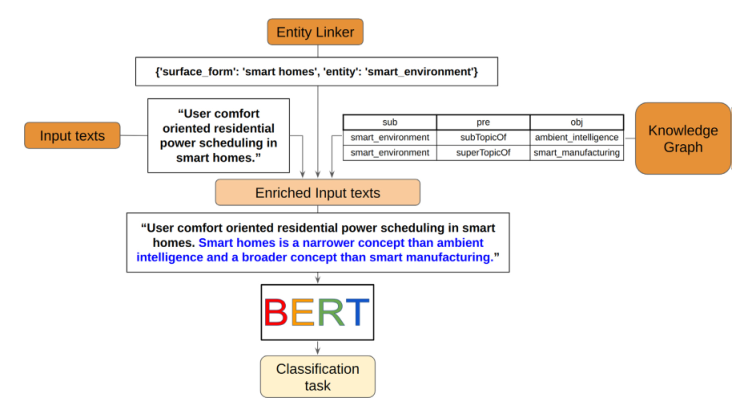


Figura 1.4: Ejemplo de una aplicación de la inyección directa de texto

### Vectores de características

En lugar de inyectar directamente los conocimientos en el texto, es posible incorporarlos como datos de características adicionales durante el proceso de clasificación. Para derivar representaciones contextualizadas a partir de características textuales, el MPT procesa el texto y devuelve las representaciones pertinentes. A continuación, esta salida se concatena con características adicionales derivadas de la representación del elemento a clasificar en una base de conocimientos [[18](#_ENREF_18), [51](#_ENREF_51)]. Dicha concatenación implica la fusión de 2 vectores, el vector con la información de salida del MPT y el vector de características del dominio. Esta representación aumentada se introduce en la capa de clasificación final.

Esta última capa de clasificación es comúnmente una capa linear cuyo tamaño de salida coincide con la cantidad de clases objetivo. Sin embargo, también hay enfoques que utilizan un Perceptrón Multicapa (PM) para llevar a cabo esta tarea o incluso una combinación de ambos como se puede ver en el ejemplo de estructura de la figura 1.5 que representa la estructura seguida por los autores de [[52](#_ENREF_52)].

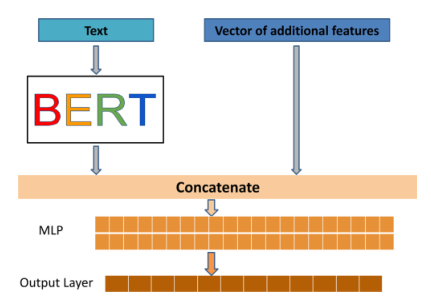


Figura 1.5: Integración de un vector de características en un MPT

### Modificación de *Embeddings*

Los *embeddings* de un modelo Transformer acumulan mucha información sobre la entrada que es procesada por las distintas capas del modelo, así que cualquier modificación en la forma en que estos se generan puede tener un impacto notable en el desempeño del modelo. Por lo tanto, es natural asumir que estos constituyen un objetivo jugoso al cual se le puede adicionar información del dominio que resulte de utilidad para el modelo, ahí surje la idea de este enfoque. De manera similar al enfoque de la sección anterior se hace uso normalmente de un vector de características que agrupe toda la información del dominio referente a la entrada que se toma como *embeddings* del dominio, este es fusionado con los *embeddings* que el MPT generó de forma natural a partir de la entrada para generar unos nuevos *embeddings* que contengan tanto la información de las palabras y sus posiciones como la información del dominio que se busca aprovechar. Dicha fusión de los *embeddings* se puede hacer de varias maneras según el contexto de la situación y está ligado al tipo de información de la que se disponga. Una representación general de esta estrategia se puede apreciar en la figura 1.6.

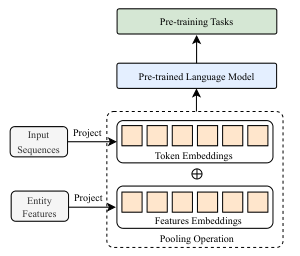


Figura 1.6: Integración del conocimiento en los embeddings del MPT

Una variante de este enfoque es la usada por [[53](#_ENREF_53)] donde en vez de utilizar un vector con características utilizan información textual extra que pueda ser utilidad para una entrada determinada, la cual es tratada como una entrada simultáneamente por otra capa de *embeddings* diferente a la original y ambos *embeddings* resultantes son concatenados antes de ser introducidos en el resto del modelo.

### Recuperación de bases de datos

Los MPTC basados en la recuperación están diseñados principalmente para tareas de respuesta a preguntas abiertas que requieren conocimientos previos relacionados con las preguntas, estos modelos emplean el conocimiento como referencia externa [[51](#_ENREF_51), [54](#_ENREF_54)]. A diferencia de los MPTC que integran el conocimiento en los parámetros del modelo, su objetivo principal es recuperar, seleccionar y codificar la información más relevante de amplios repositorios de conocimiento. Fundamentalmente, dado que no es necesario almacenar grandes cantidades de conocimientos dentro de los parámetros del modelo, estos permiten una actualización más eficaz y cómoda ante el cambio frecuente de conocimientos.

La información recuperada para ser utilizada de forma auxiliar puede tener distintos orígenes, abunda en este tipo de modelos el uso de grafos de conocimiento para extraer la información [[54](#_ENREF_54)]. Sin embargo, en los últimos años se ha potenciado enormemente en el caso de la generación de texto la incorporación de grandes bases de datos vectoriales que buscan contenido relevante para la pregunta según su similitud con el *prompt* inicial y este contenido es pasado a la ventana de contexto del modelo para que lo utilice de referencia al generar su respuesta [[55](#_ENREF_55)]. Un esquema básico de este proceso está representado en la figura 1.7.

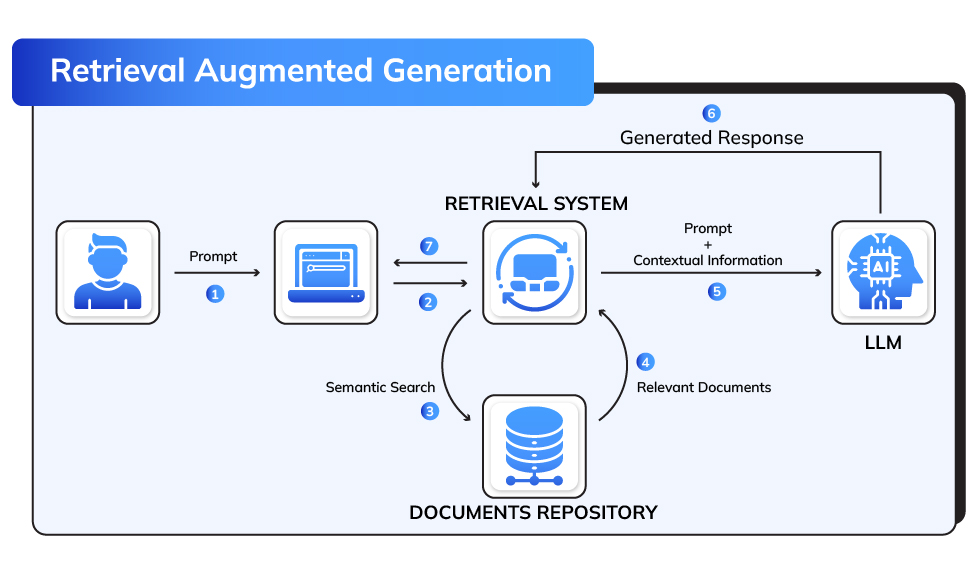


Figura 1.7: Estructura general de un modelo integrado con una base de datos vectorial para recuperación

### Guiados por reglas

Una dirección de investigación destacada es la transformación de la representación aprendida por los MPT en conceptos y el posterior razonamiento con conocimiento simbólico, manteniendo el aprendizaje efectivo de los modelos y la interpretabilidad y responsabilidad del conocimiento simbólico. Los MPTC guiados por reglas se centran principalmente en la fase de razonamiento basado en reglas [[18](#_ENREF_18)]. Sus características subyacentes permiten la combinación basada en principios de aprendizaje robusto y razonamiento eficiente, junto con la interpretabilidad que ofrecen los sistemas simbólicos.

Por ejemplo, RuleBert [[50](#_ENREF_50)] se centra en guiar al MPT para que razone en las tareas de razonamiento deductivo empleando reglas. Concretamente, predice la probabilidad correcta del resultado del razonamiento para las reglas y los hechos dados tras el ajuste fino como se observa en la figura 1.8

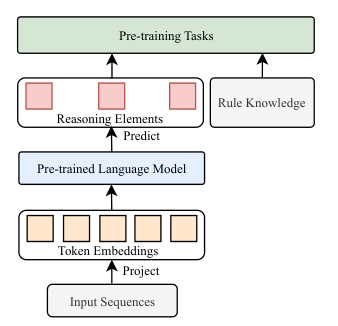


Figura 1.8: Uso de reglas en MPT

### Pre-entrenamiento adicional

Otra estrategia de incorporación de conocimiento incluye la realización de pre-entrenamiento adicional de forma no supervisada sobre corpus de texto estrechamente relacionados al dominio concreto al que se busca especializar el modelo, tal es el caso de [[56](#_ENREF_56)]. Hay autores que incluso han utilizado fuentes de datos estructuradas como grafos de conocimiento y los ha convertido a contenido textual preparado para esta tarea e incluso integrando ambos enfoques buscando la asimilación del modelo de usos prácticos de las entidades contenidas en el grafo de conocimiento. Aunque este método en muchos casos ha dado resultados positivos también demanda una cantidad recursos de procesamiento significativa en comparación al resto de enfoques. Una representación de este enfoque se puede apreciar en la figura 1.9. Una variante es realizar un pre-entrenamiento desde cero con la estrategia de entrenamiento y corpus de texto originales del modelo adicionando los corpus de texto adicionales del dominio objetivo desde esa fase de entrenamiento inicial tal como hacen en [[15](#_ENREF_15)], esto es naturalmente aún más costoso computacionalmente que la variante originalmente explicada.

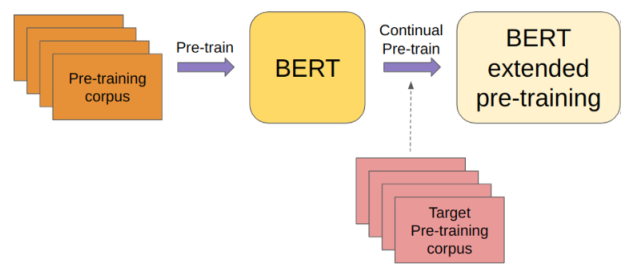


Figura 1.9: Esquema del pre-entrenamiento adicional en datos textuales del dominio

Este enfoque es muy común en la especialización de los modelos a un campo de investigación concreto sin ligarlos a una tarea concreta de minería. Por ejemplo, en [[48](#_ENREF_48)] se analizan varios MPTC especializados como *BioBERT* [[57](#_ENREF_57)], *BioELMo* [[58](#_ENREF_58)] y *BlueBERT* [[59](#_ENREF_59)] en el campo médico que son MPT normales a los que se les realizó un proceso de pre-entrenamiento extra sobre textos médicos, aunque en algunos casos se les hicieron cambios adicionales. Además en [[60](#_ENREF_60)] evalúan diferentes enfoques de pre-entrenamiento para introducir información específica de un entorno *e-comerce* concreto en un modelo con buenos resultados.

# Antecedentes

En el campo de la clasificación de comentarios de usuarios según su relevancia para equipos de desarrollo de software ha habido varias investigaciones, varias que se han centrado en este aspecto [[10-12](#_ENREF_10), [56](#_ENREF_56)] y otras que han utilizado un enfoque más general de clasificación teniendo en cuenta aspectos adicionales como reportes de errores o petición de características [[14](#_ENREF_14), [15](#_ENREF_15), [42](#_ENREF_42)].

Los autores de [[10](#_ENREF_10)] crearon una herramienta llamada AR-Miner para analizar grandes cantidades de comentarios de usuarios. La primera etapa del proceso de esta herramienta es el filtrado de las opiniones irrelevantes para reducir el ruido y la gran cantidad de datos a analizar. Para ello utilizaron un algoritmo de entrenamiento semi-supervisado, Expectation Maximization for Naive Bayes (EMNB) que permite aprovechar las grandes cantidades de comentarios sin etiquetar juntos a una cantidad significativamente menor de datos etiquetados.

En [[11](#_ENREF_11)] los autores analizan la relevancia de opiniones de usuarios para soporte de equipos de desarrollo de software realizando un análisis bastante exhaustivo probando varios enfoques tradicionales de aprendizaje automático como Support Vector Machine, Naive Bayes, Logistic Regression, k-nearest neighbors, entre otros. Dichos algoritmos probados con distintas técnicas de extracción de características como TF-IDF, PMI y la ocurrencia de palabras propias del campo de dominio a través de un glosario de términos. En el estudio evalúan la solución para distintas combinaciones de algoritmos de clasificaicón y técnicas de extracción de características. En

Los autores de [[14](#_ENREF_14)] no analizan específicamente la relevancia sino un enfoque de clasificación entre irrelevante, reporte de error y petición de características. La tarea de clasificación es realizada a través del fine-tunning de los modelos de lenguaje de arquitectura Transformer English-BERT, M-BERT e Italian-BERT.

El primer caso dentro de esta problemática donde se aplicó una técnica que incorporaba conocimiento del dominio fue [[56](#_ENREF_56)], al menos según la investigación realizada en el área. En este trabajo sus autores realizan un pre-entrenamiento adicional de modelos basados en BERT en sets de datos de diferentes campos y diferentes objetivos. Uno de estos fue precisamente en un conjunto de más de 160 mil opiniones sobre aplicaciones tomado de [[10](#_ENREF_10)] para la tarea de clasificación de opiniones. Un caso similar al anterior es el de [[15](#_ENREF_15)], donde trabajan en varios sets de datos etiquetados para clasificaciones diferentes cada uno, algunas de las cuales son: irrelevante, reporte de error, petición de características, evaluación de aspectos, elogio, duda, portabilidad, usabilidad, entre otros. Este estudio también realiza un fine-tunning de varios modelos de lenguaje de arquitectura Transformer como BERT, ALBERT, RoBERTa y XLNET. Además, hicieron pruebas con versiones personalizadas de estos modelos en las que hicieron un pre-entrenamiento desde cero con datos adicionales de redes sociales y textos de estilo informal.

Este enfoque utilizado en los 2 últimos casos cae dentro de la incorporación de conocimiento adicional mediante pre-entrenamiento adicional, explicado en la sección 1.4.2.6.

# Métodos de evaluación

Al entrenar un modelo de inteligencia artificial como en otras tareas es de vital importancia alguna forma de cuantificar que tan bien desempeña su tarea. Esto se hace mediante técnicas y métricas de evaluación que no son más que fórmulas que hacen precisamente esto, puntuar el desempeño del modelo según aspectos concretos en función del tipo de métrica utilizada [[34](#_ENREF_34)]. Además, permiten comparar el desempeño entre distintos modelos.

El método de evaluación más simple es la utilización de un set de datos que esté explícitamente dividido en un conjunto de entrenamiento y un conjunto de prueba de forma que los resultados de las métricas obtenidas con él sean replicables bajo esas condiciones [[61](#_ENREF_61)]. Sin embargo, no todos los sets de datos están tan claramente divididos, así que es común utilizar una validación cruzada en la que el set de datos es manualmente dividido 2 conjuntos, uno de entrenamiento y otro de prueba

Muchos autores [[14](#_ENREF_14), [15](#_ENREF_15), [42](#_ENREF_42)] en estos casos utilizan una variante de la validación cruzada, la validación cruzada de K grupos (K-folds cross-validation) porque proporciona una estimación más robusta de la capacidad de generalización del modelo en comparación con la simple división del conjunto de datos en conjuntos de entrenamiento y prueba [[31](#_ENREF_31)]. Para realizarla se divide aleatoriamente el conjunto de datos en un número K de subconjuntos o "folds". El modelo se prueba en uno de estos subconjuntos y se entrena en los K-1 subconjuntos restantes. Este proceso se repite K veces, con diferentes combinaciones de conjuntos de entrenamiento y prueba, hasta que cada subconjunto haya sido usado como conjunto de prueba una vez. De este proceso se obtienen K resultados de métricas, uno por cada iteración, estos son promediados para obtener los resultados de la evaluación [[26](#_ENREF_26)].

## 0Métricas

Las métricas comunes en problemas de clasificación suelen partir de una matriz de confusión como la de la tabla 1.1 que lleva un conteo según la cantidad de casos clasificados de cierta forma.

Tabla 1.1 Matriz de confusión para una clasificación específica

|  |  |  |
| --- | --- | --- |
|  | Clasificación verdadera | |
| Clasificación predicha | Positive (P) | Negative (N) |
| Positivo (P) | True Positive (TP) | False Positive (FP) |
| Negativo (N) | False Negative (FN) | True Negative (TN) |

Donde se cumple que:

TP: Cantidad de casos en los que se predice que el elemento pertenece a la clase correctamente.

FN: Cantidad de casos en los que se predice que el elemento no pertenece a la clase cuando esto es falso.

FP: Cantidad de casos en los que se predice que el elemento pertenece a la clase cuando esto es falso.

TN: Cantidad de casos en los que se predice que el elemento no pertenece a la clase correctamente.

Algunas de las métricas más comunes encontradas en la literatura son las siguientes:

**Accuracy:** Es la relación entre las muestras clasificadas correctamente y el número total de muestras. Es una métrica sencilla e intuitiva, pero puede inducir a error cuando la distribución de clases está desequilibrada y hay demasiada representatividad de una sola clase [[61](#_ENREF_61)]. Matemáticamente, puede representarse como:

**Precision:** La precisión mide la exactitud de las predicciones positivas. Más específicamente, denota la proporción de casos Predichos Positivos que son correctamente Positivos Reales [[21](#_ENREF_21)]. Una precisión alta significa que el modelo no genera muchos falsos positivos. Matemáticamente se puede representar como:

**True Positive Rate / Recall:** Mide la proporción de instancias verdaderas positivas (es decir, instancias correctamente clasificadas como positivas) del número total de instancias positivas [[34](#_ENREF_34)]. Un valor alto de recall indica que el modelo tiene menos falsos negativos, lo que significa que puede identificar correctamente la mayoría de los casos positivos. Matemáticamente se representa de la siguiente forma:

**F1 score:** Métrica que busca equilibrar los valores de precision y recall calculando su media armónica [[61](#_ENREF_61)]. De especial utilidad cuando se quiere buscar una precisión lo más general posible y balanceada. Matemáticamente se representa como:

# Herramientas utilizadas

## Lenguaje de programación: Python

Se eligió el lenguaje de programación Python porque es un lenguaje simple y cómodo de trabajar que cuenta con muchas bibliotecas y herramientas para realizar procesamiento lingüístico y tareas de aprendizaje [[26](#_ENREF_26), [31](#_ENREF_31)]. También es el lenguaje que predomina abrumadoramente en las investigaciones relacionadas consultadas y la comunidad de personas, investigadores, documentación y materiales de aprendizaje es considerablemente superior a otros como Java o C++ en este campo. Python además tiene otras ventajas como [[62](#_ENREF_62)]:

* Aparte de la gran cantidad de funcionalidades que permite su biblioteca estándar cuenta con una cantidad significativa de bibliotecas externas que facilitan enormemente casi cualquier tipo de problema informático: trabajo con bases de datos, desarrollo web, análisis de datos, entre otros.
* Python puede ser ejecutado en cualquier sistema operativo moderno, ya sea Windows, Linux/ UNIX, macOS, entre otros.
* Python tiene una licencia de tipo Open Source, que permite ser utilizado y distribuido libremente.

## Bibliotecas principales

**Transformers:** Biblioteca de Hugging Face que proporciona APIs y herramientas para descargar y entrenar fácilmente modelos pre-entrenados del estado del arte. Soporta la interoperabilidad entre PyTorch, TensorFlow y JAX. Esto proporciona la flexibilidad de utilizar un framework diferente en cada etapa de la vida de un modelo; entrenar un modelo en pocas líneas de código en un framework, y cargarlo para inferencia en otro [[63](#_ENREF_63)].

**PyTorch:** Un framework integral de Machine Learning que permite una experimentación rápida y flexible y una producción eficiente gracias a un front-end fácil de usar, una formación distribuida y un ecosistema de herramientas y bibliotecas. Algunas de las facilidades que ofrece son [[64](#_ENREF_64)]:

* Cálculo de tensores con fuerte aceleración de GPU, lo que permite operaciones numéricas arbitrarias en estructuras multidimensionales.
* Diferenciación automática para la creación y entrenamiento de redes neuronales profundas.
* Cálculo de gráficos dinámicos, lo que permite a los usuarios cambiar el comportamiento de la red sobre la marcha, en lugar de esperar a que se ejecute todo el código. Esto lo vuelve más recomendado para entornos de experimentación como este trabajo.

PyTorch fue seleccionada después de ser comprada con otras alternativas ya que para el trabajo con redes neuronales existen 3 bibliotecas o marcos de trabajo principales dentro del campo que facilitan su uso: scikitLearn, PyTorch y Tensorflow (En el caso de esta última siempre se utiliza en conjunto con un módulo llamado Keras que ofrece muchas ventajas adicionales). En la tabla 7 se puede ver una comparación de algunos aspectos de estas bibliotecas.

Tabla 1.2: Comparación de las principales bibliotecas para redes neuronales

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspecto** | **ScikitLearn** | **Tensorflow+Keras** | **PyTorch** |
| Soporte para redes neuronales | Brinda opciones pero poco personalizables | Fuertemente enfocado en aprendizaje profundo y redes neuronales | Fuertemente enfocado en aprendizaje profundo y redes neuronales |
| Facilidad de uso | Muy alta | media | alta |
| Grafos de computación | Dinámicos | Estáticos | Dinámicos |
| Despliegue y producción | Opciones simples | Opciones robustas para casi cualquier plataforma | Opciones simples pero en crecimiento |

Para el desarrollo de soluciones pensadas para producción el uso de Tensorflow ciertamente tiene una gran ventaja. sin embargo, en el campo de la investigación científica es más popular PyTorch debido al uso de grafos de computación dinámicos (dynamic computation graph) ya que permiten una mayor de flexibilidad y facilidad de debugging, ideal para la experimentación.

**Pytorch Lightning:** El módulo de Lightning es una estructura organizativa para PyTorch que permite la máxima flexibilidad y un mínimo de complejidades. Actúa como una "receta" de modelo que especifica todos los detalles de la formación. Algunas de las ventajas que ofrece son [[65](#_ENREF_65)]:

* Mejora la legibilidad del código al separar el código de investigación del código de ingeniería.
* Facilita la reproducción de modelos al automatizar gran parte del ciclo de entrenamiento y eliminar el código repetitivo.
* Ayuda a automatizar el proceso de optimización de los modelos, simplificando el entrenamiento y la experimentación.

**TorchMetrics:** es una colección de más de 100 implementaciones de métricas para PyTorch, junto con una API fácil de usar para crear métricas personalizadas. Esta biblioteca aporta varias facilidades clave para los desarrolladores de modelos de aprendizaje automático, especialmente cuando se trabaja en conjunto con PyTorch y PyTorch Lightning. Permite [[66](#_ENREF_66)]:

* Proporciona una interfaz estandarizada para aumentar la reproducibilidad de los modelos y las métricas.
* Las métricas están optimizadas para el entrenamiento distribuido.
* Las métricas pueden acumular resultados automáticamente a través de múltiples lotes, lo que es útil para calcular métricas a lo largo de todo el proceso de entrenamiento.

**Pandas:** Biblioteca especializada en el manejo y análisis de estructuras de datos. Indispensable para el trabajo con grandes sets de datos. Es la más utilizada en su campo por sus grandes ventajas, entre las que se encuentran [[67](#_ENREF_67)]:

* El objeto DataFrame, rápido y eficaz para la manipulación de datos con indexación integrada.
* Herramientas de lectura y escritura de datos entre estructuras de datos en memoria y distintos formatos.
* Unión y división de conjuntos de datos de alto rendimiento.

**Natural Language Tool Kit (NTLK):** Conjunto de bibliotecas para trabajar con el lenguaje natural, contiene sets de recursos útiles para el procesamiento y herramientas para tareas como la tokenización, lematización, entre otras [[68](#_ENREF_68)].

**NumPy:** Biblioteca fundamental para el trabajo avanzado con datos numéricos, provee entre otras herramientas estructuras muy versátiles para el trabajo con vectores multidimensionales [[69](#_ENREF_69)].

**Scikit-learn:** proporciona una colección de algoritmos para el aprendizaje automático y la minería de datos, incluye una amplia gama de algoritmos de clasificación, regresión, agrupamiento y reducción de dimensionalidad. Además, facilita el uso de métodos de validación como la validación cruzada [[70](#_ENREF_70)].

# Conclusiones parciales

* Los algoritmos de aprendizaje supervisado requieren de datos etiquetados suficientes para realizar las de etapas de entrenamiento y evaluación.
* Las redes neuronales profundas, sobre todo de la arquitectura Transformer, han obtenido resultados del estado del arte en casi todas las tareas del procesamiento del lenguaje natural.
* El uso de modelos de lenguaje pre-entrenados de arquitectura Transformer permite aprovechar mediante la transferencia de aprendizaje su conocimiento general del lenguaje para reducir significativamente la cantidad de datos de entrenamiento requeridos y ajustarlos a tareas específicas como la clasificación.
* Existen varios enfoques para inyectar conocimiento en un modelo Transformer, siendo los principales estudiados: Inyección directa de texto, uso de vectores de características, modificación de los embeddings, recuperación de bases de datos, modelos guiados por reglas y el empleo de pre-entrenamiento adicional.
* Las soluciones existentes sobre clasificación de opiniones de usuarios para el soporte de software que exploran el uso de modelos de aprendizaje profundo de arquitectura Transformer son pocos.
* Solo una solución anterior en la literatura a este problema con modelos de arquitectura Transformer utiliza un enfoque para inyectar conocimiento de dominio en el modelo y este enfoque es el más costoso en términos de recursos y tiempo de entrenamiento.
* En tareas de clasificación las métricas más utilizadas son accuracy. precision, recall y F1-score.
* La biblioteca Transformers es imprescindible para el trabajo con esta arquitectura ya que permite cargar y utilizar los distintos modelos de lenguaje.
* Se destaca el uso de PyTorch Lightning en la compactación de los procesos de entrenamiento y evaluación de modelos manteniendo una gran flexibilidad.

Capítulo 2: Método de solución planteado

En este capítulo se expone todo lo referente al diseño e implementación de la solución propuesta, haciendo énfasis en cada una de sus etapas. Se presentan además artefactos de diseño como diagramas de flujo y casos de uso del sistema que permiten mostrar el funcionamiento de la solución.

# 2.1 Descripción de la solución propuesta

## 2.1.1 Descripción general

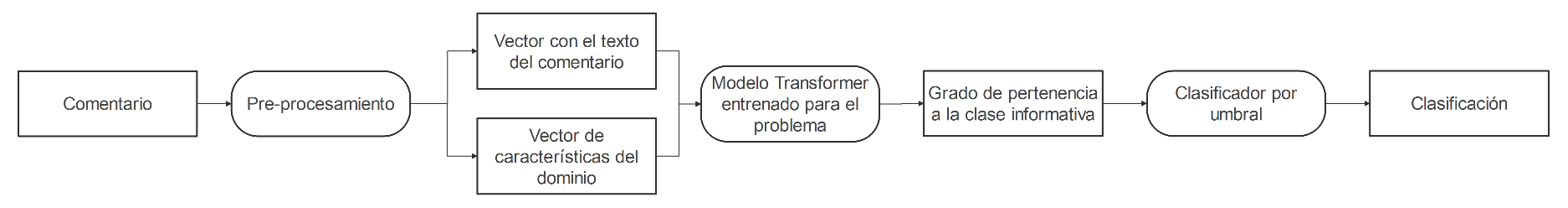
Se propone una solución para la clasificación de opiniones de usuarios según su relevancia para un equipo de desarrollo a través del uso de modelos de lenguaje de arquitectura transformer con uso de conocimiento de dominio a través de vectores de características. La clasificación consiste en decir si el comentario es relevante o no.

Figura 2.1: Esquema del flujo de trabajo de la solución

Este proceso (representado en la figura 2.1) empieza al entrar la opinión al sistema y comienza la etapa de pre-procesamiento. Una vez concluida esta fase se obtiene un vector con la información textual del comentario y otro con las características del dominio que se utilizaran como información auxiliar. Ambos vectores son utilizados por el modelo de lenguaje cuya salida fue modificada especialmente para tareas de clasificación binaria. Este modelo devolverá como salida el grado de pertenencia a la clase relevante del comentario introducido y su clasificación es decida a través de un umbral de decisión.

En este trabajo primero se desarrolló la solución sin hacer uso de conocimiento de dominio con 5 modelos pre-entrenados diferentes: *GPT-2*, *ALBERT*, *XLNET*, *RoBERTa* y *BERTweet.* De estos, se seleccionaron los 2 con mejores resultados para probar 4 variantes diferentes de la solución que utiliza conocimiento de dominio, las cuales serán más detalladas en su correspondiente sección.

## 2.1.2 Pre-procesamiento

El comentario entrado al programa es tokenizado antes de ser introducido al modelo. Este proceso es llevado a cabo por un tokenizador propio de cada modelo que es descargado automáticamente de su correspondiente repositorio en Hugging Face a través de la biblioteca Transformers. Dicho tokenizador puede ser configurado a través de sus parámetros para regular aspectos de su comportamiento, para la solución actual se decidió activar los siguientes:

* Añadir los tokens especiales: Son introducidos tokens que le indican al modelo el inicio del comentario (CLS) y las separaciones entre oraciones (SEP) para mejorar la capacidad de comprensión del modelo sobre la estructura de las oraciones.
* Tamaño límite: Se estableció un tamaño límite de entrada de 200 tokens, más que suficiente para cubrir la mayoría de comentarios en redes y tiendas de aplicaciones sobre programas y la totalidad de los comentarios utilizados en los sets de datos de entrenamiento y prueba.
* Activar padding: Son añadidos tokens especiales [[23](#_ENREF_23)] para rellenar las entradas menores a la capacidad de entrada del modelo y garantizar la uniformidad en el tamaño de los datos recibidos. Dichos tokens son ignorados durante el procesamiento del comentario.
* Activar truncación: Las entradas de tamaño mayor al permitido son recortadas en su final para acotarlas al tamaño de entrada establecido.

Al terminar este proceso el resultado son los tokens y la máscara de atención correspondientes con la información del comentario que es introducido al modelo para la siguiente etapa.

## 2.1.3 Construcción del vector de características

Durante la etapa de pre-procesamiento aparte del vector con el comentario tokenizado también se construye el vector de características del dominio asociado a la opinión. Este vector de características es el eje central de la inyección de conocimiento del dominio en el modelo y el que será utilizado por la técnica de conocimiento. El cómo se utiliza exactamente el vector y la técnica de inyección de conocimiento en sí se describen en mayor detalle en la sección 2.1.4.1.

Para la construcción del vector se cuenta con un glosario de términos relevantes dentro del campo de desarrollo de software.

En este trabajo se proponen 2 formas de construir el vector de características, que hacen uso de características distintas de la opinión respecto a los términos que aparecen en el glosario: el conteo de los términos relevantes (vector RC) y la posición de los términos relevantes (vector RP). En ambos casos el vector de características generado es de igual tamaño que la entrada máxima del modelo.

### 2.1.3.1 Vector RC

Una forma para inicializar el vector de características del dominio de una opinión es en función de la cantidad de términos relevantes que aparecen en esa opinión. En este trabajo se le refiere a un vector construido de esta forma como Conteo de Relevantes o vector RC (por las siglas en inglés *Relevant* *Count*).

Se halla cuantas de las palabras usadas en la opinión están presentes en el glosario de términos relevantes en el desarrollo de software. Esto se hace bajo la idea de que mientras más términos relevantes tenga un comentario más probable es que este sea informativo o de utilidad para un equipo de desarrollo. Entonces el vector se inicializaría como un vector del tamaño de la entrada con ese número en todas las posiciones. Se decidió repetir el número del conteo esa cantidad de veces ya que un solo valor dentro de los más de 700 parámetros que normalmente tienen los modelos utilizados en este trabajo tendría un impacto casi imperceptible.

Un ejemplo más ilustrativo se puede observar en a continuación:

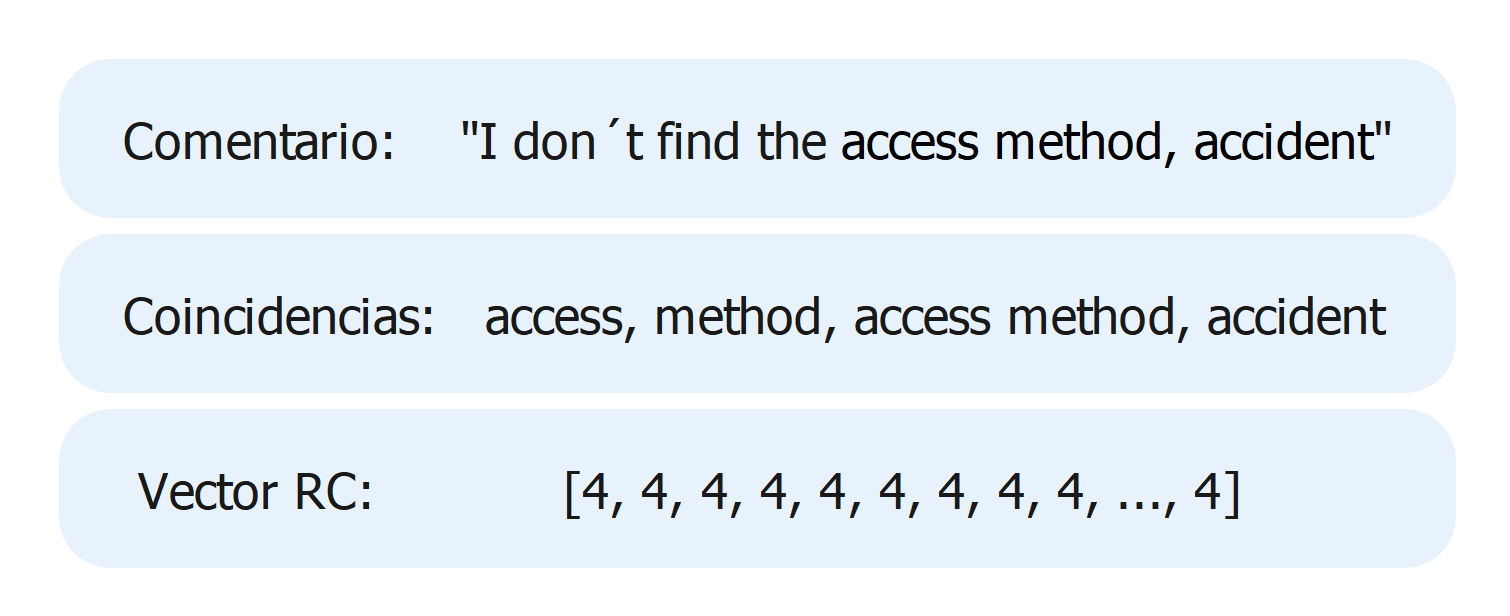


Figura 2.2: Ejemplo de vector RC

En el ejemplo de la figura 2.2 podemos ver que se encuentran 4 coincidencias en el comentario (*access*, *method*, *access method* y *accident*). Como sale a relucir en el glosario también hay términos compuestos o frases con “*access method*” siendo uno de muchos, en la mayoría de estos casos los términos independientes que componen la frase también se encuentran en el glosario por lo que encuentra coincidencias adicionales por cada término compuesto. Este conteo extra que fue originalmente un resultado accidental se mantuvo en la forma de hacer el conteo ya que tiene sentido premiar o puntuar más las formaciones complejas de palabras que son más específicas que términos sueltos a la hora de señalar que tan informativa es la opinión.

### 2.1.3.2 Vector RP

Otra forma para inicializar el vector de características del dominio de una opinión es en función de la posición de los términos relevantes que aparecen en esa opinión. En este trabajo se le refiere a un vector construido de esta forma como Posición de Relevantes o vector RP (por las siglas en inglés *Relevant* *Position*).

En este caso se crea un vector inicializado a 0 en todos sus valores de igual tamaño a la entrada máxima del modelo. Después utilizando como referencia las posiciones del vector de la opinión tokenizada introducida en el modelo se comprueban qué posiciones corresponden a términos que están presentes en el glosario de términos relevantes y en esas posiciones se les cambia el valor a 1. De esta forma el resultado es un mapeo del vector original de la opinión donde hay 1s en las posiciones relevantes según el glosario y 0 en las que no.

Retomando la opinión para este enfoque tenemos lo siguiente:

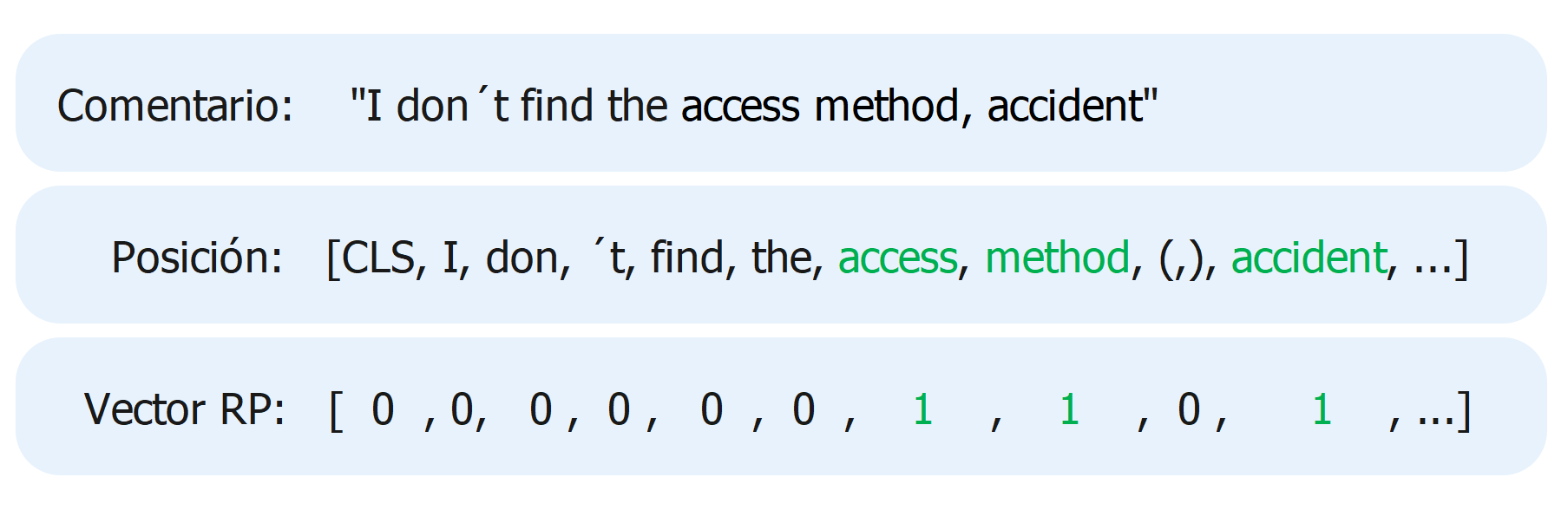


Figura 2.3: Ejemplo de vector RP

En el ejemplo de la figura 2.3 se encuentran las mismas coincidencias del enfoque anterior (*access*, *method* y *accident*) y en sus correspondientes posiciones del vector de características se le asigna el valor binario 1 indicando su relevancia como término presente en el glosario. En este caso la coincidencia compuesta de “*access* *method”* que fue contada doble en el enfoque por conteo no tiene objetivo aquí ya que sus términos independientes ya están incluidos. En caso de estuviera un término compuesto, pero no todos sus componentes fueran términos independientes del glosario se potenciarían igualmente todos los términos que forman parte del término compuesto.

## 2.1.4 Modelo de clasificación sin conocimiento

Se utilizaron modelos de leguaje pre-entrenados para probar la solución, a los cuales se les realizó un ajuste fino en el ámbito específico del problema antes de ser utilizados.

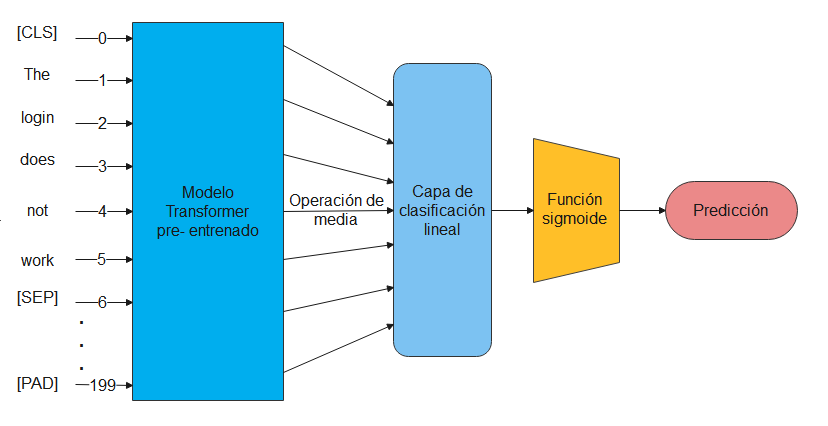


Figura 2.2: Arquitectura del modelo de clasificación sin conocimiento

La información introducida en el modelo es procesada por toda la red neuronal del modelo, La salida de este paso es un objeto que contiene el estado oculto de la última capa del modelo transformer, entre otros elementos.

Para utilizar esta salida se selecciona el estado oculto de la última capa del modelo transformer, que es una representación vectorial de la entrada. Se aplica una operación de media para obtener un vector promedio de la secuencia de entrada. Este paso esencialmente reduce la secuencia de tokens a un vector único que captura la información global de la entrada.

Luego, una capa lineal que es utilizada para ajustar el vector de características promedio a un único valor de salida. La capa lineal toma como entrada un vector de tamaño igual al tamaño del estado oculto del modelo pre-entrenado y produce una salida de tamaño 1, lo que es adecuado para una clasificación binaria. Esta capa actuaría como lo que se conoce en algunas investigaciones como la capa de clasificación final.

Finalmente, se aplica una función sigmoide al resultado de la capa lineal para convertirlo en un valor entre 0 y 1. Esto es necesario porque la salida de la capa lineal (anteriormente) no está normalizada, y la función sigmoide es una manera común de transformar esta salida en un valor que se puede interpretar como la probabilidad de pertenecer a la clase positiva en una clasificación binaria o el grado de pertenencia a la misma. En este caso dicha clase sería la clase “Relevante”

La decisión final es tomada con un simple umbral de decisión que clasifica el comentario original como “relevante” si la salida del modelo es un valor mayor a 0.5, lo cual indica que el comentario es suficientemente relevante para clasificarlo como tal, en caso contrario su relevancia es insuficiente y se clasifica como “no relevante”.

## 2.1.5 Introducción de conocimiento al modelo.

El objetivo es probar el potencial de la introducción de conocimiento del dominio en el modelo de solución recién explicado. Para lograr esto se siguió el esquema de la figura 2.3 que será explicado a continuación.

El proceso empieza con la creación de los vectores de características explicados en la sección 2.1.3, estos serán la fuente de conocimiento que guiará al modelo. El enfoque para hacer uso de estos vectores está basado en la solución explicada en la sección 1.4.2.2 donde son fusionados junto con la salida del modelo pre-entrenado para dotar a esta salida de dicho conocimiento como un extra útil. La fusión de ambos vectores no es más que su concatenación, añadiendo los valores del vector de características al final del vector de salida del modelo, obteniendo un vector final de tamaño igual a la suma de los tamaños de ambos. Este vector resultando con la información de dominio agregada es entonces introducido en la capa de clasificación final y seguirá el resto del proceso como la original explicada en la sección anterior.

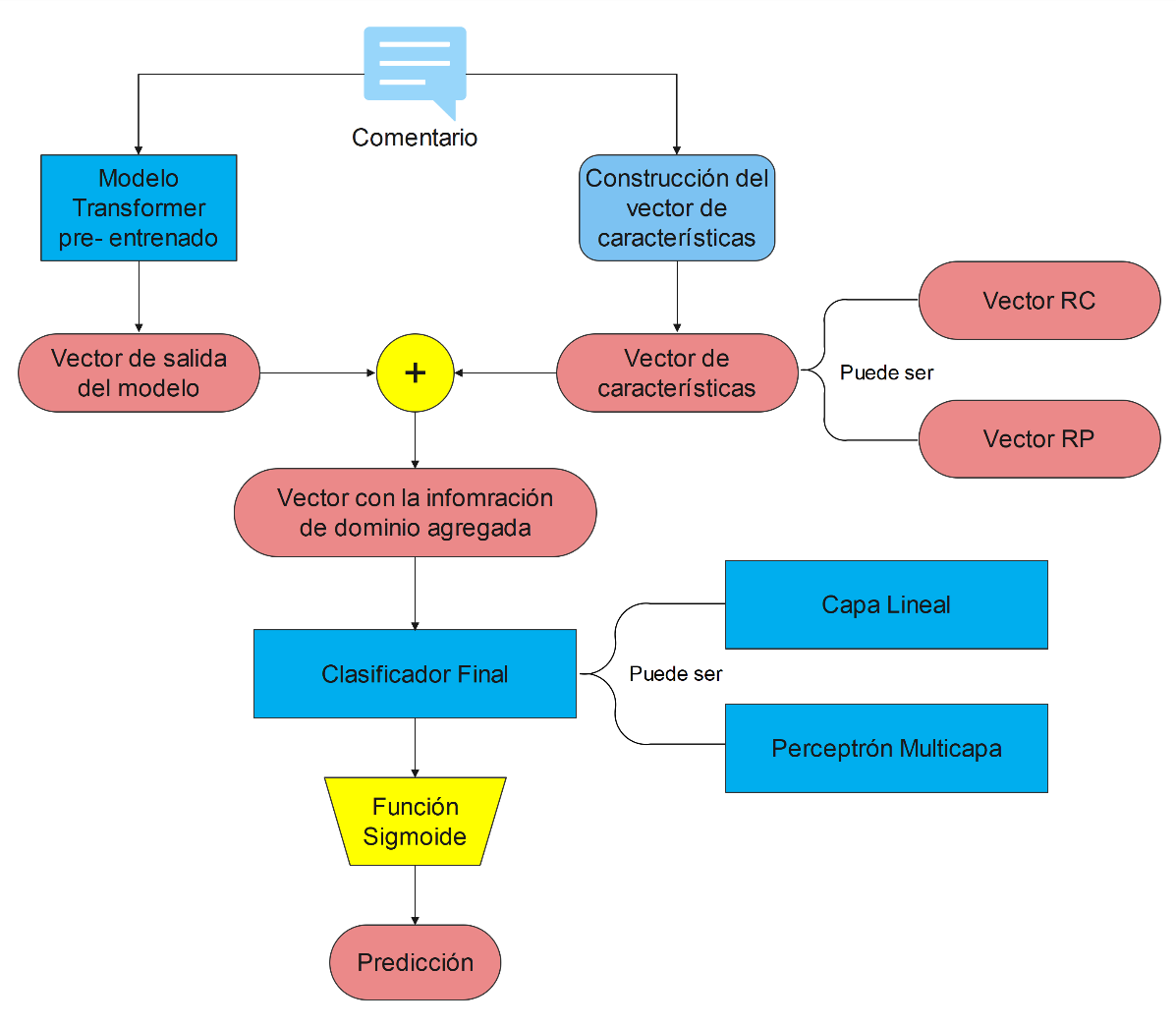


Figura 2.3: Arquitectura del modelo de clasificación con conocimiento

Se prepararon 2 variantes de la capa de clasificación final para ser evaluadas, una es la misma capa lineal que se utilizaba originalmente y la segunda variante, inspirada en el trabajo de [[51](#_ENREF_51)], constituye en un perceptrón multicapa de 3 capas, cuya última capa es una replicación de la capa lineal de la variante anterior que cumple la función de clasificador. El objetivo de esta segunda variante es tener 2 capas intermedias que procesen el vector fusionado con la información agregada y lo digieran antes de utilizarlo en la clasificación.

Como resultado de esto podemos decir que se cuenta con 4 variantes de solución que incorporan conocimiento del dominio en la solución inicial. Estas 4 variantes de solución es el resultado de combinar las 2 formas de inicializar el vector de características (sección 2.1.3) y los 2 tipos de capas de clasificación final que se plantean (párrafo anterior).

# 2.2 Desarrollo del clasificador de opiniones según su relevancia para el desarrollo de software

En este epígrafe se describen los elementos de ingeniería de software tomados en cuenta en el diseño e implementación de la solución. Entre estos elementos se encuentra el modelo de dominio, el diagrama de casos de uso del sistema y la descripción detallada de los casos de uso

## 2.2.1 Modelo de dominio

El modelo de dominio proporciona una visión estructural, ya que facilita representar el vocabulario y los conceptos claves del problema que se está modelando, así como las relaciones entre ellos. Dado que estos modelos son breves y bien estructurados facilitan capturar los conceptos de manera muy completa. Para el caso de esta investigación no existe un modelo de negocio asociado a la solución por lo que resulta más representativo modelar la información haciendo uso de un modelo de dominio como el mostrado en la figura 2.3.

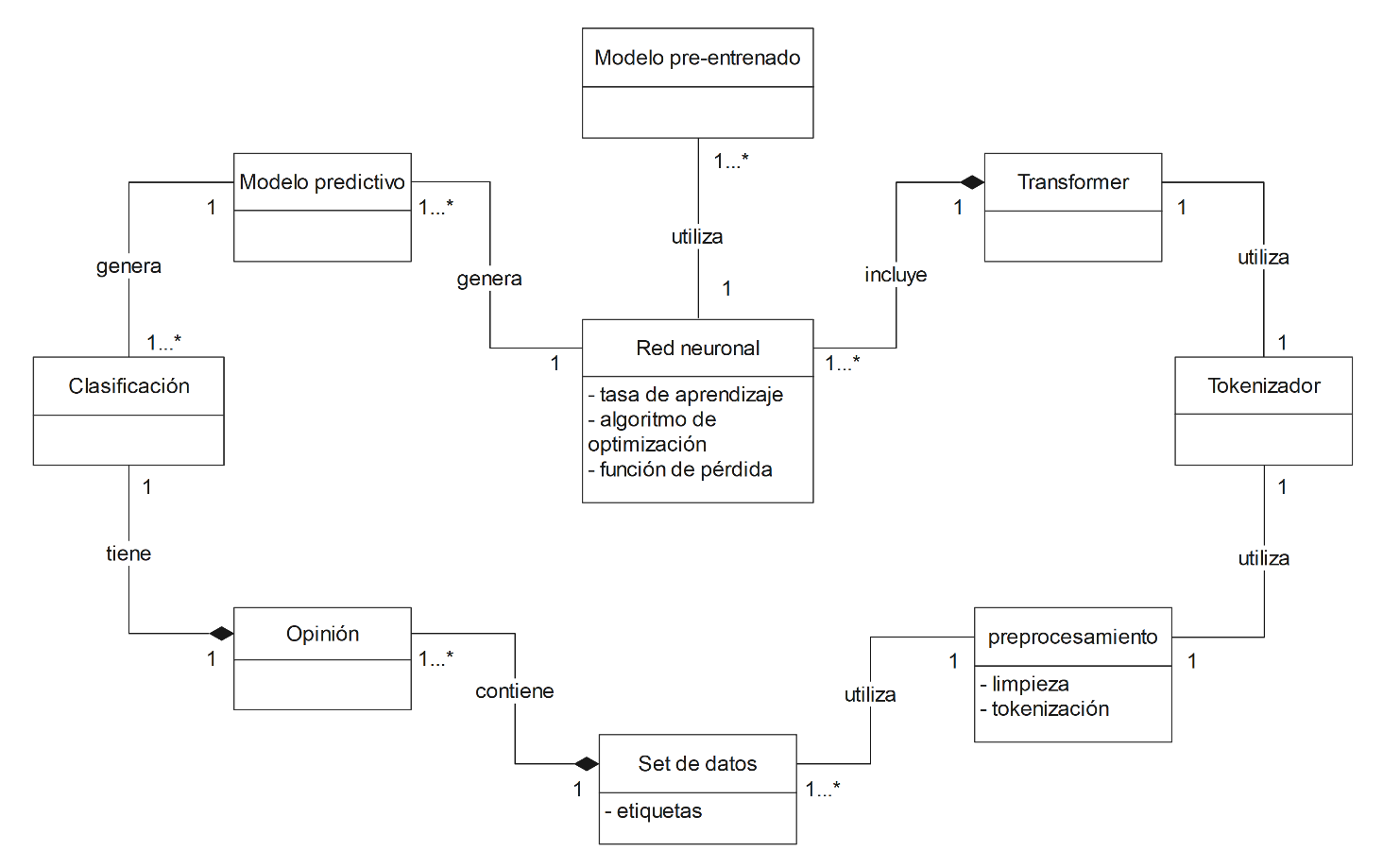
****

Figura 2.4 Modelo de dominio de la solución

En el modelo de dominio anterior solo se tomaron en cuenta aspectos directamente relacionados con la solución propuesta en esta investigación. En este se aprecia la relación entre los distintos conceptos partiendo del procesamiento del lenguaje natural hasta el resto formando hilos que relacionan a su paso todos los aspectos implicados.

## 2.2.2 Captura de requisitos

### 2.2.2.1 Diagrama de casos de uso del sistema

Los diagramas de casos de uso permiten visualizar de manera clara y concisa las funcionalidades del sistema, facilitando la identificación de requisitos, la detección de errores y la planificación de la implementación [[19](#_ENREF_19)]. En la figura 2.4 se muestra el diagrama de casos de uso diseñado como parte de la solución propuesta.

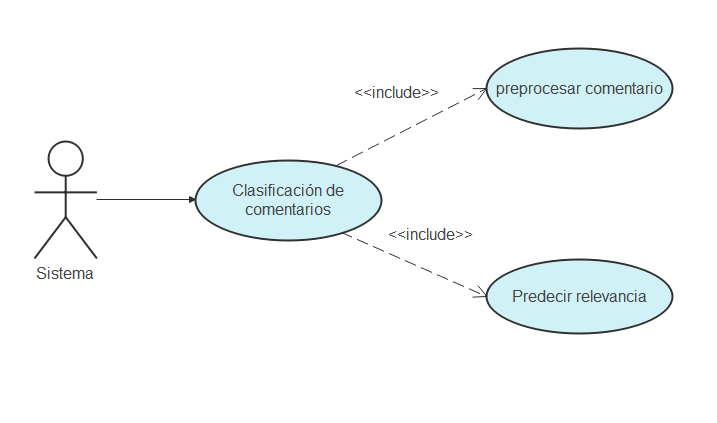


Figura 2.4: Diagrama de casos de uso del sistema

### 2.2.2.2 Descripción de los casos de uso

Los diagramas de casos de uso brindan una información general y bastante sencilla sobre las interacciones entre funcionalidades, por ello es necesario abordar con más detalle cada una. Un ejemplo de esto sería una descripción textual de algunos aspectos claves estructurados en formato tabular o como un diagrama de secuencia [[19](#_ENREF_19)]. Dichas descripciones se presentan a continuación:

Tabla 2.1: Descripción del caso de uso clasificación de comentarios

|  |  |
| --- | --- |
| **Caso de uso** | Clasificación de comentarios |
| **Actores** | Sistema |
| **Descripción** | El caso de uso comienza cuando se carga en el sistema una colección de comentarios. Luego estos datos son pre-procesados e introducidos en el modelo clasificador que indica su grado de relevancia y su clasificación es decidida por un umbral. El caso de uso termina cuando todos los comentarios han sido clasificados. |
| **Casos de uso asociados** | * Preprocesar comentario <Include> * Predecir relevancia <Include> |
| **Precondiciones** | Debe existir al menos una colección de opiniones sobre la que realizar la predicción. |
| **Postcondiciones** | Debe obtener las predicciones de todos los comentarios en la colección |

Tabla 2.2: Descripción del caso de uso preprocesar comentario

|  |  |
| --- | --- |
| **Caso de uso** | preprocesar comentario |
| **Actores** | Sistema |
| **Descripción** | El caso de uso comienza cuando se han obtenido todos los comentarios a procesar. Los comentarios son procesados por tokenizador del modelo que los convierte en representaciones vectoriales digeribles por un ordenador además de emparejar sus tamaños. |
| **Casos de uso asociados** |  |
| **Precondiciones** | Debe existir al menos una colección de comentarios que preprocesar. |
| **Postcondiciones** | Comentarios tokenizados y listos para el modelo |

Tabla 2.3: Descripción del caso de uso predecir relevancia

|  |  |
| --- | --- |
| **Caso de Uso** | predecir relevancia |
| **Actores** | Sistema |
| **Descripción** | El caso de uso comienza cuando los comentarios preprocesados son suministrados al modelo clasificador. La red neuronal del modelo procesa las entradas y devuelve el grado de relevancia de cada comentario para luego decidir por un umbral de 0.5 si es relevante o no |
| **Casos de uso asociados** |  |
| **Precondiciones** | Debe recibir los comentarios tokenizados y de tamaño uniforme |
| **Postcondiciones** | Se predice la relevancia de todos los comentarios suministrados |

# Conclusiones parciales

* Fue propuesta una solución para la predicción de la relevancia de contenido textual basada en el Procesamiento de Lenguaje Natural, el aprendizaje profundo, el uso de modelos de lenguaje pre-entrenados de arquitectura Transformer y la inyección de conocimiento del dominio.
* Se propusieron 2 formas diferentes de crear un vector de características para introducir información de dominio en el modelo: el vector RC y el vector RP.
* Se propusieron 2 opciones como clasificador final que procese la salida del modelo junto a la información de dominio: la capa linear de clasificación y el perceptrón multicapa.
* Fueron identificados 3 casos de uso que corresponden a las funcionalidades de clasificación de comentarios, pre-procesar comentarios y predecir relevancia.

Capítulo 3: Validación y análisis de los resultados

En este capítulo se describe el estudio experimental llevado a cabo como forma de validación de la solución propuesta. Se presenta una descripción detallada de los datos utilizados, las métricas y la comparación con otros trabajos reportados en la literatura que hayan utilizado el mismo conjunto de datos.

# 3.1 Marco de evaluación

La solución propuesta fue evaluada en la tarea de predicción de relevancia de comentarios de usuarios sobre aplicaciones utilizando 4 sets de datos: Facebook, Swiftkey, Tapfish y Templerun2. Para realizar la evaluación se seleccionaron 5 modelos de lenguaje: RoBERTa, GPT-2, XLNet, ALBERT y BERTweet, todos en su tamaño base. Los modelos fueron entrenados utilizando datos en el idioma inglés.

Para cada modelo se calcularon las métricas de clasificación accuracy, precision, recall y f1-score a través de una validación cruzada de 5 pliegues (5-fold cross validation) para obtener el comportamiento medio de las métricas en cada set de datos. Para controlar la aleatoriedad de las divisiones de los pliegues se utilizó como semilla 123.

Los modelos fueron entrenados con un tamaño de lote igual a 16. Esta elección se tomó teniendo en cuenta que al entrenar con tamaños muy grandes de lotes aumenta el riesgo de que ocurra un sobreajuste a los datos den entrenamiento y los modelos no generalicen lo suficientemente bien al predecir a partir de nuevos datos.

Además, los modelos fueron entrenados durante 2 épocas, la cantidad utilizada en la investigación similar [[14](#_ENREF_14)]. Se hizo una prueba aislada con 4 épocas, pero no hubo diferencias significativas así que se mantuvo con 2 durante la validación cruzada debido a que las limitaciones de hardware hacían necesario evitar duplicar el tiempo de validación.

Posteriormente se seleccionaron los 2 modelos con mejores resultados para ser evaluados utilizando 4 variantes de introducción de conocimiento de dominio en ellos. Estas variantes son el resultado de combinar 2 formas de construir el vector de características del dominio: vector RC (basado en el conteo de términos relevantes) y vector RP (basado en las posiciones de los términos relevantes) con 2 formas de procesar la unión de dicho vector con la salida del modelo: capa linear de clasificación y preceptrón multicapa. Para más información sobre ellos refiérase a su sección correspondiente del capítulo 2 (2.1.3 para el vector y 2.1.4.1 para el clasificador final).

## 3.1.1 Modelos pre-entrenados seleccionados

Se consideraron diferentes modelos pre-entrenados de arquitectura Transformer en este trabajo, algunos considerados por otros autores y otros que no habían sido considerados para tratar este problema. Todos los modelos seleccionados se utilizaron en su versión base. Aunque ya habían sido mencionados anteriormente ahora se detallarán con mayor énfasis:

1. *RoBERTa* (*Robustly Optimized BERT Pretraining Approach*) [[71](#_ENREF_71)]: Basado en la arquitectura de BERT, modificando los pasos de pre-entrenamiento de *BERT* para mejorar sustancialmente el rendimiento en todas las tareas, sobre todo de clasificación. *RoBERTa* aumentó también la cantidad de tamaños de mini-batch, datos y tiempo de entrenamiento para entrenar el modelo. Además, se entrena en un conjunto de datos que incluye secuencias más largas que su predecesor, y el patrón de enmascaramiento también se modificó para que se generara espontáneamente. *RoBERTa* fue seleccionado debido a que superó a todos los modelos similares en su lanzamiento y todavía hoy sigue aportando resultados muy competitivos en muchas tareas de procesamiento de lenguaje natural.
2. *ALBERT* (*A Little BERT*) [[72](#_ENREF_72)]: como su nombre indica es un modelo basado en la arquitectura *BERT* con la particularidad de ser pequeño, producto a la reducción del tamaño de los parámetros. En su construcción se aplicaron tres técnicas de reducción de parámetros: Parametrización de incrustación factorizada (*Factorized embedding parameterization*), Compartición de parámetros entre capas (*Cross-layer parameter sharing*) y Pérdida de coherencia entre frases (*Inter-sentence coherence loss*). Se escogió este modelo ya que es capaz de conseguir tiempos de entrenamiento casi el doble de rápidos que *BERT* y *RoBERTa* utilizando cantidades de parámetros significativamente menores.
3. *BERTweet* [[73](#_ENREF_73)]: modelo que fue entrenado para procesar tweets (nombre por el que se refiere a los mensajes de la red social twitter, ahora X). Utiliza la misma arquitectura de *BERT* y el proceso de entrenamiento de *RoBERTa*. Su aporte se encuentra en el dataset utilizado para su pre-entrenamiento, 850 millones de tweets. Esto le permite tener un conocimiento muy especializado del lenguaje utilizado en la plataforma, haciéndolo ideal para análisis sobre comentarios hechos en *twitter*, en los cuales supera en desempeño tanto a *BERT* como a *RoBERTa*. Este modelo fue seleccionado porque ha superado a *RoBERTa* no solo en comentarios dentro de la red social mencionada, sino también en casos donde el texto analizado es de carácter informal debido a la similitud en el vocabulario utilizado. Los comentarios de usuarios sobre aplicaciones son en su mayoría texto informal, lo que convierte a este modelo en una buena opción a considerar.
4. *XLNet* [[74](#_ENREF_74)]: integra los modelos autorregresivos y el modelado bidireccional del contexto, superando las desventajas de *BERT*. Utiliza un mecanismo llamado Modelado de Permutación de Lenguaje (*Permutation Language Modeling* - *PLM*) con la idea de capturar el contexto bidireccional en todas las permutaciones de términos presentes en una secuencia de entrada. Se seleccionó porque ha superado a *RoBERTa* en algunas investigaciones y es un buen punto de referencia fuera de la familia de modelos con arquitectura *BERT*.
5. *GPT-2* [[75](#_ENREF_75)] es la última versión de la familia GPT disponible de forma pública. Fue entrenado en una enorme cantidad de texto de más de 8 millones de páginas web con un único objetivo, dada una oración predecir la próxima palabra. Este objetivo implica que cada token es analizado en función de únicamente la información obtenida de los tokens previos a él. Aunque obtuvo resultados al nivel del estado del arte del momento en que fue lanzado, hoy en día es superado en la mayoría de casos para tareas de clasificación por el resto de modelos mencionados. A pesar de esto, se seleccionó como punto comparativo para ilustrar el avance de los modelos pre-entrenados de lenguaje.

La obtención de los modelos de lenguaje se hace del repositorio de hugging face, que cuenta con prácticamente la totalidad de modelos de arquitectura Transformer de código abierto que existen. Los responsables de este repositorio también mantienen la biblioteca Transformers, de enorme utilidad para el trabajo con esta arquitectura de red neuronal.

# 3.2 Descripción de los sets de datos

La solución se evalúa en los 2 set de datos mencionados anteriormente, estos fueron tomados del estudio de [[10](#_ENREF_10)]. Los autores seleccionaron como casos de estudio 4 aplicaciones disponibles en la tienda de aplicaciones Google Play: Facebook, SwiftKey Keybord, Tap Fish y Temple Run 2. Se recopilaron las reseñas de usuarios sin procesar de estas aplicaciones de Google Play aproximadamente en el período de octubre de 2012 a febrero de 2013 y fueron etiquetadas [[10](#_ENREF_10)].

La siguiente tabla muestra una descripción de la composición de los sets de datos utilizados en cuanto a la cantidad de opiniones en cada uno, la cantidad de ejemplares de cada clasificación y el porciento de comentarios relevantes.

Tabla 3.1: Composición de los sets de datos utilizados

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Facebook** | **SwiftKey** | **TapFish** | **TempleRun2** |
| **Relevante** | 1662 | 879 | 888 | 933 |
| **No relevante** | 1338 | 2121 | 2112 | 2067 |
| **Porcentaje de relevantes** | 55.4% | 29.3% | 29,6% | 31,1% |
| **Total de comentarios** | 3000 | 3000 | 3000 | 3000 |

Como se puede apreciar en la tabla el set de datos de Facebook está balanceado pero el resto presenta un desbalance con una proporción cercana a (1:2) con la clase relevante como minoritaria, lo cual tiende a generar dificultades en el proceso de generalización dando prioridad a la clase mayoritaria en la predicción.

# 3.3 Resultados experimentales

## 3.3.1 Resultados experimentales sin uso de conocimiento del dominio

Tras realizar los experimentos con los modelos pre-entrenados sin la introducción de conocimiento del dominio se obtuvieron los siguientes resultados:

Tabla 3.2: Resultados sin conocimiento en el dataset de Facebook

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modelo** | ***Accuracy*** | ***Precision*** | ***Recall*** | ***F1-Score*** |
| *GPT 2 - base* | 0,9000 | 0,9009 | 0,8804 | 0,8846 |
| *XLNet - base* | 0,8467 | 0,8777 | 0,8099 | 0,8360 |
| *BERTweet - base* | 0,9377 | **0,9199** | 0,9400 | 0,9297 |
| *ALBERT - large* | 0,8983 | 0,8680 | 0,9041 | 0,8833 |
| ***RoBERTa - base*** | **0,9397** | 0,9101 | **0,9530** | **0,9306** |

Tabla 3.3: Resultados sin conocimiento en el dataset de SwiftKey

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modelo** | ***Accuracy*** | ***Precision*** | ***Recall*** | ***F1-Score*** |
| *GPT 2 - base* | 0,7487 | 0,9235 | 0,7696 | 0,8378 |
| *XLNet - base* | 0,8660 | 0,9449 | 0,8784 | 0,9093 |
| ***BERTweet - base*** | **0,9247** | **0,9736** | 0,9240 | **0,9481** |
| *ALBERT - large* | 0,8420 | 0,9127 | 0,8851 | 0,8929 |
| *RoBERTa - base* | 0,9217 | 0,9457 | **0,9454** | 0,9446 |

Tabla 3.4: Resultados sin conocimiento en el dataset de TapFish

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modelo** | ***Accuracy*** | ***Precision*** | ***Recall*** | ***F1-Score*** |
| *GPT 2 - base* | 0,9043 | 0,7742 | 0,8769 | 0,8166 |
| *XLNet - base* | 0,8863 | 0,9578 | 0,9000 | 0,9272 |
| ***BERTweet - base*** | **0,9413** | 0,9469 | 0,9750 | **0,9605** |
| *ALBERT - large* | 0,8110 | **0,9603** | 0,8242 | 0,8852 |
| *RoBERTa - base* | 0,9273 | 0,9224 | **0,9801** | 0,9496 |

Tabla 3.5: Resultados sin conocimiento en el dataset de TempleRun2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modelo** | ***Accuracy*** | ***Precision*** | ***Recall*** | ***F1-Score*** |
| *GPT 2 - base* | 0,8980 | 0,8338 | 0,8229 | 0,8244 |
| *XLNet - base* | 0,8360 | 0,8846 | 0,8894 | 0,8827 |
| ***BERTweet - base*** | **0,9320** | 0,9477 | **0,9560** | **0,9517** |
| *ALBERT - large* | 0,8140 | 0,9353 | 0,8411 | 0,8792 |
| *RoBERTa - base* | 0,9137 | **0,9581** | 0,9244 | 0,9403 |

Según se aprecia en los resultados, *BERTweet-base* es el MPTs que presenta el rendimiento más consistente en los cuatro contextos, destacándose en particular en *Tap Fish* y *Temple Run 2.* BERTweet-base alcanza valores de *F1-Scores* superiores a 0,930 en todos los contextos, destacándose *Tap Fish* como el mejor escenario (F1-Score de 0,960), pero en *Temple Run 2* y *SwiftKey* también se obtienen resultados muy sólidos, con un excelente balance entre precisión y recall. Este comportamiento de *BERTweet-base* se entiende también como robusto porque mantiene su rendimiento estable ante la presencia de alto desbalance en los *datasets*. *RoBERTa-base* también obtiene resultados muy competitivos, con altos valores de precisión y recall en todos los entornos. *RoBERTa-base* se mantiene con un rendimiento consistente en los cuatro contextos, obteniendo altos valores de *F1-Score*en todos, especialmente en *Tap Fish* (0,950) y *SwiftKey* (0,945), aunque tiene una ligera caída en *Temple Run 2* en términos de precisión (0,958). *ALBERT-large* tiene un rendimiento bastante equilibrado, pero no tan alto en comparación con los dos anteriores, lo que lo hace menos competitivo. *ALBERT-large* muestra un rendimiento ligeramente inferior en todos los contextos. Su mayor precisión se observa en *Tap Fish* (0,960), pero falla en el *recall* que no es tan alto, lo que afecta su *F1-Score*. Su desempeño es más limitado en *Temple Run 2* y *SwiftKey*.

*XLNet-base* tiene un rendimiento intermedio, ya que muestra un comportamiento fuerte en algunos casos como *Tap Fish*, pero débil en otros como Facebook. *XLNet-base* tiene un desempeño superior en *SwiftKey* y *Tap Fish*, alcanzando *F1-Scores* de 0,909 y 0,927, respectivamente. Sin embargo, en Facebooky *Temple Run 2*, su rendimiento disminuye en relación con otros modelos, especialmente en *recall* (0,810 en Facebook). Por último, *GPT2-base* muestra un rendimiento irregular, especialmente en *SwiftKey* y *Tap Fish*, donde tiene una precisión notablemente baja. GPT2-base tiene una disparidad notable entre los diferentes contextos. En *Facebook*, por ejemplo, obtiene resultados aceptables con un *F1-Score* de 0,885, pero su rendimiento disminuye en *SwiftKey* (F1-Score de 0,838) debido a un *recall* bajo (0,770). Su peor desempeño está en *Temple Run 2* y *Tap Fish* en términos de precisión, lo que indica que podría no ser ideal para contextos más exigentes, por ejemplo, en presencia de desbalance en el *dataset*. En este tipo de problema, la clasificación correcta de todas las opiniones “informativas” (medido a través del *recall*) tiene más relevancia que clasificar erróneamente algunas opiniones “no-informativa” (medido a través de *Precision*), ya que las opiniones “no-informativas” se descartan en el proceso de filtrado y se corre el riesgo de perder valiosos comentarios de los usuarios.

Tabla 3.6: Resultados generales sin conocimiento

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Modelos (F1-score)** | ***Facebook*** | ***SwiftKey*** | ***Tap Fish*** | ***Templerun2*** | **Promedio** |
| *GPT2 - base* | 0,885 | 0,838 | 0,817 | 0,824 | 0,841 |
| *ALBERT- large* | 0,883 | 0,893 | 0,885 | 0,879 | 0,885 |
| *XLNet - base* | 0,836 | 0,909 | 0,927 | 0,883 | 0,889 |
| *RoBERTa - base* | **0,931** | 0,945 | 0,950 | 0,940 | 0,942 |
| ***BERTweet - base*** | 0,930 | **0,948** | **0,960** | **0,952** | **0,948** |

El análisis realizado sobre el comportamiento de los modelos asociados a cada conjunto de datos queda corroborado por el comportamiento medio en función de la métrica *F1-score* mostrado en la Tabla 3.6. Como se puede observar en la tabla, los modelos *RoBERTa* y *BERTweet* obtienen los mejores resultados con diferencia entre los modelos evaluados, siendo los únicos con valores superiores a 0,9 en todos los sets de datos probados, siendo *BERTweet* ligeramente mejor. Existe una diferencia significativa en el valor medio entre el 2º mejor (*RoBERTa* con 0,942) y el 3º mejor (*XLNet* con 0,889) lo que confirma la superioridad de los 2 mejores sobre los demás estudiados.

## 3.3.2 Resultados experimentales utilizando conocimiento del dominio

Se concluyó en los experimentos anteriores que los modelos pre-entrenados que mejores resultados aportaban en la clasificación de opiniones de usuarios para el soporte de software entre los evaluados son RoBERTa-base y BERTweet-base. A raíz de esto se seleccionaron ambos modelos para ser evaluados aplicándo las técnicas de incorporación de conocimiento de dominio mencionadas en la sección 2.1.5. Estas técnicas dan lugar a 4 variantes de solución con conocimiento del dominio:

* *Linear + RC*: Uso de una capa linear como clasificador final y del vector de características RC basado en el conteo de términos relevantes.
* *Linear + RP*: Uso de una capa linear como clasificador final y del vector de características RP basado en las posiciones de los términos relevantes.
* *MLP + RC*: Uso de un perceptrón multicapa como clasificador final y del vector de características RC basado en el conteo de términos relevantes.
* *MLP + RP*: Uso de un perceptrón multicapa como clasificador final y del vector de características RP basado en las posiciones de los términos relevantes.

Cabe recordar que las siglas MLP hacen referencia *Multilayer Perceptron*, el inglés de perceptrón multicapa y RC y RP son *Relevant Count* y *Relevant Position* respectivamente.

Cada una de estas variantes fueron evaluadas y comparadas con los resultados originales del modelo. Por cuestiones del tiempo disponible para la experimentación solo se mostrarán y analizarán los resultados correspondientes al set de datos de *Facebook.*

En las tablas 3.7 y 3.8 los resultados de cada en técnica en cada modelo se comparan contra los resultados originales del modelo. Las celdas verdes indican una mejora en la métrica correspondiente respecto al original y las naranjas una disminución.

Tabla 3.7: Resultados comparativos con conocimiento de RoBERTa en Facebook

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Variante de solución | *Accuracy* | *Precision* | *Recall* | *F1-Score* |
| *RoBERTa* | 0,940 | 0,910 | **0,953** | 0,931 |
| ***RoBERTa + Linear + RC*** | **0,941** | 0,917 | 0,950 | **0,933** |
| *RoBERTa + Linear + RP* | 0,930 | 0,902 | 0,943 | 0,920 |
| *RoBERTa + MLP + RP* | 0,932 | 0,924 | 0,925 | 0,924 |
| *RoBERTa + MLP + RC* | 0,931 | **0,941** | 0,910 | 0,924 |

Tabla 3.8: Resultados comparativos con conocimiento de BERTweet en Facebook

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Variante de solución | *Accuracy* | *Precision* | *Recall* | *F1-Score* |
| ***BERTweet*** | **0,938** | 0,920 | **0,940** | **0,930** |
| *BERTweet + Linear + RC* | 0,935 | **0,937** | 0,917 | 0,927 |
| *BERTweet + Linear + RP* | 0,933 | 0,933 | 0,921 | 0,925 |
| *BERTweet + MLP + RP* | 0,923 | 0,901 | 0,928 | 0,912 |
| *BERTweet + MLP + RC* | 0,934 | 0,933 | 0,921 | 0,927 |

En el caso de *RoBERTa* se puede apreciar en la tabla 3.7 que solo la variante de combinar la capa linear con el vector RC basado en el conteo de términos relevantes logra un aumento estable del desempeño del modelo en comparación a los resultados originales sin uso del conocimiento en la mayoría de métricas solo disminuyendo un poco *recall*. Esta solución se lleva, además, los mejores resultados de *accuracy* y *F1-score,* mientras que el valor más alto de *precision* se alcanza con la combinación del mismo vector RC con el perceptrón multicapa*.*

Para *BERTweet* en la tabla 3.8 la situación es ligeramente diferente a *RoBERTa*, ya que ninguna de las variantes ofrece una mejora completa sobre el modelo original. Aunque, se puede destacar en el caso de la variante utilizando una capa linear junto al vector RC que obtiene una mejora significativa de *precision* (0,920 a 0,937) a cambio perder en igual medida *recall* (0,940 a 0,917) y una muy ligera disminución en *accuracy* (0,938 a 0,935) y *F1-Score* (0,930 a 0,927).

De forma general, en las variantes con conocimiento de dominio para ambos modelos evaluados se puede observar una tendencia muy marcada (6 de 8) al aumento del valor de la métrica precision y la disminución del *recall* independientemente de si en el resto de métricas subió o bajó el valor. Incluso, es en estas 2 métricas donde se puede ver con mayor intensidad el impacto del uso de este tipo de técnicas. Por ejemplo, en RoBERTa el uso del MLP con el vector RC provocó un aumento de *precision* de 0,031 (0,910 a 0,941) y una disminución de 0,043 (0,953 a 0,910) en *recall*. Esta tendencia abre algunas opciones a tener en cuenta, ya que la priorización de *precision* sobre *recall* con la aplicación de técnicas de conocimiento del dominio implica que el modelo está dando mayor prioridad a minimizar la cantidad de falsos positivos (opiniones no-informativas clasificadas como informativas incorrectamente), en otras palabras, está siendo más estricto a la hora de dejar pasar las opiniones por el filtro de si es informativa la opinión. Esto implica que es posible que se pierdan algunas opiniones informativas, pero a su vez menos opiniones no informativas son consideradas por error. Por consiguiente, este enfoque está muy alineado a la idea de filtrar la gran cantidad de opiniones no informativas que hay en la red priorizando descartar las no informativas. Además, si el modelo duda tanto al clasificar una opinión informativa que la termina clasificando como no informativa debido a esta pequeña diferencia en el enfoque es muy posible que dicha opinión contenga poca información en primer lugar. Es una cuestión del que vaya a usar un modelo de esta índole decidir qué va a priorizar, ya que, aunque a veces el uso de estas técnicas provoca un ligero aumento estable en los resultados lo que sí está más garantizado es que casi siempre va a provocar el cambio en el enfoque antes explicado y son las necesidades del equipo de software que lo vaya a implementar quienes tendrán la última palabra.

Tabla 3.9: Resultados generales con conocimiento en Facebook

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variante de solución** | ***Accuracy*** | ***Precision*** | ***Recall*** | ***F1-Score*** |
| *RoBERTa* | 0,940 | 0,910 | **0,953** | 0,931 |
| ***RoBERTa + Linear + RC*** | **0,941** | 0,917 | 0,950 | **0,933** |
| *RoBERTa + Linear + RP* | 0,930 | 0,902 | 0,943 | 0,920 |
| *RoBERTa + MLP + RP* | 0,932 | 0,924 | 0,925 | 0,924 |
| *RoBERTa + MLP + RC* | 0,931 | **0,941** | 0,910 | 0,924 |
| *BERTweet* | 0,938 | 0,920 | 0,940 | 0,930 |
| *BERTweet + Linear + RC* | 0,935 | 0,937 | 0,917 | 0,927 |
| *BERTweet + Linear + RP* | 0,933 | 0,933 | 0,921 | 0,925 |
| *BERTweet + MLP + RP* | 0,923 | 0,901 | 0,928 | 0,912 |
| *BERTweet + MLP + RC* | 0,934 | 0,933 | 0,921 | 0,927 |

Al comparar todos los resultados obtenidos la solución que destaca como mejor es la de usar RoBERTa-base con una capa lineal como clasificador final y el vector de características basado en el conteo de términos relevantes (RoBERTa + Linear + RC) ya que obtiene los mejores valores de accuracy (0,941) y F1-score (0,933).

## 3.3.3 Comparación con otros resultados en la literatura

A continuación, se compara la mejor solución obtenida en este trabajo (*RoBERTa-base + Linear + RC*) con otras soluciones aportadas en la literatura de clasificación de opiniones para el soporte de software en el set de datos de Facebook. Para la comparación se utilizaron los valores de *F1-Score* obtenidos por cada una de las soluciones.

Tabla 3.10: Comparación con otras soluciones de la literatura

|  |  |
| --- | --- |
| **Soluciones reportadas (*Facebook*)** | ***F1-Score*** |
| *AR-MINER* [Chen, et al. [10]](#_ENREF_10) | 0,877 |
| *Hybrid ML* [Martín [13]](#_ENREF_13) | 0,907 |
| *StackOBERTflow* [Prenner and Robbes [56]](#_ENREF_56) | 0,909 |
| *BERT-base* [Prenner and Robbes [56]](#_ENREF_56) | 0,906 |
| *BERT-SO-1M* [Prenner and Robbes [56]](#_ENREF_56) | 0,921 |
| ***BERT-reviews*** [Prenner and Robbes [56]](#_ENREF_56) | **0,933** |
| ***RoBERTa-base + Linear + RC* (propuesta)** | **0,933** |

En la tabla 3.10 se puede ver como la solución propuesta en este trabajo iguala en resultados a la mejor solución presente en la literatura para este set de datos (BERT-reviews) teniendo ambas un F1-Score de 0,933. De esta forma se puede ver que el uso de conocimiento del dominio puede tener un buen impacto en el desempeño del modelo, específicamente el uso de técnicas basadas en el uso de vectores de características que son las evaluadas en este trabajo. Sin embargo, este aumento no es tan significativo muchas veces y su verdadero valor quizás está en el enfoque que prioriza filtrar cuantas opiniones no-informativas sea posible, aunque se pierdan algunas pocas informativas por el camino, tal como se explicó en la sección anterior.

# 3.4 Conclusiones parciales

* De los 5 modelos pre-entrenados de arquitectura evaluados sin conocimiento para la tarea de clasificación de opiniones para soporte de software los mejores resultados fueron obtenidos por *RoBERTa-base* y *BERTweet-base.*
* Se evaluaron 4 variantes de solución para introducir conocimiento del dominio en cada uno los 2 modelos seleccionados como mejores para el problema (8 en total).
* De las variantes de solución evaluadas solo la combinación de la capa linear como clasificador final y el vector RC basado en la cantidad de términos relevantes obtuvieron una mejora estable sobre su modelo original en el caso del modelo *RoBERTa-base*. También se coronó como la mejor solución en general para el set de datos estudiado.
* La solución propuesta con *RoBERTa-base + Linear + RC* (la mencionada en el punto anterior) superó a casi todos los demás resultados mostrados en la literatura y empató en desempeño con el mejor de ellos.
* Existe una tendencia en la aplicación de este tipo de técnicas a incrementar en la mayoría de los casos el valor de precision manteniendo una leve pérdida en el resto de métricas en el caso de no mejorarlas. Esto abre una puerta a la posibilidad de, en caso de que así se quiera, dar prioridad en el filtrado a aceptar la menor cantidad posible de opiniones no informativas, aunque se pierdan algunas pocas informativas en el proceso, con el objetivo de reducir en mayor medida la cantidad de opiniones resultantes luego del proceso de filtrado.

Conclusiones

Con la culminación de esta investigación se pudo llegar a las siguientes conclusiones:

* El análisis de los trabajos reportados en la literatura demostró que hay muy pocos trabajos en la literatura que hacen uso de modelos Transformer y aún menos que utilizan conocimiento del dominio. Siendo la única técnica de introducción de conocimiento del dominio el pre-entrenamiento adicional que es extremadamente costoso en términos de tiempo y recursos.
* Fue viable implementar una técnica que aplica vectores de características a partir de un glosario de términos relevantes para el desarrollo de software como apoyo auxiliar a los modelos Transformers.
* Se determinaron 2 formas de construir el vector de características, una basada en la cantidad de términos relevantes en la opinión (denominada vector RC) y otra basada en las posiciones de los términos relevantes en la opinión (denominada vector RC). Estas en combinación con 2 enfoques distintos para la capa de clasificación final (Linear o perceptrón multicapa) dieron lugar a 4 variantes de solución que se le pueden aplicar a un modelo pre-entrenado Transformer para introducirle conocimiento del dominio.
* Los modelos pre-entrenados que mejores resultados obtuvieron sin el uso de las técnicas de conocimiento del dominio de entre los evaluados fueron *RoBERTa-base* y *BERTweet-base.*
* De las variantes de solución evaluadas solo la combinación de la capa linear como clasificador final y el vector RC basado en la cantidad de términos relevantes obtuvieron una mejora estable sobre su modelo original en el caso del modelo *RoBERTa-base*. También se coronó como la mejor solución en general para el set de datos estudiado.
* La solución propuesta con *RoBERTa-base + Linear + RC* (la mencionada en el punto anterior) superó a casi todos los demás resultados mostrados en la literatura y empató en desempeño con el mejor de ellos.
* Existe una tendencia en la aplicación de este tipo de técnicas a incrementar en la mayoría de los casos el valor de precision manteniendo una leve pérdida en el resto de métricas en el caso de no mejorarlas. Esto abre una puerta a la posibilidad de, en caso de que así se quiera, dar prioridad en el filtrado a aceptar la menor cantidad posible de opiniones no informativas, aunque se pierdan algunas pocas informativas en el proceso, con el objetivo de reducir en mayor medida la cantidad de opiniones resultantes luego del proceso de filtrado.

Recomendaciones

Como continuidad a la investigación desarrollada se proponen las siguientes recomendaciones:

* Evaluar las variantes de solución propuestas en otros sets de datos, lo cual no se llevó a cabo en este trabajo por falta de tiempo.
* Explorar el uso de otras técnicas de introducción de conocimiento del dominio como la inyección directa de texto o la modificación de *embeddings.*
* Buscar y probar con otras fuentes de conocimiento además del glosario ya estudiado, como por ejemplo grafos de conocimiento.
* Evaluar la solución utilizando otros hiperparámetros de entrenamiento en busca de los óptimos.
* Explorar la posibilidad de llevar a cabo un análisis similar con opiniones en lenguaje español.

Bibliografía

[1] P. A. Laplante and M. H. Kassab, F. Edition, Ed. *Requirements Engineering for Software and Systems*. CRC Press, 2022.

[2] D. M. Fernández *et al.*, "Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice," *Empirical software engineering,* vol. 22, pp. 2298-2338, 2017.

[3] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe, "Toward data-driven requirements engineering," *IEEE software,* vol. 33, no. 1, pp. 48-54, 2015.

[4] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *2013 21st IEEE international requirements engineering conference (RE)*, 2013, pp. 125-134: IEEE.

[5] H. H. Khan, M. N. Malik, Y. Alotaibi, A. Alsufyani, and S. Alghamdi, "Crowdsourced Requirements Engineering Challenges and Solutions: A Software Industry Perspective," *Computer Systems Science & Engineering,* vol. 39, no. 2, 2021.

[6] E. Guzman, R. Alkadhi, and N. Seyff, "A needle in a haystack: What do twitter users say about software?," in *2016 IEEE 24th international requirements engineering conference (RE)*, 2016, pp. 96-105: IEEE.

[7] S. U. Hassan, J. Ahamed, and K. Ahmad, "Analytics of machine learning-based algorithms for text classification," *Sustainable Operations and Computers,* vol. 3, pp. 238-248, 2022.

[8] A. Al-Hawari, H. Najadat, and R. Shatnawi, "Classification of application reviews into software maintenance tasks using data mining techniques," *Software Quality Journal,* vol. 29, pp. 667-703, 2021.

[9] N. Aslam, W. Y. Ramay, K. Xia, and N. Sarwar, "Convolutional neural network based classification of app reviews," *IEEE Access,* vol. 8, pp. 185619-185628, 2020.

[10] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang, "AR-miner: mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 767-778.

[11] V. Milián Núñez, T. Blanco Martín, A. Simón-Cuevas, H. González Diéz, and A. Hernández González, "A Knowledge-Based User Feedback Classification Approach for Software Support," in *International Workshop on Artificial Intelligence and Pattern Recognition*, 2023, pp. 237-247: Springer.

[12] V. Milián Núñez, H. González Diéz , and A. Simón Cuevas, "Predicción de Requisitos de Software a partir de Opiniones de Usuarios," in *“Transferencia de Conocimiento en Tecnologías de la Información” (STCTI). IV Convención Científica Internacional UCLV 2023*, Santa Clara, 2021.

[13] T. B. Martín, "Método de predicción de la relevancia en opiniones de usuarios para el soporte de software," Universidad Tecnológica de La Habana “José Antonio Echeverría”, 2022.

[14] P. R. Henao, J. Fischbach, D. Spies, J. Frattini, and A. Vogelsang, "Transfer learning for mining feature requests and bug reports from tweets and app store reviews," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, 2021, pp. 80-86: IEEE.

[15] M. A. Hadi and F. H. Fard, "Evaluating pre-trained models for user feedback analysis in software engineering: A study on classification of app-reviews," *Empirical Software Engineering,* vol. 28, no. 4, p. 88, 2023.

[16] R. R. Mekala, A. Irfan, E. C. Groen, A. Porter, and M. Lindvall, "Classifying user requirements from online feedback in small dataset environments using deep learning," in *2021 IEEE 29th International requirements engineering conference (RE)*, 2021, pp. 139-149: IEEE.

[17] T. Dash, S. Chitlangia, A. Ahuja, and A. Srinivasan, "A review of some techniques for inclusion of domain-knowledge into deep neural networks," *Scientific Reports,* vol. 12, no. 1, p. 1040, 2022.

[18] L. Hu, Z. Liu, Z. Zhao, L. Hou, L. Nie, and J. Li, "A survey of knowledge enhanced pre-trained language models," *IEEE Transactions on Knowledge and Data Engineering,* 2023.

[19] I. Sommerville, *Software engineering*, 10th ed. Pearson, 2016.

[20] R. Kasauli, E. Knauss, J. Horkoff, G. Liebel, and F. G. de Oliveira Neto, "Requirements engineering challenges and practices in large-scale agile system development," *Journal of Systems and Software,* vol. 172, p. 110851, 2021.

[21] A. Gasparetto, M. Marcuzzo, A. Zangari, and A. Albarelli, "A survey on text classification algorithms: From text to predictions," *Information,* vol. 13, no. 2, p. 83, 2022.

[22] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information,* vol. 10, no. 4, p. 150, 2019.

[23] L. Hickman, S. Thapa, L. Tay, M. Cao, and P. Srinivasan, "Text preprocessing for text mining in organizational research: Review and recommendations," *Organizational Research Methods,* vol. 25, no. 1, pp. 114-146, 2022.

[24] S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing A Comprehensive Guide to Building Real-World NLP Systems*, 1st ed. O´Reilly, 2020.

[25] A. Kedia and M. Rasu, *Hands-On Python Natural Language Processing*. Packt Publishing, 2020.

[26] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, *An Introduction to Statistical Learning with Aplication in Python*. Springer Texts in Statistics, 2023.

[27] S. Fransiska, R. Rianto, and A. I. Gufroni, "Sentiment Analysis Provider by. U on Google Play Store Reviews with TF-IDF and Support Vector Machine (SVM) Method," *Scientific Journal of Informatics,* vol. 7, no. 2, pp. 203-212, 2020.

[28] S. M. Sarsam, H. Al-Samarraie, A. I. Alzahrani, W. Alnumay, and A. P. Smith, "A lexicon-based approach to detecting suicide-related messages on Twitter," *Biomedical Signal Processing and Control,* vol. 65, p. 102355, 2021.

[29] V. Vangara, S. Vangara, and K. Thirupathur, "Opinion mining classification using naive bayes algorithm," *International Journal of Innovative Technology and Exploring Engineering (IJITEE),* vol. 9, no. 5, pp. 495-498, 2020.

[30] M. T. Akter, M. Begum, and R. Mustafa, "Bengali sentiment analysis of e-commerce product reviews using k-nearest neighbors," in *2021 International conference on information and communication technology for sustainable development (ICICT4SD)*, 2021, pp. 40-44: IEEE.

[31] F. Chollet, *Deep Learning with Python*, 2nd Edition ed. Manning Publications Co., 2021.

[32] I. Lauriola, A. Lavelli, and F. Aiolli, "An introduction to deep learning in natural language processing: Models, techniques, and tools," *Neurocomputing,* vol. 470, pp. 443-456, 2022.

[33] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electronic Markets,* vol. 31, no. 3, pp. 685-695, 2021.

[34] M. J. Zaki and J. Wagner Meira, *Data Mining and Machine Learning Fundamental Concepts and Algorithms*, 2nd ed. Cambridge University Press, 2020.

[35] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Physica D: Nonlinear Phenomena,* vol. 404, p. 132306, 2020.

[36] M. A. Rahman, M. A. Haque, M. N. A. Tawhid, and M. S. Siddik, "Classifying non-functional requirements using RNN variants for quality software development," in *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, 2019, pp. 25-30.

[37] J. Á. González Barba, "Aprendizaje profundo para el procesamiento del lenguaje natural," 2017.

[38] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems,* vol. 30, 2017.

[39] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems,* vol. 26, 2013.

[40] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532-1543.

[41] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics,* vol. 5, pp. 135-146, 2017.

[42] Y. Guo, X. Dong, M. A. Al-Garadi, A. Sarker, C. Paris, and D. M. Aliod, "Benchmarking of transformer-based pre-trained models on social media text classification datasets," in *Proceedings of the the 18th annual workshop of the australasian language technology association*, 2020, pp. 86-91.

[43] H. M. Zahera and M. A. Sherif, "ProBERT: Product Data Classification with Fine-tuning BERT Model," in *MWPD@ISWC*, 2020.

[44] F. Ruggeri, M. Lippi, and P. Torroni, "Combining Transformers with Natural Language Explanations," *arXiv preprint arXiv:2110.00125,* 2021.

[45] Z. Li, X. Ding, K. Liao, B. Qin, and T. Liu, "Causalbert: Injecting causal knowledge into pre-trained models with minimal supervision," *arXiv preprint arXiv:2107.09852,* 2021.

[46] A. Lauscher, O. Majewska, L. F. Ribeiro, I. Gurevych, N. Rozanov, and G. Glavaš, "Common sense or world knowledge? investigating adapter-based knowledge injection into pretrained transformers," *arXiv preprint arXiv:2005.11787,* 2020.

[47] J. Bai *et al.*, "Syntax-BERT: Improving pre-trained transformers with syntax trees," *arXiv preprint arXiv:2103.04350,* 2021.

[48] L. Cai, J. Li, H. Lv, W. Liu, H. Niu, and Z. Wang, "Integrating domain knowledge for biomedical text analysis into deep learning: A survey," *Journal of Biomedical Informatics,* vol. 143, p. 104418, 2023.

[49] Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun, and Q. Liu, "ERNIE: Enhanced language representation with informative entities," *arXiv preprint arXiv:1905.07129,* 2019.

[50] M. Saeed, N. Ahmadi, P. Nakov, and P. Papotti, "RuleBERT: Teaching soft rules to pre-trained language models," *arXiv preprint arXiv:2109.13006,* 2021.

[51] A. Cadeddu *et al.*, "A comparative analysis of knowledge injection strategies for large language models in the scholarly domain," *Engineering Applications of Artificial Intelligence,* vol. 133, p. 108166, 2024.

[52] M. Ostendorff, P. Bourgonje, M. Berger, J. Moreno-Schneider, G. Rehm, and B. Gipp, "Enriching bert with knowledge graph embeddings for document classification," *arXiv preprint arXiv:1909.08402,* 2019.

[53] R. Yan, L. Sun, F. Wang, and X. Zhang, "A general method for transferring explicit knowledge into language model pretraining," *Security and Communication Networks,* vol. 2021, no. 1, p. 7115167, 2021.

[54] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems,* vol. 33, pp. 9459-9474, 2020.

[55] B. Sarmah, D. Mehta, B. Hall, R. Rao, S. Patel, and S. Pasquali, "Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction," in *Proceedings of the 5th ACM International Conference on AI in Finance*, 2024, pp. 608-616.

[56] J. A. Prenner and R. Robbes, "Making the most of small Software Engineering datasets with modern machine learning," *IEEE Transactions on Software Engineering,* vol. 48, no. 12, pp. 5050-5067, 2021.

[57] J. Lee *et al.*, "BioBERT: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics,* vol. 36, no. 4, pp. 1234-1240, 2020.

[58] Q. Jin, B. Dhingra, W. W. Cohen, and X. Lu, "Probing biomedical embeddings from language models," *arXiv preprint arXiv:1904.02181,* 2019.

[59] Y. Peng, S. Yan, and Z. Lu, "Transfer learning in biomedical natural language processing: an evaluation of BERT and ELMo on ten benchmarking datasets," *arXiv preprint arXiv:1906.05474,* 2019.

[60] S. Xu *et al.*, "K-plug: Knowledge-injected pre-trained language model for natural language understanding and generation in e-commerce," *arXiv preprint arXiv:2104.06960,* 2021.

[61] M. Bramer, *Principles of Data Mining*. Springer-Verlang London, 2007.

[62] (10 de febrero del 2024). *Python web*. Available: <https://www.python.org/about/>

[63] T. Wolf *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771,* 2019.

[64] (25 de febrero de 2024). *PyTorch web*. Available: <https://pytorch.org/features/>

[65] (25 de febrero de 2024). *PyTorch Lightning web*. Available: <https://lightning.ai/pytorch-lightning>

[66] (25 de febrero de 2024). *TorchMetrics web*. Available: <https://lightning.ai/docs/torchmetrics/stable/>

[67] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, 2010, vol. 445, no. 1, pp. 51-56: Austin, TX.

[68] (10 de febrero del 2024). *NLTK web*. Available: <https://www.nltk.org/>

[69] (10 de febrero del 2024). *NumPy web*. Available: <https://numpy.org/about/>

[70] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *the Journal of machine Learning research,* vol. 12, pp. 2825-2830, 2011.

[71] Y. Liu, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692,* vol. 364, 2019.

[72] Z. Lan, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942,* 2019.

[73] D. Q. Nguyen, T. Vu, and A. T. Nguyen, "BERTweet: A pre-trained language model for English Tweets," *arXiv preprint arXiv:2005.10200,* 2020.

[74] Z. Yang, "XLNet: Generalized Autoregressive Pretraining for Language Understanding," *arXiv preprint arXiv:1906.08237,* 2019.

[75] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog,* vol. 1, no. 8, p. 9, 2019.