

Universidad Tecnológica de la Habana

“José Antonio Echeverría”

CUJAE

|  |
| --- |
| Clasificación de opiniones según relevancia para soporte de software |

Informe de Prácticas Profesionales 1

Autores:

Ray Maestre Peña

Alejandro Montes Hernández

Tutor:

Dr. C. Alfredo Simón Cuevas

La Habana, febrero de 2024

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 clasificación de opiniones de usuarios según su relevancia para un equipo de software, con el objetivo de filtrar aquellas opiniones que no son de utilidad y facilitar su posterior análisis. Esta solución está basada en el uso de modelos de lenguajes de arquitectura transformer, específicamente RoBERTa y GPT 2. Los modelos fueron evaluados mediante experimentos utilizando una validación cruzada de 5 pliegues en 2 sets de datos que contienen miles de opiniones recopiladas sobre las aplicaciones Facebook y SwiftKey respectivamente. Los resultados obtenidos fueron favorables, aunque la solución con GPT 2 no superó todos los resultados anteriores que utilizaban técnicas de aprendizaje automático tradicionales, la solución utilizando RoBERTa sí superó en todos los casos los resultados anteriores reportados en la literatura.

**Palabras Clave:** relevancia de opiniones, clasificación de texto, modelos de lenguaje, RoBERTa, GPT 2.

Abstract

The success of a software system can be measured by how useful it is to its users. Therefore, it becomes necessary for development teams to get good feedback from their users to identify their interests and needs. With the popularization of platforms such as social networks or app stores, many ways to get feedback from users regarding a given software have emerged. However, the quantity is such that their manual analysis becomes unaffordable and even the vast majority of these opinions do not provide useful information, they are irrelevant. This work proposes the development of an algorithm to classify user opinions according to their relevance for a software team, with the aim of filtering out those opinions that are not useful and facilitating their subsequent analysis. This solution is based on the use of transformer architecture language models, specifically RoBERTa and GPT 2. The models were evaluated through experiments using a 5-fold cross-validation on two datasets containing thousands of opinions collected on Facebook and SwiftKey applications respectively. The achieved results were favorable, although the solution using GPT 2 did not outperform all previous results using traditional machine learning techniques, the solution using RoBERTa did outperform in all cases the previous results reported in the literature.

**Keywords:** opinion relevance, text classification, language models, RoBERTa, GPT 2.

Índice

[Capítulo 1: Fundamentación teórica 5](#_Toc160367450)

[1.1 Ingeniería de requisitos basada en datos 5](#_Toc160367451)

[1.2 Clasificación de textos 6](#_Toc160367452)

[1.2.1 Pre-procesamiento 7](#_Toc160367453)

[1.2.2 Representaciones de texto 8](#_Toc160367454)

[1.2.3 Aprendizaje de máquina 9](#_Toc160367455)

[1.2.4 Enfoques tradicionales de solución 10](#_Toc160367456)

[1.3 Clasificación de textos con aprendizaje profundo 11](#_Toc160367457)

[1.3.1 Redes Neuronales 11](#_Toc160367458)

[1.3.1.1 Arquitecturas más utilizadas 14](#_Toc160367459)

[1.3.2 Representaciones distribuidas 15](#_Toc160367460)

[1.3.3 Transferencia de aprendizaje 16](#_Toc160367461)

[1.3.3.1 Fine-tuning de modelos Transformers para clasificación 16](#_Toc160367462)

[1.4 Antecedentes 18](#_Toc160367463)

[1.5 Métodos de evaluación 19](#_Toc160367464)

[1.5.1 Métricas 20](#_Toc160367465)

[1.6 Herramientas utilizadas 22](#_Toc160367466)

[1.6.1 Lenguaje de programación: Python 22](#_Toc160367467)

[1.6.2 Bibliotecas principales 23](#_Toc160367468)

[1.7 Conclusiones parciales 25](#_Toc160367469)

[Capítulo 2: Método de solución planteado 27](#_Toc160367470)

[2.1 Descripción de la solución propuesta 27](#_Toc160367471)

[2.1.1 Descripción general 27](#_Toc160367472)

[2.1.2 Pre-procesamiento 28](#_Toc160367473)

[2.1.3 Modelo de clasificación 29](#_Toc160367474)

[2.2 Desarrollo del clasificador de opiniones según su relevancia para el desarrollo de software 30](#_Toc160367475)

[2.2.1 Modelo de dominio 30](#_Toc160367476)

[2.2.2 Captura de requisitos 32](#_Toc160367477)

[2.2.2.1 Diagrama de casos de uso del sistema 32](#_Toc160367478)

[2.2.2.2 Descripción de los casos de uso 32](#_Toc160367479)

[2.3 Conclusiones parciales 35](#_Toc160367480)

[Capítulo 3: Validación y análisis de los resultados 36](#_Toc160367481)

[3.1 Marco de evaluación 36](#_Toc160367482)

[3.2 Descripción de los sets de datos 37](#_Toc160367483)

[3.3 Resultados experimentales 38](#_Toc160367484)

[3.3.1 Resultados experimentales para el set de datos Facebook 38](#_Toc160367485)

[3.3.2 Resultados experimentales para el set de datos Swiftkey 39](#_Toc160367486)

[3.3.3 Análisis general de los resultados experimentales 40](#_Toc160367487)

[3.3.4 Comparación con otros resultados en la literatura 41](#_Toc160367488)

[3.4 Conclusiones parciales 42](#_Toc160367489)

[Conclusiones 43](#_Toc160367490)

[Recomendaciones 44](#_Toc160367491)

[Bibliografía 45](#_Toc160367492)

Índice de Figuras

[Figura 1.1: Estructura general una red neuronal [18] 12](#_Toc160393993)

[Figura 1.2: Esquema del proceso de entrenamiento de una red neuronal [24] 13](#_Toc160393994)

[Figura 1.3: Estructura común de un modelo fine-tuned para una tarea de clasificación 18](#_Toc160393995)

[Figura 2.1: Esquema del flujo de trabajo de la aplicación 27](#_Toc160393996)

[Figura 2.2: Arquitectura del modelo de clasificación planteado 29](#_Toc160393997)

[Figura 2.3 Modelo de dominio de la solución 31](#_Toc160393998)

[Figura 2.4: Diagrama de casos de uso del sistema 32](#_Toc160393999)

[Figura 3.1: resultados utilizando roBERTa en ambos sets de datos 40](#_Toc160394000)

[Figura 3.2: resultados utilizando roBERTa en ambos sets de datos 41](#_Toc160394001)

Índice de Tablas

[*Tabla 1.1 Matriz de confusión para una clasificación específica* 20](#_Toc160394400)

[*Tabla 2.1: Descripción del caso de uso clasificación de comentarios* 33](#_Toc160394401)

[*Tabla 2.2: Descripción del caso de uso preprocesar comentario* 34](#_Toc160394402)

[*Tabla 2.3: Descripción del caso de uso predecir relevancia* 35](#_Toc160394403)

[*Tabla 3.1: Composición de los sets de datos utilizados* 37](#_Toc160394404)

[*Tabla 3.2: Resultado de la validación cruzada con roBERTa en el set de datos Facebook* 38](#_Toc160394405)

[*Tabla 3.3: Resultado de la validación cruzada con GPT 2 en el set de datos Facebook* 38](#_Toc160394406)

[*Tabla 3.4: Comparación de ambos modelos en el set de datos Facebook* 38](#_Toc160394407)

[*Tabla 3.5: Resultado de la validación cruzada con roBERTa en el set de datos SwiftKey* 39](#_Toc160394408)

[*Tabla 3.6: Resultado de la validación cruzada con GPT 2 en el set de datos SwiftKey* 39](#_Toc160394409)

[*Tabla 3.7: Comparación de ambos modelos en el set de datos SwiftKey* 40](#_Toc160394410)

[*Tabla 3.8 Comparación de la solución propuesta con otras soluciones de la literatura* 42](#_Toc160394411)

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, [[10-12](#_ENREF_10)] han hecho un enfoque especial en la detección de relevancia en las opiniones para reducir considerablemente el número de opiniones a analizar. Sin embargo, este proceso de detección de relevancia no ha sido tratado en la literatura mediante el uso de técnicas más recientes de aprendizaje profundo. En especial, el uso de modelos de lenguaje de arquitectura Transformer como GPT 2, BERT y RoBERTa, que han demostrado un desempeño al nivel del estado del arte en la mayoría de campos en los que se han implementado, 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 irrelevantes.
* La existencia de modelos de lenguaje con el potencial de mejorar los resultados actuales en este campo

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

¿Puede el uso de modelos de lenguaje de arquitectura transformer mejorar la eficacia del proceso de detección de relevancia en 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 clasificación de opiniones de usuarios según su relevancia para un equipo de desarrollo de software mediante el uso de 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 los distintos algoritmos de clasificación tradicionales basados en aprendizaje automático. (Ray y Alejandro)
  2. Investigar las técnicas de clasificación de texto utilizando aprendizaje profundo, sobre todo de arquitectura Transformer. (Ray y Alejandro)
  3. Realizar un estudio de las soluciones anteriores presentes en la literatura sobre la predicción de relevancia en comentarios de usuarios. (Ray y Alejandro)

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

Tareas asociadas:

* 1. Diseñar una solución de clasificación de opiniones de usuarios para soporte de software utilizando modelos pre-entrenados de lenguaje. (Ray y Alejandro)
  2. Implementar la solución planteada. (Ray y Alejandro)

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 (Alejandro)
  2. Realizar experimentos con el modelo RoBERTa en cada set de datos (Ray)
  3. Realizar experimentos con el modelo GPT 2 en cada set de datos (Alejandro)
  4. Comparar los resultados obtenidos con los de otras investigaciones en la literatura. (Ray)

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 la clasificación de opiniones según su relevancia. Se plantean 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 [[13](#_ENREF_13)].

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 [[14](#_ENREF_14)]**.**

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. [[15](#_ENREF_15)].

En general, el sistema de clasificación de textos contiene cuatro niveles diferentes de alcance que pueden aplicarse [[16](#_ENREF_16)]:

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) [[17](#_ENREF_17), [18](#_ENREF_18)].

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 [[17](#_ENREF_17)]. 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) [[15](#_ENREF_15)]. 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) [[18](#_ENREF_18)].

**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 [[19](#_ENREF_19)].

**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 [[18](#_ENREF_18)].

## 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. [[20](#_ENREF_20)]. 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. [[18](#_ENREF_18)]. 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).[[20](#_ENREF_20)]

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 [[21](#_ENREF_21)] 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 [[20](#_ENREF_20)]. 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 [[18](#_ENREF_18)].

## 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 [[18](#_ENREF_18)]. En los últimos años se ha utilizado mayormente de forma complementaria junto a algoritmos de aprendizaje de máquina como es el caso de [[22](#_ENREF_22)]

**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.[[18](#_ENREF_18)]. Este enfoque se puede ver en [[23](#_ENREF_23)]

**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), [19](#_ENREF_19)]. Método utilizado en [[21](#_ENREF_21)]

**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 [[15](#_ENREF_15)]. Ejemplo de uso en [[24](#_ENREF_24)]

# 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 [[25](#_ENREF_25)].

## 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 [[26](#_ENREF_26)]. 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 [[27](#_ENREF_27)]. 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 [[20](#_ENREF_20)]. 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 [[19](#_ENREF_19)].

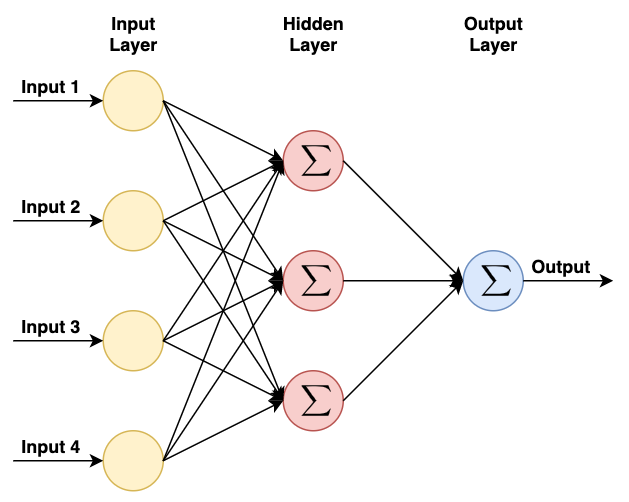


Figura 1.1: Estructura general una red neuronal [[19](#_ENREF_19)]

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 [[28](#_ENREF_28)]. 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). [[25](#_ENREF_25)].

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. [[25](#_ENREF_25)]. 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

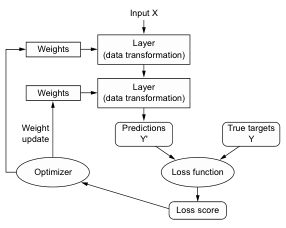


Figura 1.2: Esquema del proceso de entrenamiento de una red neuronal [[25](#_ENREF_25)]

### 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 [[18](#_ENREF_18)]. 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 [[29](#_ENREF_29)]. Esta arquitectura fue abordada para el problema concreto de requerimientos de software en [[30](#_ENREF_30)].

**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 [[18](#_ENREF_18)]. 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 [[31](#_ENREF_31)]. 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 [[16](#_ENREF_16)]. También se han desarrollado soluciones de clasificación de requerimientos con esta arquitectura [[9](#_ENREF_9)].

**Transformers:** arquitectura presentada en [[32](#_ENREF_32)] 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 [[25](#_ENREF_25)]. 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. [[15](#_ENREF_15)].

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 [[18](#_ENREF_18)]. 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 [[15](#_ENREF_15)].

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 [[18](#_ENREF_18)]. Algunos de los embeddings pre-entrenados más utilizados en el estado del arte son Word2vec de Google [[33](#_ENREF_33)], GloVe de Stanford [[34](#_ENREF_34)], y fasttext embeddings de Facebook [[35](#_ENREF_35)].

## 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 [[15](#_ENREF_15)]. 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 [[18](#_ENREF_18)]. 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 [[25](#_ENREF_25)]. 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 [[36](#_ENREF_36)], roBERTa [[37](#_ENREF_37), [38](#_ENREF_38)], alBERT [[38](#_ENREF_38)], BERTweet [[37](#_ENREF_37)], 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 [[18](#_ENREF_18), [19](#_ENREF_19)]. 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 [[36-38](#_ENREF_36)] 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 [[25](#_ENREF_25)].

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 [[36](#_ENREF_36)] utilizan el primer token de la salida como representación del texto completo, mientras otros como [[37](#_ENREF_37)] 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 [[36-38](#_ENREF_36)] 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 [[39](#_ENREF_39)]) 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.

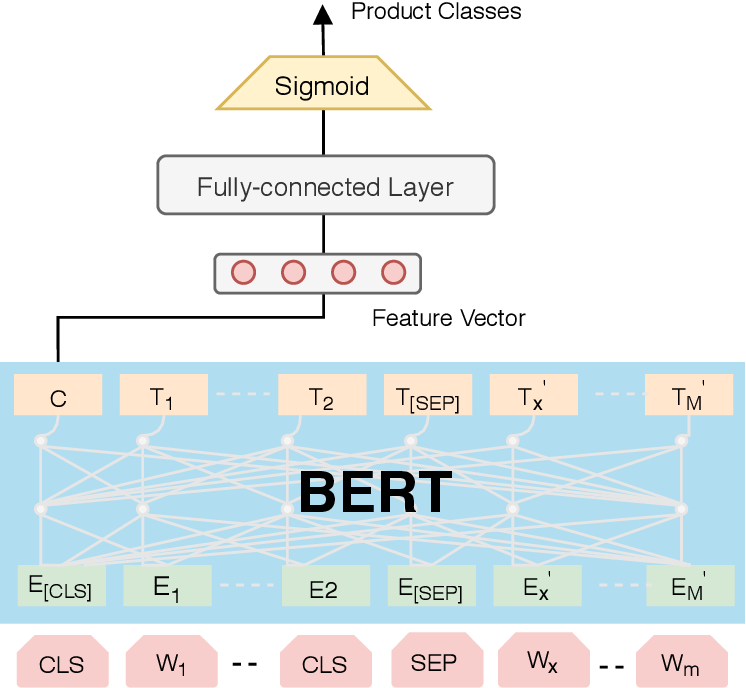


Figura 1.3: Estructura común de un modelo fine-tuned para una tarea de clasificación

# 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)] y otras que han utilizado un enfoque más general de clasificación teniendo en cuenta aspectos adicionales como reportes de errores o petición de características [[36-38](#_ENREF_36)].

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.

Los autores de [[36](#_ENREF_36)] 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.

En el caso de [[38](#_ENREF_38)] 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 entrenados con datos adicionales de redes sociales y textos de estilo informal.

# 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 [[28](#_ENREF_28)]. 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 [[40](#_ENREF_40)]. 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 [[36-38](#_ENREF_36)] 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 [[25](#_ENREF_25)]. 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 [[20](#_ENREF_20)].

## Métricas

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.

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

Tabla 1.1 Matriz de confusión para una clasificación específica

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 [[40](#_ENREF_40)]. 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 [[15](#_ENREF_15)]. 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 [[28](#_ENREF_28)]. 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 [[40](#_ENREF_40)]. 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 [[20](#_ENREF_20), [25](#_ENREF_25)]. 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 [[41](#_ENREF_41)]:

* 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 [[42](#_ENREF_42)].

**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 [[43](#_ENREF_43)]:

* 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.

**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 [[44](#_ENREF_44)]:

* 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 [[45](#_ENREF_45)]:

* 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 [[46](#_ENREF_46)]:

* 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 [[47](#_ENREF_47)].

**NumPy:** Biblioteca fundamental para el trabajo avanzado con datos numéricos, provee entre otras herramientas estructuras muy versátiles para el trabajo con vectores multidimensionales [[48](#_ENREF_48)].

**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 [[49](#_ENREF_49)].

# 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.
* Las soluciones centradas en relevancia de comentarios para soporte de software existentes utilizan algoritmos de aprendizaje automático tradicionales y no se explota el posible uso de soluciones con aprendizaje profundo.
* 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. La clasificación consiste en decir si el comentario es relevante o no.

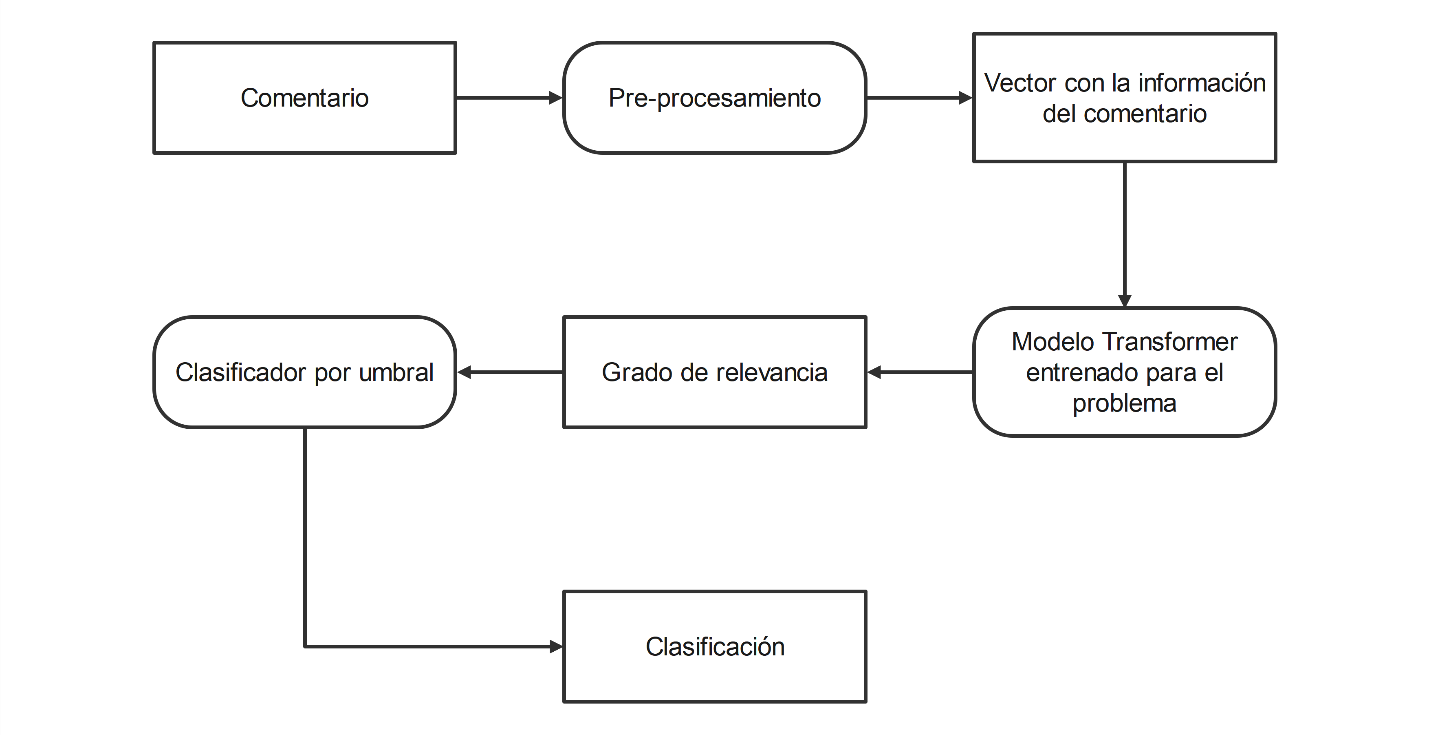


Figura 2.1: Esquema del flujo de trabajo de la aplicación

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 del comentario que es introducido al 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 la solución se realizó a partir de 2 modelos pre-entrenados diferentes con fines comparativos, GPT-2 y RoBERTa.

## 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 [PAD] 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 Modelo de clasificación

Se escogieron 2 modelos de leguaje pre-entrenados para probar la solución: GPT-2 y roBERTa los cuales fueron entrenados en el ámbito específico del problema antes de ser utilizados en la solución.

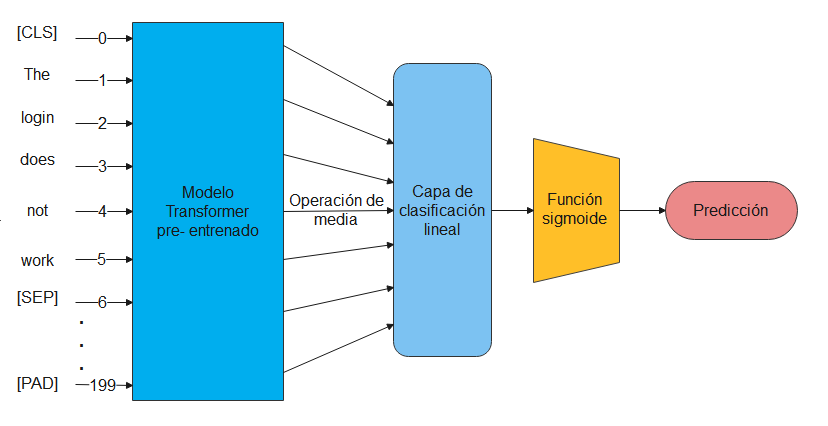


Figura 2.2: Arquitectura del modelo de clasificación planteado

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.

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”

# 2.2 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

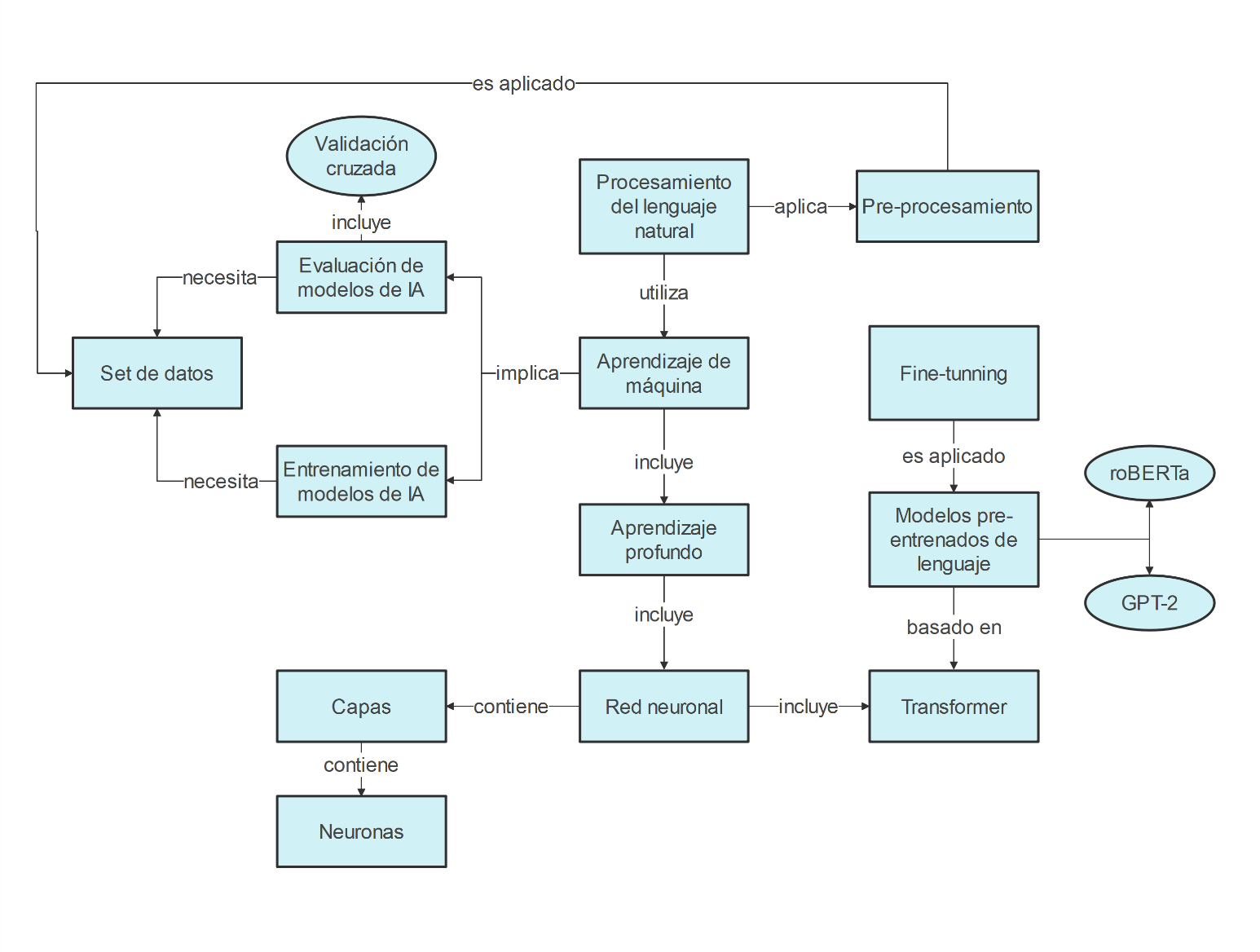
El modelo de dominio proporciona una visión estructural, ya que facilita representar el vocabulario y los conceptos claves del problema que se está modelando, así como las relaciones entre ellos. Dado que estos modelos son breves y bien estructurados facilitan capturar los conceptos de manera muy completa. 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 2.3 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 [[13](#_ENREF_13)]. En la figura 2.4 se muestra el diagrama de casos de uso diseñado como parte de la solución propuesta.

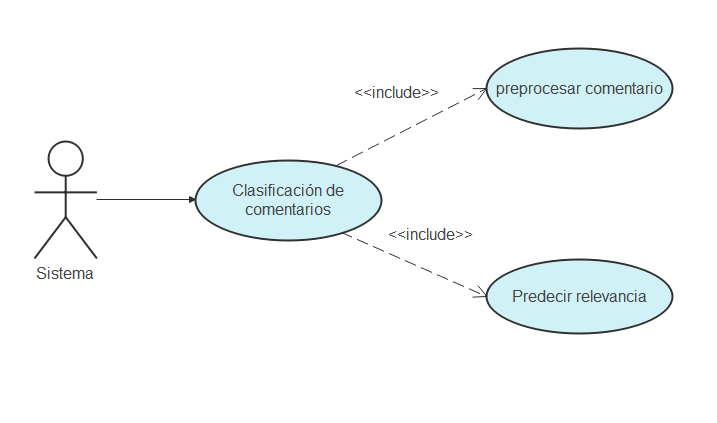


Figura 2.4: Diagrama de casos de uso del sistema

### 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 [[13](#_ENREF_13)]. Dichas descripciones se presentan a continuación:

|  |  |
| --- | --- |
| **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 2.1: Descripción del caso de uso clasificación de comentarios

|  |  |
| --- | --- |
| **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 2.2: Descripción del caso de uso preprocesar comentario

|  |  |
| --- | --- |
| **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 |

Tabla 2.3: Descripción del caso de uso predecir relevancia

# 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 y el uso de modelos de lenguaje pre-entrenados de arquitectura Transformer.
* Fueron creados 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 2 sets de datos: Facebook y Swiftkey. Para realizar la evaluación se seleccionaron 2 modelos de lenguaje: roBERTa (en su tamaño base) y GPT-2 (en su tamaño base). Los modelos fueron entrenados utilizando datos en el idioma inglés. RoBERTa fue elegido porque ha obtenido resultados del estado del arte en muchas tareas de clasificación[[37](#_ENREF_37), [38](#_ENREF_38), [50](#_ENREF_50)] y GPT 2 para tener un punto de comparación fuera de la familia de los BERT.

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 [[36](#_ENREF_36)]. 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.

# 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)].

Por dificultades de tiempo en el presente estudio solo se hicieron evaluaciones con 2 de los sets de datos mencionados anteriormente: Facebook y SwiftKey. 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.

|  |  |  |
| --- | --- | --- |
|  | **Facebook** | **SwiftKey** |
| **Relevante** | 1662 | 879 |
| **No relevante** | 1338 | 2121 |
| **Porcentaje de relevantes** | 55.4% | 29.3% |
| **Total de comentarios** | 3000 | 3000 |

Tabla 3.1: Composición de los sets de datos utilizados

Como se puede apreciar en la tabla el set de datos de Facebook está balanceado pero el de SwiftKey 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. Fueron escogidos estos 2 sets de datos precisamente debido a esta discrepancia para tener más valor comparativo.

# 3.3 Resultados experimentales

## 3.3.1 Resultados experimentales para el set de datos Facebook

Las siguientes tablas muestran los resultados de las métricas analizadas en el proceso de validación cruzada con ambos modelos para el set de datos Facebook:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RoBERTa-base** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **1ra iteración** | 0.935 | 0.888 | 0.947 | 0.916 |
| **2da iteración** | 0.945 | 0.885 | 0.987 | 0.933 |
| **3ra iteración** | 0.951 | 0.916 | 0.976 | 0.945 |
| **4ta iteración** | 0.933 | 0.930 | 0.930 | 0.930 |
| **5ta iteración** | 0.933 | 0.930 | 0.923 | 0.927 |
| **Media** | **0.939** | **0.910** | **0.953** | **0.930** |

Tabla 3.2: Resultado de la validación cruzada con roBERTa en el set de datos Facebook

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GPT 2 - base** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **1ra iteración** | 0.920 | 0.917 | 0.888 | 0.902 |
| **2da iteración** | 0.878 | 0.938 | 0.811 | 0.870 |
| **3ra iteración** | 0.863 | 0.777 | 0.910 | 0.838 |
| **4ta iteración** | 0.918 | 0.944 | 0.891 | 0.916 |
| **5ta iteración** | 0.920 | 0.927 | 0.900 | 0.913 |
| **Media** | **0.900** | **0.900** | **0.880** | **0.884** |

Tabla 3.3: Resultado de la validación cruzada con GPT 2 en el set de datos Facebook

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modelo** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **RoBERTa-base** | **0.939** | **0.910** | **0.953** | **0.930** |
| **GPT 2 - base** | 0.900 | 0.900 | 0.880 | 0.884 |

Tabla 3.4: Comparación de ambos modelos en el set de datos Facebook

Resultados:

* En cada una de las métricas analizadas roBERTa presenta un mayor desempeño que GPT 2.
* La métrica precision es donde único la diferencia entre ambos modelos no es significativa.

## 3.3.2 Resultados experimentales para el set de datos Swiftkey

Las siguientes tablas muestran los resultados de las métricas analizadas en el proceso de validación cruzada con ambos modelos para el set de datos SwiftKey:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RoBERTa-base** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **1ra iteración** | 0.925 | 0.953 | 0.937 | 0.945 |
| **2da iteración** | 0.926 | 0.936 | 0.958 | 0.947 |
| **3ra iteración** | 0.906 | 0.979 | 0.900 | 0.937 |
| **4ta iteración** | 0.943 | 0.976 | 0.945 | 0.960 |
| **5ta iteración** | 0.906 | 0.882 | 0.984 | 0.930 |
| **Media** | **0.921** | **0.945** | **0.945** | **0.944** |

Tabla 3.5: Resultado de la validación cruzada con roBERTa en el set de datos SwiftKey

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GPT 2 - base** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **1ra iteración** | 0.726 | 0.815 | 0.792 | 0.803 |
| **2da iteración** | 0.741 | 0.992 | 0.734 | 0.844 |
| **3ra iteración** | 0.751 | 0.916 | 0.777 | 0.841 |
| **4ta iteración** | 0.776 | 0.934 | 0.790 | 0.856 |
| **5ta iteración** | 0.746 | 0.957 | 0.752 | 0.842 |
| **Media** | **0.748** | **0.923** | **0.769** | **0.837** |

Tabla 3.6: Resultado de la validación cruzada con GPT 2 en el set de datos SwiftKey

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modelo** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **RoBERTa-base** | **0.921** | **0.945** | **0.945** | **0.944** |
| **GPT 2 - base** | 0.748 | 0.923 | 0.769 | 0.837 |

Tabla 3.7: Comparación de ambos modelos en el set de datos SwiftKey

Resultados:

* En cada una de las métricas analizadas roBERTa presenta un mayor desempeño que GPT 2.
* A pesar del notable desbalance del set de datos roBERTa fue capaz de mantener buenos resultados.
* Se puede notar el impacto del desbalance en los resultados de GPT al ver la gran brecha entre precision y recall.

## 3.3.3 Análisis general de los resultados experimentales

Como se pudo observar en las 2 secciones anteriores el modelo roBERTa tuvo un desempeño superior a GPT 2 en todas las métricas analizadas para ambos sets de datos. Pero en estos resultados hay detalles interesantes que merecen mención. Dicho detalle es la adaptación al balanceo de los sets de datos, pues como ya se mencionó anteriormente los datos de Facebook están balanceados mientras los de SwiftKey no.

Figura 3.1: resultados utilizando roBERTa en ambos sets de datos

En estas condiciones el modelo roBERTa logra mantener unos resultados similares en ambos sets de datos como se puede apreciar en la figura 3.1.

Figura 3.2: resultados utilizando roBERTa en ambos sets de datos

Sin embargo, GPT 2 presenta una caía notable en su desempeño como se puede ver en la figura 3.2, esto se debe a que el modelo está priorizando a la clase mayoritaria debido a la falta de información sobre la clase minoritaria, la mayor prueba de esto es la brecha existente entre las métricas precision y recall.

## 3.3.4 Comparación con otros resultados en la literatura

La tabla a continuación presenta la comparación de los resultados obtenidos frente a los reportados en las investigaciones anteriores. Para la comparación se estableció la métrica F1-score como medida de comparación.

|  |  |  |
| --- | --- | --- |
| Solución | Facebook | SwiftKey |
| [Chen, et al. [10]](#_ENREF_10) | 0.877 | 0.764 |
| [Milián Núñez, et al. [12]](#_ENREF_12) | 0.803 | - |
| [Martín [51]](#_ENREF_51) | 0.907 | 0.899 |
| GPT 2 - base | 0.884 | 0.837 |
| RoBERTa - base | **0.930** | **0.944** |

Tabla 3.8 Comparación de la solución propuesta con otras soluciones de la literatura

Como se puede apreciar en la tabla anterior la solución propuesta utilizando el modelo de lenguaje roBERTa para la clasificación según relevancia en comentarios de usuarios alcanza resultados superiores a los de sus antecedentes. Sin embargo la solución utilizando GPT 2 se mantiene por debajo de la propuesta por [[11](#_ENREF_11)]. Esto demuestra que el simple hecho de utilizar un modelo de lenguaje de arquitectura Transformer no eclipsa las soluciones de aprendizaje automático tradicionales, sino que se debe escoger un modelo adecuado.

# 3.4 Conclusiones parciales

* La solución utilizando el modelo roBERTa alcanza mejores resultados en cada métrica analizada y set de datos analizados.
* La solución utilizando roBERTa alcanza mejores resultados que sus antecedentes en la literatura que utilizan técnicas tradicionales de aprendizaje automático.
* El modelo RoBERTa logró adaptarse al desbalance presentado en el set de datos SwiftKey mientras GPT 2 sufrió una notable pérdida de eficacia.
* Los resultados obtenidos con el modelo GPT 2 son inferiores a uno de los resultados anteriores con aprendizaje automático tradicional. Por lo tanto, el uso de modelos de lenguaje de arquitectura Transformer no garantiza resultados superiores a las técnicas de aprendizaje automático tradicionales, influye significativamente la elección del modelo.

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 no había soluciones en la literatura enfocadas en la detección de relevancia de opiniones de usuarios que utilizaran modelos de lenguaje de arquitectura Transformer.
* Se desarrolló una solución para la clasificación de opiniones de usuarios según su relevancia para el soporte de software utilizando modelos de lenguaje de arquitectura Transformer.
* Los resultados experimentales probaron que la solución propuesta utilizando RoBERTa obtiene mejores resultados en todas las métricas y sets de datos utilizados que su homóloga utilizando GPT 2.
* Los resultados obtenidos con el modelo RoBERTa superan a los registrados en la literatura para la detección de relevancia en opiniones de usuarios.
* Los resultados obtenidos con GPT 2 no superaron los resultados de todas las soluciones anteriores en la literatura que utilizan técnicas de aprendizaje automático tradicionales, demostrando que el simple hecho de utilizar modelos de lenguaje de arquitectura Transformer no eclipsa las soluciones anteriores, es necesario elegir un modelo adecuado

Recomendaciones

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

* Extender la evaluación a otros sets de datos disponibles que hayan sido utilizados en soluciones similares.
* Evaluar la solución propuesta con otros modelos de lenguaje como BERT, ALBERT o XLNET
* Evaluar la solución utilizando otros hiper-parámetros de entrenamiento en busca de los óptimos
* Adaptar la solución para hacerla extensible a idioma 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] I. Sommerville, *Software engineering*, 10th ed. Pearson, 2016.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

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

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

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

[30] 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.

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

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

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

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

[41] (10 de febrero del 2024). *Python web*. Available: <https://www.python.org/about/>

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

[43] (25 de febrero de 2024). *PyTorch web*. Available: <https://pytorch.org/features/>

[44] (25 de febrero de 2024). *PyTorch Lightning web*. Available: <https://lightning.ai/pytorch-lightning>

[45] (25 de febrero de 2024). *TorchMetrics web*. Available: <https://lightning.ai/docs/torchmetrics/stable/>

[46] 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.

[47] (10 de febrero del 2024). *NLTK web*. Available: <https://www.nltk.org/>

[48] (10 de febrero del 2024). *NumPy web*. Available: <https://numpy.org/about/>

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

[50] A. Murarka, B. Radhakrishnan, and S. Ravichandran, "Classification of mental illnesses on social media using RoBERTa," in *Proceedings of the 12th international workshop on health text mining and information analysis*, 2021, pp. 59-68.

[51] 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.