



## CARRERA DE ESPECIALIZACIÓN EN INTELIGENCIA ARTIFICIAL

MEMORIA DEL TRABAJO FINAL

### **Clasificación de reclamos de usuario**

**Autor:**

**Ing. Lucas Rivela**

Director:

Dr. Lic. Rodrigo Cárdenas (FIUBA)

Jurados:

Nombre del jurado 1 (pertenencia)

Nombre del jurado 2 (pertenencia)

Nombre del jurado 3 (pertenencia)

*Este trabajo fue realizado en la Ciudad Autónoma de Buenos Aires,  
entre mayo de 2023 y septiembre de 2023.*



## *Resumen*

La presente memoria describe el diseño e implementación de un sistema de clasificación de reclamos desarrollado para Ualá. La solución permite optimizar el tiempo que lleva el proceso de clasificación mediante la asignación automática de categorías de primer y segundo nivel; y por otro lado, reducir la cantidad de personas necesarias para esta tarea.

Para poder realizar este trabajo se aplicaron conceptos de bases de datos, procesamiento del lenguaje natural y aprendizaje profundo para realizar la extracción de los datos, el procesamiento del texto, el entrenamiento de los modelos de IA y el despliegue de los mismos en un ambiente de desarrollo.



## *Agradecimientos*

A mi familia y amigos por apoyarme durante la realización de esta carrera.

A mis compañeros y profesores por el acompañamiento.

A mi director, Dr. Lic. Rodrigo Cárdenas por orientarme y guiarme en la realización de este trabajo.



# Índice general

<b>Resumen</b>	<b>I</b>
<b>1. Introducción general</b>	<b>1</b>
1.1. Introducción	1
1.1.1. Qué es la atención al cliente	1
1.1.2. Organización de un equipo de atención al cliente	1
1.1.3. El proceso de atención al cliente	2
1.2. Motivación	3
1.3. Estado del arte	4
1.3.1. Word Embeddings	4
1.3.2. Transformers	4
1.4. Objetivos y alcance	5
1.4.1. Objetivos	5
1.4.2. Alcance	6
<b>2. Introducción específica</b>	<b>7</b>
2.1. Requerimientos	7
2.2. Preprocesamiento del texto	8
2.2.1. Natural Language Toolkit	8
Segmentación	8
Remoción de palabras vacías	8
Derivación	9
2.2.2. Scikit-Learn	9
Normalización de etiquetas	9
Vectorizador Term Frequency - Inverse Document Frequency	9
2.3. Modelos de inteligencia artificial utilizados	10
2.3.1. Complement Naive Bayes	10
2.3.2. Representaciones de codificador bidireccional de transformadores	11
2.4. Herramientas de software utilizadas	13
2.4.1. Python	13
2.4.2. GitHub	13
2.4.3. BigQuery	13
2.4.4. Vertex AI workbench	13
2.4.5. Docker	13
2.4.6. Artifact registry	14
2.4.7. Cloud Composer y Apache Airflow	14
<b>3. Diseño e implementación</b>	<b>15</b>
3.1. Arquitectura del sistema	15
3.2. Extracción y preprocesamiento del texto	16
3.2.1. Extracción de datos de BigQuery	16

3.2.2. <i>Pipeline</i> de preprocesamiento . . . . .	17
Codificación para CNB . . . . .	19
Codificación para BERT . . . . .	19
3.3. Entrenamiento de los modelos . . . . .	20
3.4. Desarrollo de GitHub Actions . . . . .	20
3.5. Empaquetamiento del código y artefactos . . . . .	20
3.6. Desarrollo del <i>pipeline</i> de predicción . . . . .	20
3.7. Almacenamiento de predicciones . . . . .	20
<b>4. Ensayos y resultados</b>	<b>21</b>
4.1. Pruebas funcionales del hardware . . . . .	21
<b>5. Conclusiones</b>	<b>23</b>
5.1. Conclusiones generales . . . . .	23
5.2. Próximos pasos . . . . .	23
<b>Bibliografía</b>	<b>25</b>



# Índice de figuras

1.1. Organigrama de un área de atención al cliente <sup>1</sup> . . . . .	2
1.2. Evolución de las redes en cantidad de parámetros. <sup>2</sup> . . . . .	5
1.3. Diagrama general de la solución. . . . .	6
2.1. Ejemplo de codificación realizado sobre una variable <sup>3</sup> . . . . .	9
2.2. Ejemplo de funcionamiento del mecanismo de atención propia <sup>4</sup> . . .	12
2.3. Ejemplo de entrada de datos a BERT con MLM y NSP combinados <sup>5</sup> . .	12
3.1. Diagrama de la arquitectura de alto nivel. . . . .	15
3.2. Cantidad de categorías L2 para cada L1. . . . .	17
3.3. Diagrama del procesamiento del texto crudo. . . . .	18
3.4. Diagrama de codificación para CNB. . . . .	19
3.5. Diagrama de codificación para BERT. . . . .	19



# Índice de tablas



# Capítulo 1

## Introducción general

En este capítulo se realiza una introducción al funcionamiento de un área de atención al cliente. Además, se menciona el estado del arte de los sistemas de procesamiento del lenguaje natural, y por último se explican los objetivos y alcances del presente trabajo.

### 1.1. Introducción

En esta sección se introduce un área típica de atención al cliente en una empresa.

#### 1.1.1. Qué es la atención al cliente

El área de atención al cliente se encarga de dar soporte al consumidor y tiene como objetivo resolver sus problemas [1].

A menudo se confunde el área de atención con el área de servicio al cliente. La principal diferencia radica en que el servicio es una función proactiva con la intención de anticiparse a las necesidades inmediatas y a largo plazo del cliente. La atención al cliente es una función reactiva que busca resolver los problemas que el cliente manifestó [2].

A continuación se listan las características principales del proceso [2]:

1. Inicio y duración: inicia cuando un cliente se pone en contacto con la empresa. Demora el tiempo que sea necesario hasta brindar una solución.
2. Objetivo: solucionar problemas que surjan del funcionamiento del producto o condiciones del servicio.
3. Actitud: reactiva.
4. Interacción: canales específicos. Por ejemplo, telefónicos, *email* y *chat*.
5. Participantes: cliente y representante del centro de atención. Raras veces entran en juego otros trabajadores de la empresa.

#### 1.1.2. Organización de un equipo de atención al cliente

A continuación se enumeran los principales roles que podemos encontrar en el área de atención al cliente:

1. Gerente de atención al cliente: tiene bajo su responsabilidad el cumplimiento de metas estratégicas. Gestionan el volumen de casos entrantes y comunican las tendencias a otros departamentos. [3]

2. Coordinador de atención al cliente: es quien está en el día a día en contacto con los agentes. Apoya la resolución de problemas y escala aquellos que requieren validaciones o decisiones que no estén a su alcance [4].
3. Analista de atención al cliente: canaliza las quejas, reclamos y sugerencias. Se encarga de proveer soporte a los usuarios. [4]

En algunas organizaciones, dependiendo de su tamaño, se pueden encontrar más o menos mandos intermedios entre el gerente y el coordinador. Una variante puede ser tener varios coordinadores respondiendo a un supervisor. Este supervisor puede estar a cargo de la atención de reclamos para ciertos productos de la empresa que están relacionados.

En la figura 1.1 podemos ver la principal tendencia que hay en cuanto a organización de equipos. La idea es que cada equipo sea especialista en un conjunto de casos.



FIGURA 1.1. Organigrama de un área de atención al cliente<sup>1</sup>.

### 1.1.3. El proceso de atención al cliente

El proceso de atención al cliente consiste en una serie de pasos que se realizan para atender los reclamos y/o consultas que recibe la empresa [5].

A continuación se describen los pasos principales de un proceso de atención típico [6].

- Contacto y captura de la demanda del cliente: en esta etapa se recibe el mensaje del cliente por cualquiera de los canales establecidos y se procede a registrarlo en el sistema.
- Análisis y clasificación: se analiza la información recibida y se evalúa si es suficiente para proceder a la clasificación del caso. De lo contrario se vuelve a contactar al cliente para obtener más detalles.

<sup>1</sup>Imagen tomada de <https://blog.hubspot.es/service/que-es-atencion-al-cliente>

- Resolución: en esta etapa se busca dar una respuesta a la consulta o reclamo del cliente. En algunos casos puede involucrar varios equipos o personas, por lo que su duración es variable.
- Cierre: en esta etapa se presenta la solución al cliente y se le comunican los próximos pasos si los hubiera.

## 1.2. Motivación

En la actualidad, resulta sumamente sencillo para el cliente optar por la competencia si la empresa no logra satisfacer sus expectativas. Una mala CX (*Customer Experience*) puede generar un efecto de bola de nieve, ya que los usuarios que hayan tenido una mala experiencia, son mas propensos a comunicarlo con sus conocidos haciendo que la empresa no sólo pierda un cliente sino que además, se le dificulte expandir su mercado [7].

Según informes de *CX Trends* [8][9][10] y de Esteban Kolsky [11]:

- El 70 % de los consumidores gastará más en una empresa que ofrezca una buena CX.
- El 50 % de los clientes se pasaría a la competencia después de haber tenido tan solo una mala experiencia. Este valor sube a un 80 % si se les pregunta si hubieran tenido dos o más malas experiencias.
- El 72 % de los clientes compartiría una experiencia positiva con 6 o más personas.
- El 13 % de los clientes compartiría su experiencia con 15 o más personas si no está satisfecho.
- Sólo el 3.85 % de los clientes insatisfechos se lo comunican a la empresa.

Estos números muestran que la ausencia de quejas no es un signo de satisfacción. Por el contrario, es probable que los clientes hayan abandonado la empresa y estén compartiendo su insatisfacción con otras personas. También muestran la importancia de mantenerlos satisfechos, ya que son propensos a divulgarlo.

Más aún, entre las principales razones de una mala atención al cliente se encuentran tener tiempos de espera demasiado largos [12] y tener muchas derivaciones internas [13].

De lo anterior se infiere que resulta muy importante que los procesos del área se realicen de forma eficiente. La utilización de modelos de IA para automatizar el proceso de clasificación de reclamos y consultas, ayuda a la empresa a responder a sus clientes con tiempos de respuesta menores y permite disponer de más analistas en la etapa de resolución y cierre. De esta manera, se genera una imagen positiva de la empresa que facilita la fidelización de clientes existentes y aumenta la incorporación de los nuevos.

### 1.3. Estado del arte

El PNL (Procesamiento del lenguaje natural) es un subcampo de la inteligencia artificial que busca enseñar a un programa informático a comprender, interpretar y generar texto en lenguaje humano. A principios del siglo XXI, el PNL experimentó un crecimiento significativo gracias a la aplicación de algoritmos de aprendizaje profundo y la disponibilidad de grandes corpus de texto [14].

#### 1.3.1. Word Embeddings

En 2003 se entrena la primer red neuronal orientada al lenguaje utilizando vectores para representar palabras. Llamarían a este proceso “aprender una representación distribuida para cada palabra” [15].

En 2008 se introduce el concepto de *Word Embeddings* como una herramienta potente para tareas de PNL. Se distinguirían de sus antecesores mencionando que su objetivo era predecir la relevancia de una palabra dada la parte previa y posterior de la oración, a diferencia del trabajo previo que buscaba predecir la probabilidad de una palabra dada la parte previa de una oración.

En 2013, publican un artículo dónde detallan las arquitecturas *CBOW* y *Skip-Gram*. Además liberan el primer modelo pre-entrenado llamado *Word2Vec* (basado en *Skip-Gram*) popularizando el uso de los *Word Embeddings* [16].

Posteriormente, en 2014 publicarían *GloVe* como otro método de generación de *Word Embeddings* que utiliza una probabilidad de co-ocurrencia [17]. Con esta técnica, si dos palabras co-existen muchas veces, ambas palabras tienen una probabilidad alta de tener un mismo significado.

Los *Word Embeddings* se convirtieron en la herramienta principal dentro del PNL. Capturan el significado de una palabra y la traducen a una representación numérica que puede ser usada como entrada para las redes neuronales.

#### 1.3.2. Transformers

El avance del aprendizaje profundo, permitió a los investigadores desarrollar arquitecturas de redes neuronales más avanzadas y eficientes.

Sin embargo, el verdadero hito no fue hasta 2017, cuando se publica la utilización de un mecanismo de atención para desarrollar una nueva arquitectura que se llamaría *Transformers* [18]. Las redes más usadas hoy en día están basadas en esta mejora.

Básicamente, en los *Transformers* se reemplazan las capas recurrentes que se venían utilizando hasta ese momento por “capas de atención” que codifican cada palabra en función del resto de la frase, permitiendo de esta forma, introducir el contexto en la representación matemática del texto. Por este motivo, sus *Embeddings* generados son denominados *Embeddings* contextuales.

Otra de las innovaciones introducidas es el uso de *Embeddings* posicionales. Con ellos se logra una mayor paralelización, ya que no es más necesario pasar una palabra a la vez. Agregando un valor secuencial con cada palabra, hacen posible pasarle a la red todas las palabras en simultáneo.



Estas redes fueron evolucionando en tamaño y complejidad conforme transcurría el tiempo. En la figura 1.2 se puede ver la evolución de la cantidad de parámetros de estas redes a lo largo de los años.

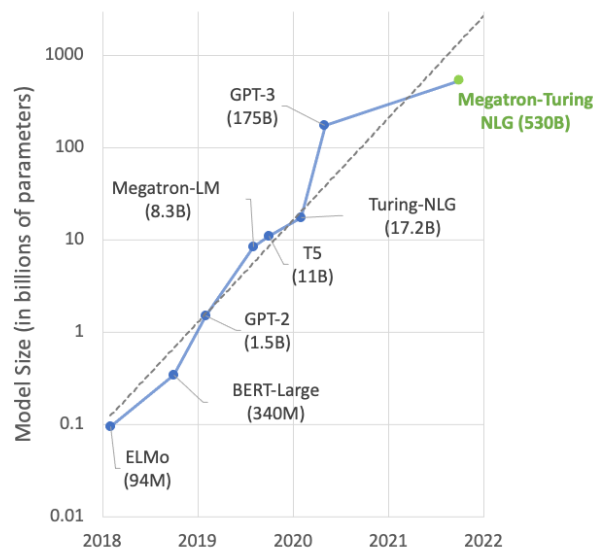


FIGURA 1.2. Evolución de las redes en cantidad de parámetros.<sup>2</sup>

Los modelos basados en *Transformers* se convirtieron en el modelo por defecto para tareas como traducción, clasificación de texto y resumen de textos por citar algunos ejemplos.

En el área de atención al cliente, los principales usos de estas redes son [19]:

- *Chatbots*: ayudan a responder consultas de forma inmediata evitando filas de espera.
- Análisis de sentimiento: para realizar análisis sobre comentarios y quejas de los clientes.
- Redireccionamiento de tickets: eliminan cuellos de botella en la parte de clasificación al redirigir el ticket directamente al área correspondiente.

## 1.4. Objetivos y alcance

### 1.4.1. Objetivos

El propósito de este trabajo fue el desarrollo de modelos de inteligencia artificial (IA) para mejorar y agilizar la atención al cliente. Para lograr este objetivo es necesario realizar la clasificación automática de los reclamos de usuario en categorías de primer y segundo nivel.

En la figura 1.3 se presenta un diagrama de alto nivel de la solución. Se observa que en primera instancia, los reclamos se reciben por correo y chat, y luego son registrados en Salesforce, el sistema de CRM (*Customer Relationship Management*) que utiliza el área de atención al cliente. Estos datos se replican en BigQuery a través de un proceso orquestado por Apache Airflow.

<sup>2</sup>Imagen tomada de <https://developer.nvidia.com/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nl>

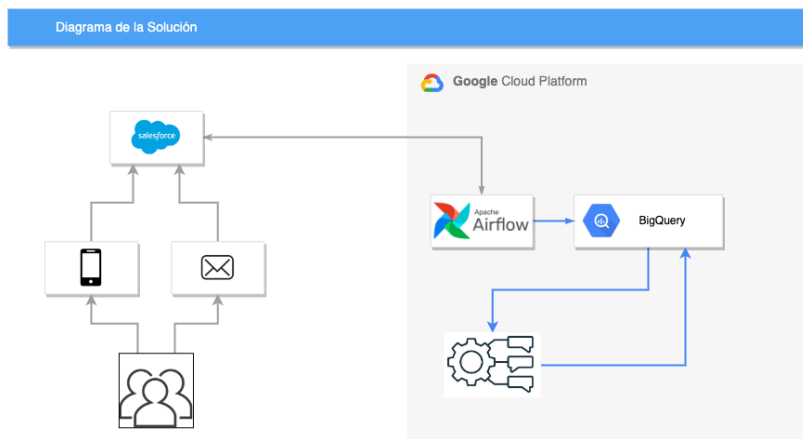


FIGURA 1.3. Diagrama general de la solución.

Los modelos de IA clasifican los casos que están en BigQuery y guardan la categoría de primer y segundo nivel en una tabla que luego el usuario podrá utilizar.

#### 1.4.2. Alcance

El alcance de este proyecto estuvo orientado a desarrollar un prototipo de solución de software que incluyó las siguientes actividades:

- Obtención de los datos: se corresponde con el análisis de las fuentes de datos disponibles, tanto para entrenar los modelos de IA como para la parte de predicción de nuevos casos.
- Análisis exploratorio de los datos: se corresponde con las actividades necesarias para generar nuevos *insights*, que sirvieron para guiar el desarrollo de los modelos.
- Modelado: se corresponde con la generación de variables a partir de los datos disponibles.
- Entrenamiento: se corresponde con el entrenamiento de los modelos a partir de las variables obtenidas. También incluye la selección del mejor modelo para cada clasificación.
- Despliegue: se corresponde con el diseño de la infraestructura para ejecutar los modelos de IA y su despliegue en un ambiente no productivo.
- Documentación: se corresponde con los documentos de soporte que explican los procesos de modelado, entrenamiento y despliegue.

El alcance de este trabajo no cubre:

- La adaptación de los modelos a nuevas categorías inexistentes en los años 2021 y 2022.
- El despliegue de los modelos en un ambiente productivo.
- El soporte de la infraestructura desplegada en el ambiente no productivo.

## Capítulo 2

# Introducción específica

En este capítulo se enumeran los requisitos que la solución debe cumplir y luego se describen principalmente las herramientas utilizadas durante el desarrollo para el entrenamiento y despliegue de los modelos para satisfacerlos.

### 2.1. Requerimientos

En esta sección se detallan los requerimientos funcionales y las restricciones de implementación del trabajo.

#### 1. Requerimientos funcionales

- a) El sistema debe poder detectar la categoría de un reclamo escrito en lenguaje natural.
- b) El sistema debe poder detectar la categoría de una consulta escrita en lenguaje natural.
- c) El usuario debe poder utilizar los resultados de la clasificación desde una base de datos.
- d) El proceso debe ser capaz de interpretar errores de ortografía.
- e) El proceso debe ser capaz de adaptarse a distinta cantidad de palabras en el mensaje.
- f) La solución debe ejecutarse en forma *batch*, corriendo diariamente y tomando los casos del día anterior.

#### 2. Requerimientos no funcionales

- a) El sistema debe estar desarrollado en lenguaje Python.
- b) El código debe ser versionado con Git.
- c) La solución debe estar desplegada sobre infraestructura de Google Cloud Platform.
- d) La salida de los modelos debe ser almacenada en BigQuery.
- e) El proceso debe ser ejecutado a través del orquestador Apache Airflow.

#### 3. Requerimientos de testing

- a) Se deben generar métricas de desempeño de los modelos con el dataset de entrenamiento y de prueba.

#### 4. Requerimientos de documentación

- a) Se debe confeccionar un documento con el diseño de la arquitectura de alto nivel.
- b) Se debe confeccionar un documento con el diseño de los modelos de IA.
- c) Se debe confeccionar un documento que especifique los datos que consumen los modelos y su origen.

## 2.2. Preprocesamiento del texto

En esta sección se introducen las herramientas utilizadas para realizar la limpieza y preparación del texto que sirvió tanto para la parte de entrenamiento como para la parte de predicción en el despliegue.

### 2.2.1. Natural Language Toolkit

NLTK (*Natural Language Toolkit*) es una librería de Python para preprocesar texto. Es una de las herramientas principales del mercado y entre otras cosas nos permite:

- Realizar la “tokenización” o segmentación del texto.
- Remover *stopwords* o palabras vacías.
- Aplicar *stemming* sobre las palabras.
- Realizar “lematización” sobre las palabras.
- Etiquetar cada término en una oración según sea sustantivo, verbo, adjetivo, preposición, etc.

A continuación se profundizará en las funcionalidades de esta librería que se usaron para este trabajo.

#### Segmentación

La segmentación consiste en separar un documento en términos individuales. Estos términos se llaman *tokens* y es la unidad mínima de análisis de texto. Dependiendo de la tarea en cuestión, un *token* puede ser una palabra, una sílaba o incluso un solo carácter.

Esta etapa es importante porque sirve como base para las etapas posteriores del preprocesamiento del texto.

#### Remoción de palabras vacías

Las *stopwords* o palabras vacías son palabras que son muy comunes en nuestro idioma y están destinadas a aparecer en todas las oraciones, sin importar el tema al que hagan referencia [20], por ejemplo: “en”, “este”, “el”, “las”.

Eliminar palabras comunes nos permite eliminar palabras con poco valor discriminativo entre textos (o categorías de reclamos). Además reduce el volumen de datos a procesar.

## Derivación

El *stemming* o derivación o poda consiste en recortar las palabras para reducirlas a una base común. Es decir, devuelve el tallo de una palabra, que no necesariamente es igual a la raíz morfológica de la palabra. Por ejemplo, para la palabra “comienza” su derivación será “comienz” y para “peces” será “pec”.

La idea detrás de esta técnica es tratar de agrupar las palabras similares, reduciendo la cantidad de palabras únicas en un *dataset*.

### 2.2.2. Scikit-Learn

Scikit-Learn es una librería de Python para *Machine Learning* que cuenta con una variedad de algoritmos y modelos para clasificación y regresión. Además cuenta con un conjunto de herramientas de extracción y generación de variables, tanto para datos numéricos como para texto.


A continuación se dará una explicación de los módulos usados de esta librería.

## Normalización de etiquetas

La normalización de etiquetas o *label encoding* consiste en codificar variables categóricas en valores numéricos que irán entre 0 y  $n - 1$  siendo  $n$  la cantidad de categorías. Es decir, asigna un número único a cada categoría. De esta forma, al entrenar los modelos de IA se puede pasar la variable *target* en formato numérico.

Otra utilidad muy importante de la versión que trae Scikit-Learn es la posibilidad de realzar la transformación inversa. Esto resulta muy útil para determinar el texto de la variable numérica que predice el modelo.

En la figura 2.1 podemos ver un ejemplo de la transformación.



State (Nominal Scale)	State (Label Encoding)
Maharashtra	3
Tamil Nadu	4
Delhi	0
Karnataka	2
Gujarat	1
Uttar Pradesh	5

FIGURA 2.1. Ejemplo de codificación realizado sobre una variable<sup>1</sup>.

## Vectorizador Term Frequency - Inverse Document Frequency

La vectorización es una técnica de PNL que convierte una secuencia de *tokens* (obtenidos previamente en la etapa de segmentación) a un vector numérico.

Hay distintas maneras de realizar este proceso, la elegida para este trabajo es la que se conoce como TF-IDF (*Term Frequency - Inverse Document Frequency*). Su objetivo es reflejar cuán importante es una palabra respecto al documento. *Nota:*

<sup>1</sup>Imagen tomada de <https://www.mygreatlearning.com/blog/label-encoding-in-python/>

Para este trabajo, la palabra documento representa un caso de reclamo o consulta de un cliente.

Tiene dos partes: el cálculo de TF y el cálculo de IDF.

TF calcula para cada documento del *dataset*, la cantidad de veces que un *token* aparece en él. Es decir, calcula la frecuencia de un *token* en cada documento. Luego, ese número se divide por la cantidad de palabras que tenía ese documento. La fórmula para calcularla es la siguiente [21]

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}} \quad (2.1)$$

IDF calcula la proporción de documentos del *dataset* que poseen el *token*. Es decir, divide la cantidad total de documentos sobre la cantidad de documentos con ese *token*, y luego a ese resultado le aplica el logaritmo. Los términos raros tendrán un puntaje más alto. La siguiente ecuación muestra como se calcula:

$$idf(w) = \log\left(\frac{N}{df_i}\right) \quad (2.2)$$

Finalmente, una vez obtenidos el TF y el IDF, lo que se hace es multiplicar ambos términos para cada *token* en todos los documentos:

$$w_{i,j} = tf_{i,j} \times idf(w) \quad (2.3)$$

## 2.3. Modelos de inteligencia artificial utilizados

En esta sección se presentan los modelos de IA utilizados.

### 2.3.1. Complement Naive Bayes

Los algoritmos de *Naive Bayes* o Bayes ingenuo son muy utilizados para tareas de clasificación por su velocidad de cómputo. Son modelos probabilísticos que deben su nombre a la probabilidad condicional y el teorema de Bayes. Se les denomina ingenuos porque suponen una independencia entre variables, que muchas veces no es real [22].

Para este trabajo se seleccionó particularmente el modelo CNB (*Complement Naive Bayes*), que es una adaptación del MNB (*Multinomial Naive Bayes*), pero que se adecúa mejor al desbalanceo entre clases. Por lo general, CNB supera a MNB en tareas de clasificación de texto [23].

Lo que hace el algoritmo es calcular, para cada clase, la probabilidad de que un documento no le pertenezca. Luego, la clase del documento será la clase con menor probabilidad, ya que aquí se quiere minimizar la probabilidad de no pertenecer.

La fórmula de CNB es la siguiente [24]

$$l(t) = \arg \min \sum_i t_i w_{ci} \quad (2.4)$$

donde  $t_i$  es la cantidad de veces que aparece el *token*  $i$  en el documento y:

$$w_{ci} = \frac{\log \hat{\theta}_{ci}}{\sum_k |\log \hat{\theta}_{ck}|} \quad (2.5)$$

siendo:

$$\hat{\theta}_{ci} = \frac{\alpha_i + \sum_{j: y_j \neq c} d_{ij}}{\alpha + \sum_{j: y_j \neq c} \sum_k d_{kj}} \quad (2.6)$$

donde las sumatorias son sobre todos los documentos  $j$  que no sean de la clase  $c$ ,  $d_{ij}$  es la cantidad de veces que aparece el *token* en el documento  $j$ ,  $\alpha_i$  es un hiperparámetro de suavizado (por lo general equivale a 1) y  $\alpha = \sum_i \alpha_i$ .

### 2.3.2. Representaciones de codificador bidireccional de transformadores

BERT (*Bidirectional Encoder Representations from Transformers*) es un modelo *open source* de PNL desarrollado por Google en el año 2018. El modelo original fue entrenado con texto de Wikipedia y el *dataset* BookCorpus de Google.

Inicialmente, dos modelos de BERT fueron presentados:

- *base*: con 12 capas, 12 cabezas de atención y 110 millones de parámetros.
- *large*: con 24 capas, 16 cabezas de atención y 340 millones de parámetros.

Cada capa de BERT es un *Transformer Encoder*. Cuando se habla de, por ejemplo, 12 capas, son 12 de esos codificadores apilados uno encima de otro. Los datos de entrada son consumidos por la primer capa, y van pasando por cada una hasta llegar a la última. Como salida, habrá un *embedding* donde cada posición del *token* tendrá un vector de 768 posiciones para BERT *base* y 1024 para BERT *large* [25].

La innovación introducida por BERT es aplicar dos técnicas en la etapa de entrenamiento: MLM (*Masked Language Model*) y NSP (*Next Sentence Prediction*).

MLM consiste en dar a BERT una secuencia para que optimice sus pesos internos y produzca la misma secuencia. Sin embargo, previamente a dar a BERT la secuencia, se enmascaran ciertos *tokens* para que BERT los tenga que “adivinar”. La implicancia de esta técnica es que BERT termina aprendiendo del contexto de la oración para predecirlos.

Para lograr esto, BERT se apoya en un mecanismo de atención propia o *self-attention* posibilitado por los *Transformers* bidireccionales en el núcleo del diseño de BERT. Esto es importante ya que el significado de una palabra puede cambiar su significado a medida que se desarrolla una oración. Cuantas más palabras haya en cada oración o frase, mas ambigua se vuelve la palabra en cuestión. BERT se encarga de esta ambigüedad leyendo una oración bidireccionalmente, teniendo en cuenta el efecto de todas las palabras en la oración y no sólo las que la preceden [26].

En la figura 2.2 se puede ver como para una palabra, el modelo de IA se hace una ponderación de la importancia del resto de las palabras en esa oración.

<sup>2</sup>Imagen tomada de <https://towardsdatascience.com/understand-self-attention-in-bert-intuitively-cd480cbff30b>

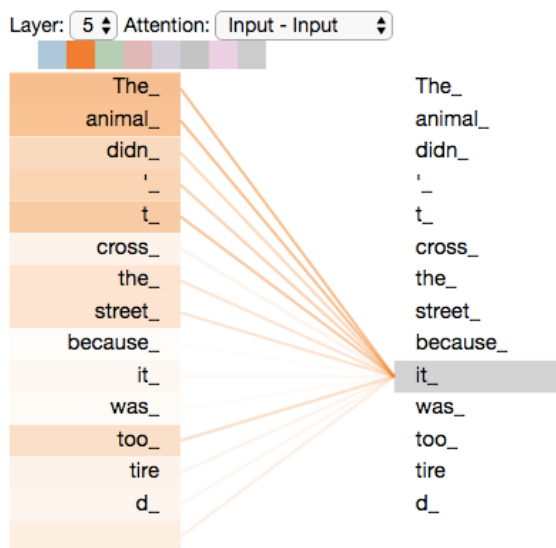


FIGURA 2.2. Ejemplo de funcionamiento del mecanismo de atención propia<sup>2</sup>.

NSP, por otro lado, es otro mecanismo en donde el modelo recibe oraciones de a pares como entrada y aprende a predecir si la segunda oración en el par es subsecuente de la primera. Durante el entrenamiento, la mitad de los datos son pares de oraciones subsecuentes y en la otra mitad la segunda oración es seleccionada de manera aleatoria.

Para saber distinguir cuándo comienza la primer oración y cuándo la segunda, la entrada es procesada de la siguiente forma:

1. Un *token* “[CLS]” se inserta al comienzo de la primer oración y un *token* “[SEP]” es insertado al final de las dos oraciones.
2. Se agrega un *embedding* indicando para cada *token* si corresponde con la oración A o B.
3. Se agrega un *embedding* posicional indicando para cada *token* su posición en la secuencia.

En la figura 2.3 se puede ver cómo se combinan estas dos técnicas cuando se entrena BERT.

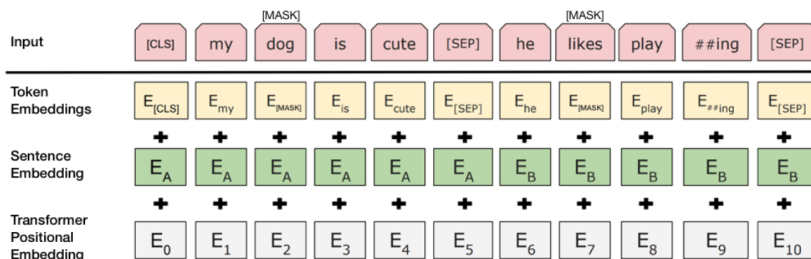


FIGURA 2.3. Ejemplo de entrada de datos a BERT con MLM y NSP combinados<sup>3</sup>.

<sup>3</sup>Imagen tomada de <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b627>



En la primera fila se observa como fueron separadas las dos oraciones. Luego en el *embedding* amarillo, se observa como algunos *tokens* son enmascarados para que luego la red los tenga que predecir. En el tercera, en verde, se observa la asignación de cada *token* a la oración A o B y por último, se observa un *embedding* con el orden de cada *token*.

## 2.4. Herramientas de software utilizadas

En esta sección se presentan las herramientas que se utilizaron tanto para la fase de entrenamiento como para el despliegue de los modelos.

### 2.4.1. Python

Python es un lenguaje de programación utilizado principalmente para desarrollo de aplicaciones y *machine learning*. Entre sus características se destaca que es orientado a objetos, multiplataforma e interpretado y su tipado es dinámico y fuerte.

### 2.4.2. GitHub

GitHub es un servicio web donde los usuarios pueden alojar repositorios con Git como sistema de control de versiones. Este sistema permite gestionar y rastrear cambios en el código a través del tiempo, ayudando a los equipos de desarrolladores a trabajar en proyectos de manera colaborativa.

### 2.4.3. BigQuery

BigQuery es un servicio *serverless* de *Data Warehouse* de GCP (Google Cloud Platform). Los datos se guardan en colecciones de tablas, que a su vez, se agrupan en *datasets*. Cada columna de las tablas se almacena por separado, por eso se dice que BigQuery es una base de datos columnar.

El acceso a los datos se hace a través del lenguaje SQL. Los recursos que se necesitan para ejecutar una consulta se calcula dinámicamente en base a las características de la misma y también en cómo esté configurada la tabla.

### 2.4.4. Vertex AI workbench

Vertex es un entorno de desarrollo especializado para ciencia de datos. Permite disponibilizar un *Jupyter notebook* que se integra con BigQuery y Google Cloud Storage para la lectura de datos y con GitHub para la sincronización de código. Además posee instancias con distintas configuraciones de CPU y GPU.

### 2.4.5. Docker

Docker es una herramienta que empaqueta *software* en unidades llamadas contenedores, que incluyen todas las dependencias para que el programa se ejecute: ejecutables, archivos de configuración, librerías, bibliotecas, etc. Funciona de manera similar a una máquina virtual, solo que además de virtualizar el *hardware* también virtualiza el sistema operativo.

La principal ventaja que otorga el uso de Docker es que asegura que el contenedor se pueda ejecutar de manera fiable en cualquier plataforma compatible, como Kubernetes.

#### **2.4.6. Artifact registry**

Artifact Registry permite almacenar y administrar imágenes de Docker. Posibilita armar un *pipeline* de integración y despliegue continuo al permitir cargar una imagen de Docker para que pueda ser consumida desde las distintas plataformas de orquestación de contenedores.

#### **2.4.7. Cloud Composer y Apache Airflow**

Cloud Composer es un servicio de organización de flujos de trabajo administrado por Google. Está basado en el proyecto *open source* Apache Airflow.

Un flujo de trabajo representa una serie de tareas para trabajar con datos. Estos flujos son creados mediante grafos acíclicos dirigidos o DAGs (*Directed Acyclic Graphs*). Los DAGs se crean con código en Python que especifica su estructura.

## Capítulo 3

# Diseño e implementación

En este capítulo se describe cómo se han utilizado las herramientas mencionadas en el capítulo 2 y sus integraciones. Se presenta la arquitectura de alto nivel completa y luego se describe detalladamente desde el consumo de datos hasta la salida del proceso y su almacenamiento.

### 3.1. Arquitectura del sistema

En la figura 3.1 se puede observar la arquitectura de alto nivel que integra las herramientas utilizadas en la fase de entrenamiento y evaluación de los modelos de IA con las herramientas utilizadas para el despliegue.

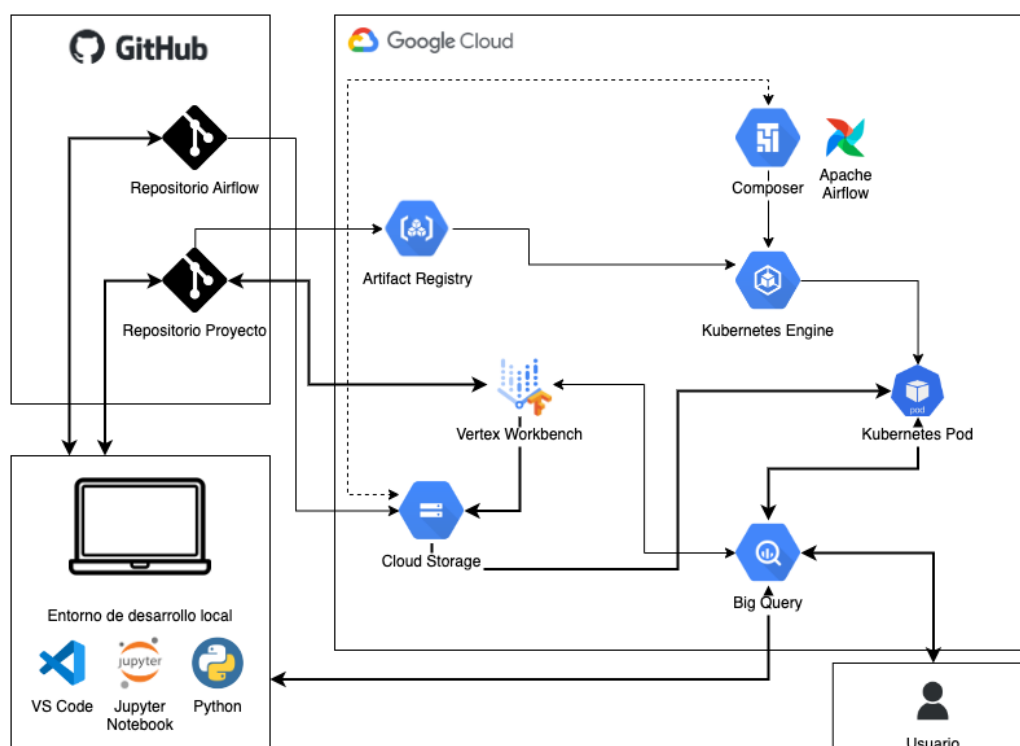


FIGURA 3.1. Diagrama de la arquitectura de alto nivel.

La mayor parte del desarrollo se realizó en una computadora local, que contaba con un intérprete de Python y Visual Studio Code con Jupyter para la escritura de código. Estos archivos se sincronizaban contra un repositorio de GitHub.

Los datos para realizar el entrenamiento de los modelos de IA fueron tomados del *data warehouse* BigQuery utilizando consultas SQL y bibliotecas de Google para Python para manejar su ejecución.

Estos datos son llevados a BigQuery por un proceso ajeno a este desarrollo, que se encarga de sincronizar la base de Salesforce hacia el *data warehouse* diariamente. Particularmente, hay una tabla que contiene los reclamos y consultas de los usuarios, con su identificador de caso, fecha de creación y estado.

En las ocasiones en las que fue necesario un mayor poder de cómputo a través del uso de una placa de vídeo dedicada, se utilizó una instancia de Vertex AI Workbench en la nube de Google. Por lo general, estas situaciones se dieron al momento de realizar distintas pruebas con las redes neuronales BERT. Los modelos eran luego exportados al almacenamiento de la nube llamado Cloud Storage en formato “Pickle” o “HDF5” (*Hierarchical Data Format 5*).

El repositorio del proyecto cuenta con un Workflow de GitHub Action que se corre manualmente y genera una imagen de Docker, incluyendo bibliotecas y librerías necesarias, archivos de código y de configuración. Esta imagen de Docker es enviada al Artifact Registry de la nube de Google y almacenada allí.

Por otro lado, se trabajó en otro repositorio separado, que es único para manejar todos los flujos de trabajo de Apache Airflow que tiene el área de *machine learning*. Este repositorio es exclusivo para la etapa de despliegue, y se agregó el DAG que administra el proceso de predicción.

El repositorio de Airflow cuenta con un Workflow de GitHub Action que sincroniza el código de los DAGs contra un Cloud Storage que está conectado a la instancia del orquestador. Airflow periódicamente controla si hay un flujo de trabajo nuevo y cuando lo detecta, lo carga.

Al momento de ejecutar el proceso, Airflow utiliza un operador dentro del DAG para conectarse con la plataforma de Kubernetes en instancia un Pod. Este Pod ejecuta los archivos de código del contenedor que se empaquetaron en la imagen de Docker cargada previamente en el Artifact Registry.

Al finalizar, se guardan los resultados de las predicciones en una tabla en BigQuery y el Pod es eliminado de Kubernetes. Los datos quedan disponibles en la base para que cualquier usuario con los suficientes permisos pueda consumirlos ejecutando una consulta a la base.

## 3.2. Extracción y preprocesamiento del texto

En esta sección se describen en detalle los pasos realizados para obtener los datos crudos y aplicarles un preprocesamiento para poder usarlos de entrada en el posterior entrenamiento de los modelos de IA.

### 3.2.1. Extracción de datos de BigQuery

Inicialmente se realizó un relevamiento de las tablas existentes, y en particular se encontró una tabla que contenía información de los reclamos y consultas.

Las columnas de interés de esa tabla fueron:

- El identificador del caso.

- El número del caso.
- La categoría L2 (de segundo nivel).
- La categoría L3 (de tercer nivel).
- El estado del caso.
- La descripción: el contenido del mensaje que escribió el usuario.
- La fecha de creación.
- La fecha de cierre.

Para la fase de entrenamiento, se limitó la extracción de casos a los que tenían fecha de creación entre enero de 2021 y diciembre de 2022. Además, su estado debía ser “Resuelto” o “Cerrado”. Sin embargo, por la poca cantidad de datos que había para algunos clasificadores L2, fue necesario realizar consultas adicionales para tomar casos de 2023. Una vez obtenidos los datos, se guardaron en archivos “Parquet” para su posterior uso con la librería Pandas para Python.

Por otro lado, el área de atención al cliente maneja un archivo CSV (*comma-separated values*) con la jerarquía de casos L1, L2 y L3. Es decir, para cada L1, qué casos L2 le corresponden, y para cada uno de esos L2, qué casos L3 le corresponden. Las categorías L3 fueron ignoradas para este trabajo, por lo que se hará foco en las primeras dos.

En total, se encontraron 15 categorías L1 y 122 categorías L2. Luego de una reunión con el área de atención al cliente para un mejor entendimiento de cada una, se descartó una categoría especial. Luego de este refinamiento quedaron 14 L1 y 117 L2.

En la figura 3.2 se puede observar para cada categoría L1 cuántas subcategorías L2 contiene.

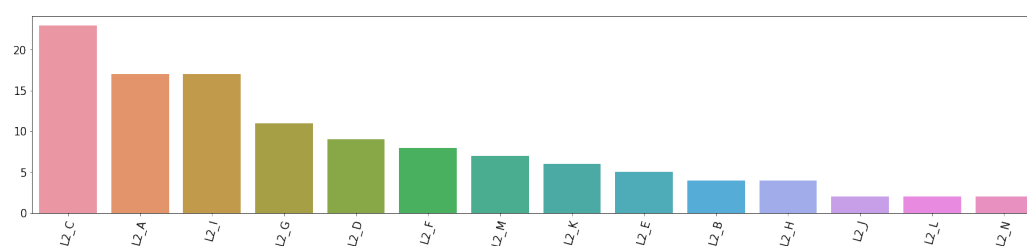


FIGURA 3.2. Cantidad de categorías L2 para cada L1.

Para trabajar en la fase de preprocesamiento, se combinó el archivo de los casos con el archivo que contenía la jerarquía de categorías, obteniendo así la L1 de cada reclamo y consulta.

### 3.2.2. Pipeline de preprocesamiento

El preprocesamiento es una de las tareas más importantes en PNL. Los datos en crudo pueden contener información que no es relevante para la tarea en cuestión, que se debe eliminar. Por otro lado, muchos modelos de IA requieren un formateo previo de los datos para poder consumirlos.

En la figura 3.3 se detallan los pasos realizados para “limpiar” el texto.

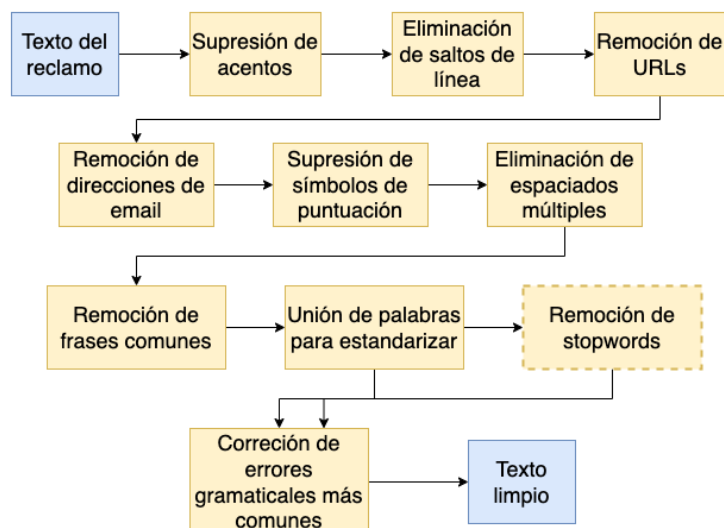


FIGURA 3.3. Diagrama del procesamiento del texto crudo.

El proceso comienza con el texto original del reclamo tal cual quedó guardado en la base. Luego se le aplican las siguientes acciones:

1. Se suprimen los acentos de las palabras (si los hubiera) utilizando una expresión regular.
2. Se eliminan los saltos de línea.
3. Se sacan URLs en el mensaje en caso de que hubiese hipervínculos.
4. Se sacan direcciones de *email*. Estos por lo general aparecen cuando el reclamo se hace a través de ese medio.
5. Se suprimen signos de puntuación utilizando otra expresión regular.
6. Se eliminan espacios múltiples para normalizar en un espacio entre dos palabras.
7. Se remueven frases que se repiten en muchos reclamos: algunos casos quedan registrados en la base con la respuesta por parte de la empresa. Estos mensajes tienen una plantilla de base con frases que luego se repiten entre casos.
8. Se unen palabras con un significado específico: por ejemplo “pedidos ya” se deja como “pedidosya” porque hace referencia a una empresa.
9. Para el caso de los modelos de CNB entrenados, se aplicó un paso extra para eliminar *stopwords*.
10. Por último se realizó un análisis con los datos de entrenamiento para ver los errores gramaticales más comunes utilizando la librería PyEnchant para Python. De esto se obtuvo un listado con los errores más comunes y un mapeo a su versión correcta. En este paso se aplica la corrección sobre ese listado de palabras.

Una vez obtenido el texto limpio, se procede a pasarlo a un vector numérico para que sea interpretado por los modelos de IA. Se distinguen dos formas de hacerlo, dependiendo de si se usa el modelo de CNB o el de BERT.

### Codificación para CNB

En la figura 3.4 se explica el proceso para obtener los vectores numéricos para entrenar el modelo de CNB. Al texto limpio se le aplica un paso de segmentación para obtener los *tokens* que luego serán vectorizados utilizando el método TF-IDF. Por otro lado, a la variable objetivo que también está en formato de texto, se le aplica el proceso de *label encoding* para obtener una variable numérica.

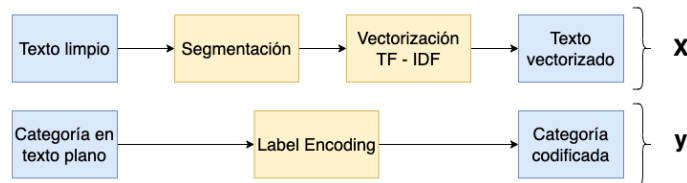


FIGURA 3.4. Diagrama de codificación para CNB.

### Codificación para BERT

En la figura 3.5 se puede ver el proceso para el caso de BERT. Se puede notar que es un poco mas largo que el de CNB, pero en realidad, todos esos pasos quedan encapsulados en una sola función del BertTokenizer de la librería Transformers de Hugging Face para Python.

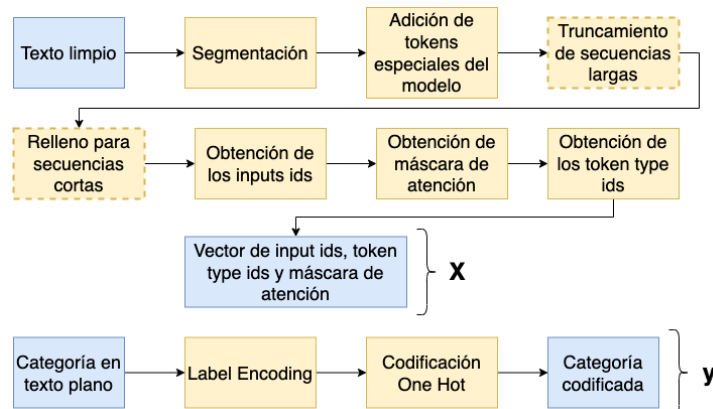


FIGURA 3.5. Diagrama de codificación para BERT.

El vocabulario del modelo pre-entrenado de BERT que se utilizó está compuesto por [27]

- 977 palabras reservadas por *tokens* del estilo “[MASK]”, “[PAD]”, etc.
- *Tokens* de caracteres individuales (números y letras).
- *Tokens* de subpalabras.
- *Tokens* de palabras.

- 3.3. Entrenamiento de los modelos**
- 3.4. Desarrollo de GitHub Actions**
- 3.5. Empaquetamiento del código y artefactos**
- 3.6. Desarrollo del *pipeline* de predicción**
- 3.7. Almacenamiento de predicciones**



## Capítulo 4

# Ensayos y resultados

### 4.1. Pruebas funcionales del hardware

La idea de esta sección es explicar cómo se hicieron los ensayos, qué resultados se obtuvieron y analizarlos.



## Capítulo 5

# Conclusiones

### 5.1. Conclusiones generales

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

Algunas preguntas que pueden servir para completar este capítulo:

- ¿Cuál es el grado de cumplimiento de los requerimientos?
- ¿Cuán fielmente se pudo seguir la planificación original (cronograma incluido)?
- ¿Se manifestó algunos de los riesgos identificados en la planificación? ¿Fue efectivo el plan de mitigación? ¿Se debió aplicar alguna otra acción no contemplada previamente?
- Si se debieron hacer modificaciones a lo planificado ¿Cuáles fueron las causas y los efectos?
- ¿Qué técnicas resultaron útiles para el desarrollo del proyecto y cuáles no tanto?

### 5.2. Próximos pasos

Acá se indica cómo se podría continuar el trabajo más adelante.



# Bibliografía

- [1] Tiendanube. *Atención al cliente: qué es y claves para mejorar tu servicio*. <https://www.tiendanube.com/blog/que-es-servicio-atencion-cliente/>. Mar. de 2023. (Visitado 01-07-2023).
- [2] HubSpot. *¿En qué se diferencian el servicio al cliente y la atención al cliente?* <https://blog.hubspot.es/service/diferencia-servicio-cliente-y-atencion-cliente>. Ene. de 2023. (Visitado 01-07-2023).
- [3] HubSpot. *¿Qué hace un gerente de servicio al cliente?* <https://blog.hubspot.es/service/gerente-servicio-cliente>. Ene. de 2023. (Visitado 01-07-2023).
- [4] Zendesk. *Manual para un departamento de atención al cliente exitoso*. <https://www.zendesk.com.mx/blog/manual-de-funciones-de-servicio-al-cliente/>. Abr. de 2023. (Visitado 01-07-2023).
- [5] HubSpot. *Qué es el proceso de atención al cliente y cuáles son sus fases clave*. <https://blog.hubspot.es/service/proceso-atencion-cliente>. Abr. de 2023. (Visitado 01-07-2023).
- [6] Zendesk. *Entienda lo que son las fases del proceso de atención al cliente y en que puedes ayudar tener ese proceso interno en una empresa*. <https://www.zendesk.com.mx/blog/fases-del-proceso-de-atencion-al-cliente/>. Jun. de 2020. (Visitado 01-07-2023).
- [7] Smart Tribune. *The Importance of Customer Experience*. <https://blog.smart-tribune.com/en/importance-of-customer-experience>. Sep. de 2021. (Visitado 01-07-2023).
- [8] Zendesk. *CX Trends 2023*. <https://cxtrends.zendesk.com/mx>. Feb. de 2023. (Visitado 01-07-2023).
- [9] HubSpot. *Qué es la atención al cliente, elementos clave e importancia*. <https://blog.hubspot.es/service/que-es-atencion-al-cliente>. Abr. de 2023. (Visitado 01-07-2023).
- [10] Zendesk. *5 examples of bad customer service (and how to be great instead)*. <https://www.zendesk.com/blog/what-is-bad-customer-service/>. Mayo de 2023. (Visitado 01-07-2023).
- [11] SuperOffice. *32 CUSTOMER EXPERIENCE STATISTICS YOU NEED TO KNOW FOR 2023*. <https://www.superoffice.com/blog/customer-experience-statistics/>. Feb. de 2023. (Visitado 01-07-2023).
- [12] HubSpot. *8 ejemplos de mal servicio al cliente (y cómo evitarlos)*. <https://blog.hubspot.es/service/mal-servicio-cliente>. Ene. de 2023. (Visitado 02-07-2023).
- [13] Zendesk. *8 problemas comunes en servicio al cliente y cómo resolverlos*. <https://www.zendesk.com.mx/blog/problemas-comunes-con-clientes/>. Feb. de 2021. (Visitado 02-07-2023).
- [14] Piperlab. *Cómo hemos llegado hasta ChatGPT y el resto de LLMs*. <https://piperlab.es/2023/03/29/como-hemos-llegado-hasta-chatgpt-y-el-resto-de-llms/>. Mar. de 2023. (Visitado 02-07-2023).

- [15] Yoshua Bengio, Réjean Ducharme y Pascal Vincent. «A Neural Probabilistic Language Model». En: (2003).
- [16] Tomas Mikolov y col. «Efficient Estimation of Word Representations in Vector Space». En: (2013).
- [17] Jeffrey Pennington, Richard Socher y Christopher D. Manning. «GloVe: Global Vectors for Word Representation». En: (2014).
- [18] Ashish Vaswani y col. «Attention Is All You Need». En: (2017).
- [19] TechTarget. *5 examples of effective NLP in customer service*. <https://www.techtarget.com/searchenterpriseai/feature/5-examples-of-effective-NLP-in-customer-service>. Feb. de 2021. (Visitado 03-07-2023).
- [20] Tacos de datos. *Introducción al análisis de texto*. <https://old.tacosdedatos.com/analisis-texto>. Ago. de 2020. (Visitado 07-07-2023).
- [21] freeCodeCamp. *How to process textual data using TF-IDF in Python*. <https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/>. Jun. de 2018. (Visitado 08-07-2023).
- [22] freeCodeCamp. *Cómo funcionan los clasificadores Naïve Bayes: con ejemplos de código de Python*. <https://www.freecodecamp.org/espanol/news/como-funcionan-los-clasificadores-naive-bayes-con-ejemplos-de-codigo-de-python/>. Abr. de 2021. (Visitado 08-07-2023).
- [23] Jason Rennie y col. «Tackling the Poor Assumptions of Naive Bayes Text Classifiers». En: (2003).
- [24] Scikit-Learn. *Complement Naïve Bayes*. [https://scikit-learn.org/stable/modules/naive\\_bayes.html](https://scikit-learn.org/stable/modules/naive_bayes.html). (Visitado 08-07-2023).
- [25] Analytics Vidhya. *Why and how to use BERT for NLP Text Classification?* <https://www.analyticsvidhya.com/blog/2021/06/why-and-how-to-use-bert-for-nlp-text-classification/>. Jun. de 2021. (Visitado 11-07-2023).
- [26] TechTarget. *BERT language model*. <https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model>. (Visitado 10-07-2023).
- [27] Omar Espejel. *BETO (BERT) 01 : Importación y tokenizing*. <https://espejel.substack.com/p/beto-bert-01-importacion-y-tokenizing>. (Visitado 15-07-2023).