Estudio e implementación de un modelo de procesamiento de lenguaje natural que analice la satisfacción de compra mediante el análisis de comentarios de usuarios

Autor:

Julián David Longas Arteaga

Documento para optar al título de ingeniero electrónico

Asesores:

Hernán Felipe García Arias, PhD

Eduardo Enrique Morales Martínez, Ingeniero industrial

Universidad de Antioquia

Facultad de ingeniería

Ingeniería electrónica

Medellín, Antioquia

2022

Cita	(Longas Arteaga, 2022)
Referencia **Estilo APA 7 (2020)**	Longas Arteaga, J. (2022). *Estudio e implementación de un modelo de NPL que analice la satisfacción de compra mediante el análisis de comentarios de usuarios* [Semestre de industria]. Universidad de Antioquia, ciudad UdeA.

Agradecimientos

Agradezco a la empresa C.I HERMECO S.A y a la empresa OCXCI S.A.S con su proyecto 7Bytes por la confianza prestada y por brindarme las herramientas y el tiempo para realizar este proyecto.

Resumen:

En la actualidad la opinión del cliente es muy importante para valorar una empresa, conocer de forma rápida las opiniones del cliente se ha convertido un reto para que las empresas sigan creciendo y realizando las transformaciones que exige el mercado. Una de las herramientas más usadas en la actualidad para conocer la satisfacción del cliente es el procesamiento de lenguaje natural y la inteligencia artificial, con estas herramientas se pueden usar modelos previamente entrenados para un idioma y analizar la satisfacción de compra de un cliente hacia una empresa. Para la empresa C.I HERMECO S.A es muy importante conocer que piensan sus clientes de la empresa y por esto se ha implementado un modelo de procesamiento de lenguaje natural llamado BERT que ha sido entrenado con comentarios reales de clientes reales teniendo como resultado un modelo especialmente creado para su empresa y para el lenguaje que usan sus clientes.

CONTENIDO

INTRODUCCIÓN

En la actualidad la inteligencia artificial ha sido implementada en muchas tecnologías debido al crecimiento continuo de el desarrollo de computadores más rápidos y capaces de recibir gran cantidad de datos, por este motivo, muchas empresas han visto como prioridad usar modelos de *machine learning* en sus procesos. Podemos definir la inteligencia Artificial como la capacidad de las máquinas para usar algoritmos, aprender de los datos y utilizar lo aprendido para hacer toma de decisiones [1].

Uno de los motores más importantes de la empresa OCXCI S.A.S, la cual se encuentra prestando servicios a la empresa C.I HERMECO S.A, es utilizar métodos de aprendizaje supervisado basados en el lenguaje natural para medir la satisfacción y aprecio que tiene un usuario con una marca. Esta empresa realiza, entre otras cosas, encuestas de satisfacción con preguntas dinámicas para tener una visión global de las opiniones de los usuarios y así realizar cambios oportunos a problemas específicos en tiendas, campañas y servicios realizando así una comunicación más efectiva entre usuario y marca.

Actualmente la empresa OCXCI S.A.S utiliza el modelo que presta Google llamado Cloud Natural Language y más precisamente la API de Natural Language. Este modelo se basa en el autoaprendizaje continuo y es usado por compañías grandes como Paypal y HSBC

Uno de los desarrollos que la empresa OCXCI S.A.S piensa implementar en el futuro es entrenar un modelo de inteligencia artificial con los datos que actualmente tiene la empresa, donde se busca recolectar las respuestas que se tienen actualmente de usuarios reales, clasificarla, para luego entrenar un algoritmo y de esta manera tener un modelo que se base en las respuestas que son principalmente usadas en Colombia.

Para implementar este modelo se va a hacer uso del *procesamiento del lenguaje natural (NPL),* con el que se busca conocer el sentimiento de las personas a partir de frases que se puedan encontrar en textos. Con el análisis de sentimientos se puede lograr encontrar la polaridad del texto y con este se puede determinar si el texto va a ser positivo, negativo o neutral. Este análisis tiene diversas dificultades que pueden ser el tono del lenguaje neutral o el sarcasmo.

Uno de los estudios que abrieron un nuevo horizonte de posibilidades en el campo del procesamiento del lenguaje natural fue *BERT (pre-training of Deep bidirectional Transformers for language understanding)* publicado en el 2018 por Google, este modelo introduce al modelado del lenguaje un entrenamiento bidireccional de transformadores [2]. Con *BERT* se abre la posibilidad de utilizar varios idiomas y, además, con su codificación, permite leer secuencias enteras a diferencia de otros modelos bidireccionales tradicionales, pudiendo, de esta forma, conocer mejor el contexto de la palabra. Este modelo lo usa Google en sus búsquedas para hacer control y generar búsquedas más rápidas [3].

OBEJTIVOS

Objetivo general

Implementar un modelo de procesamiento de lenguaje natural que realice un análisis de satisfacción contextualizado al lenguaje coloquial colombiano usando una arquitectura *BERT* como análisis de sentimientos con el fin de conocer la experiencia de compra del cliente y medir sus opiniones en tiempo real.

Objetivos específicos

1. Realizar el depurado y etiquetado de los datos que actualmente se encuentran en la base de datos de la empresa OCXCI administrados por Google Natural language haciendo uso del lenguaje de consulta estructurado para generar el archivo CSV con el que se va a realizar el entrenamiento del modelo.

2. Entrenar del modelo *BERT* con los datos previamente etiquetados usando Google Colab e implementarlo en Google Cloud Console para medir la satisfacción de usuarios reales en tiempo real.

3. Evaluar la ejecución del modelo realizando métricas de exactitud, sensibilidad, f1score y AUC para medir la eficiencia y fiabilidad de la arquitectura *BERT* con los datos tomados como entrenamiento ante respuestas reales en tiempo real.

MARCO TEÓRICO

Procesamiento de lenguaje natural (NPL):

Es un campo de la ciencia de la computación, de la inteligencia artificial y de la lingüística que principalmente estudia las interacciones máquina y usuario. El procesamiento del lenguaje natural involucra una transformación a una representación formal, manipula esta transformación y, por último, si es necesario, lleva estos resultados al lenguaje natural [4].

Actualmente el procesamiento de lenguaje natural se trabaja en marcas que necesiten conocer la opinión del cliente mediante comentarios. Por ende, se ha estado trabajando tanto *chatbots* como en *callbots* usando análisis de sentimientos.

***Análisis de sentimientos*:**

El análisis de sentimientos o también llamado minería de opiniones es un proceso que involucra el uso de herramientas de *NPL* y software de análisis de textos para automatizar el proceso.

Dicho análisis se basa en el aprendizaje para evaluar emociones tanto positivas como negativas de un corpus etiquetado cuya forma básica es una clasificación polarizada de sentimientos que se asignan en un rango de -10 hasta 10 [4].

Muchos de los modelos actualmente usados usan bases de datos de comentarios en redes sociales de internet y se usan para analizar gramaticalmente oraciones o descomponer textos. Con estos modelos se conocen y se resumen grandes volúmenes de texto de opiniones de marca que puedan ayudar en visibilidad de estas o en la imagen de esta.

GPU (Graphics Processing Unit)

Es un procesador dedicado exclusivamente al procesamiento gráfico, debido a que las gráficas en 3 dimensiones requieren mucho cálculo y operaciones aritméticas, por ende, este procesador libera de trabajo al procesador principal. Las *GPU* son fundamentales para el entrenamiento de redes neuronales ya que una *GPU* potente disminuye considerablemente el tiempo de entrenamiento de la red neuronal.

Actualmente se están creando *GPU's* exclusivas para el trabajo con inteligencia artificial y el internet de las cosas, haciendo de esta manera que se puedan procesar cada vez más datos en menos tiempo [4].

Tensores

Se define como una entidad algebraica que es utilizada en la inteligencia artificial para crear redes neuronales con *TensorFlow* (librería desarrollada para implementar redes neuronales orientadas al aprendizaje automático). Estos métodos algebraicos requieren un procesamiento matricial masivo y por ende se usan unidades de procesamiento de tensores *TPU* el cual es un circuito *ASIC* (application specific integrated circuit) que principalmente usa Google para implementar redes neuronales [4].

Word2vec

Esta es una técnica que se usa para realizar procesamiento de lenguaje natural publicada en el 2013. Este algoritmo utiliza un modelo de red neuronal para aprender asociaciones entre palabras y luego, de ser entrenado, este modelo puede detectar palabras sinónimas o sugerir palabras adicionales [5].

Esta técnica usa dos algoritmos diferentes:

•*Skip-gram model*: Con este algoritmo se pasa de una palabra a un vector con valores numéricos que sean útiles para encontrar palabras similares a la palabra ingresada. Como se puede apreciar en la figura 1, una palabra de entrada w(t) se encuentra en un espacio de palabras desde w(t + n) hasta w(t - n) con las que se pueden reemplazar.

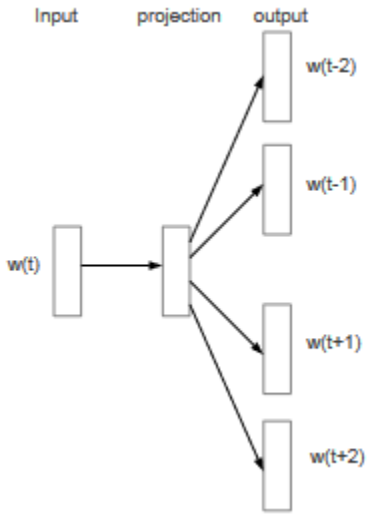

Figura 1: *Algoritmo Skip-gram model.* [5]

•Continious bag of words: Con este algoritmo se buscan los vectores similares a la palabra que se ingresó con los vectores que se encuentran en el dataset (documento de preentrenamiento) y de esta manera se busca predecir su contexto y por lo tanto predecir con que palabra puede ser cambiada. En la figura 2 se puede observar lo contrario al algoritmo skip-gram model, en este se toma del espacio de palabras la palabra por la cual se puede permutar la palabra de entrada.

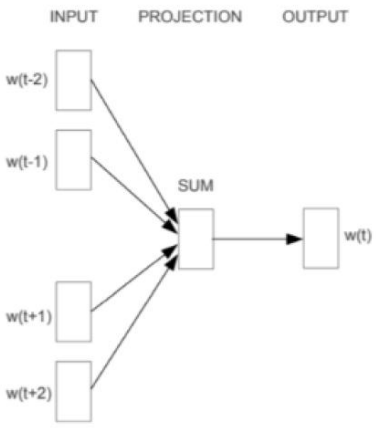

Figura 2: *Algoritmo Continious bag of words.*

El objetivo que se busca al usar Word2Vec es la agrupación de vectores de palabras similares en un espacio vectorial. Teniendo un vector muy grande, con varios espacios vectoriales diferentes, se pueden calcular similitudes de palabras usando la función coseno.

Esta similitud se representa como el valor del coseno en función de los grados, si no existe ninguna similitud indica que los vectores son perpendiculares y en el caso que sean idénticas indica que los vectores de palabras son paralelos [5].

Redes transformer:

Son descritas en el artículo *Attention is all you need* publicado por Google en el 2017 y en este se expone una solución al problema de la traducción de textos de un idioma a otro. Estas redes, a diferencia de las redes recurrentes usadas en la traducción de textos, se procesan en paralelo [6]. Su arquitectura es la siguiente:

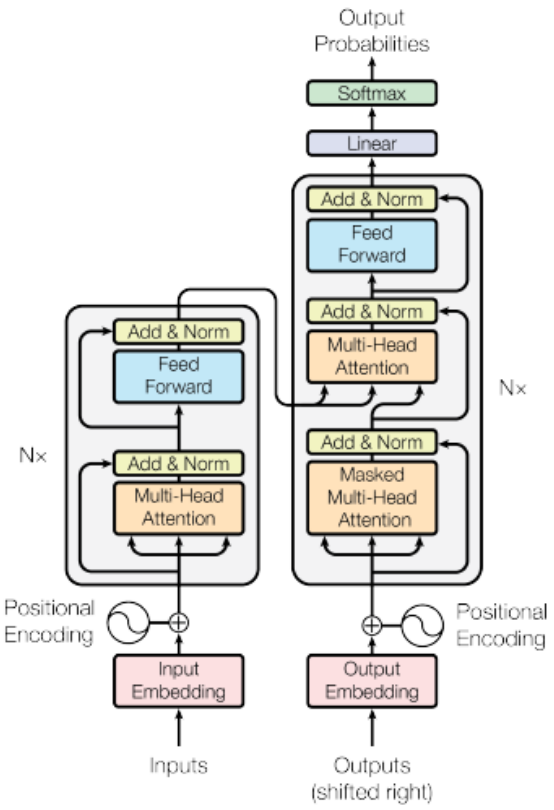

Figura 3: Arquitectura de una red transformer. [6]

En primer lugar, como se puede observar en la figura 3, las palabras de entrada pasan por el bloque *inputs embedding*. En este bloque se usa un algoritmo que convierte el texto de entrada en una serie de vectores o tokens. La salida de este bloque es una representación numérica que va a ser comprendida por la red. En la figura 4 se puede observar que cada vector de palabras, al pasar por el bloque *inputs embedding*, se intercambia por un vector o token específico.

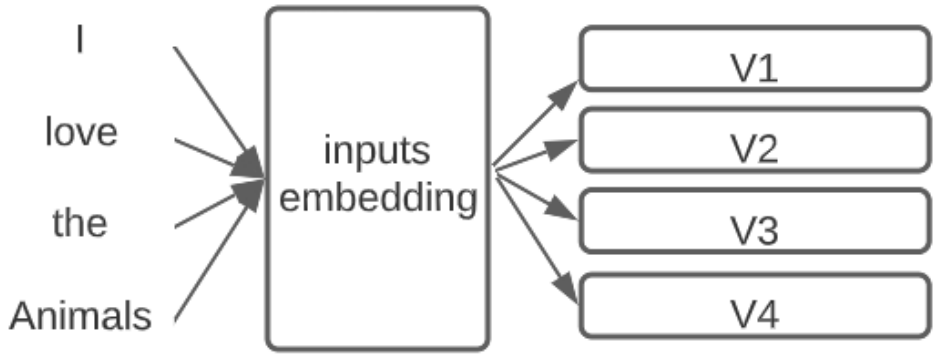

Figura 4: Bloque Inputs embedding.

El siguiente bloque es *Positional Encoding* y es el encargado de indicarle a la red el orden en el que se encuentran las palabras dentro del texto.

Como se puede observar en la figura 5, este bloque genera un vector de posición que serán sumados a los vectores de palabras. Para realizar las sumas se usan funciones senoidales para las posiciones pares y cosenoidales para las posiciones impares esto para que cada vector generado tenga un patrón único con la información de la posición.

Es necesario realizar la codificación de posición debido a que los procesos se realizan en paralelo.

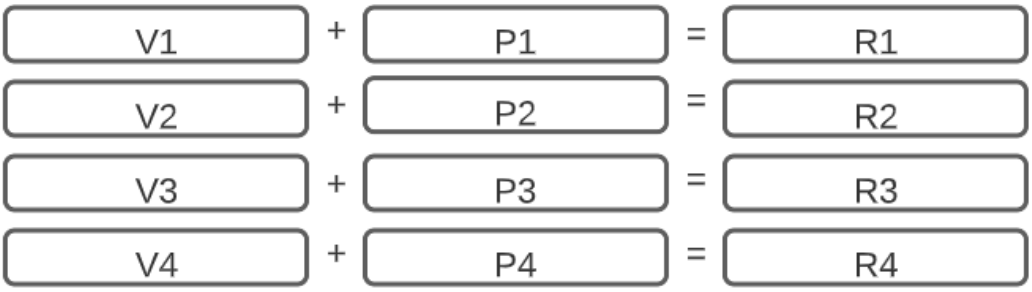

Figura 5: Bloque *Positional Encoding*.

El bloque siguiente es un codificador atencional, este aparece en la figura 3 como *Nx* debido a que el proceso se puede realizar *N* veces, y está conformado por el bloque atencional (*Multi-Head Attention*), una red neuronal (*Feed Forward*) y un bloque de normalización (*Add and norm*).

El bloque atencional (*Multi-Head Attention*) se encarga de analizar la totalidad de la secuencia de entrada y encontrar las relaciones entre las palabras de la secuencia. Con el texto “I love the animals”, tenemos dos posibles asociaciones entre palabras: el verbo “love” y el sujeto “I” y el sustantivo “Animals” y el artículo “the”. Además de la asociación de la frase completa.

Lo que hace el bloque atencional es expresar numéricamente las relaciones existentes y luego codifica cada una de esas relaciones con la información de contexto, de esta forma indica a que elementos del texto se deben prestar más atención.

Para lograr expresar las relaciones del texto los vectores de palabras se llevan a tres redes neuronales, para calcular los vectores *query, key y value*. Luego se realiza la multiplicación de los vectores *query y key* donde se mide el grado de asociación entre pares de palabras. Con este resultado se ponderan los vectores *values* indicando la importancia de la palabra al momento de la codificación. En la figura 6 se puede observar el diagrama de flujo de este codificador atencional.

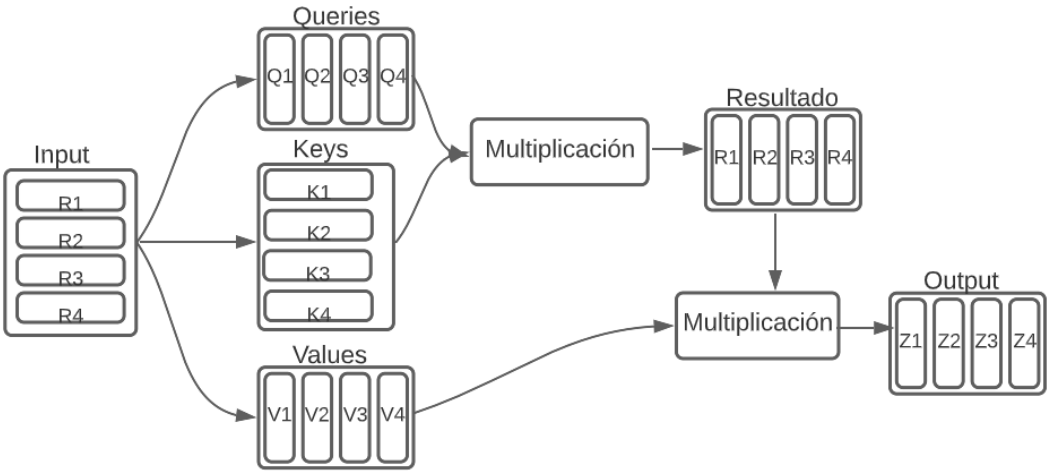

Figura 6: Bloque atencional.

Dado que hay más de una asociación en la frase original podemos decir que un bloque de atencional no es suficiente, además al usar múltiples bloques atencionales es posible detectar y codificar asociaciones a diferentes niveles. En la figura 7 observamos que una de las ventajas que tienen las *redes transformer* la cual es la paralelización del bloque atencional, esto para tener múltiples posibles asociaciones entre palabras.

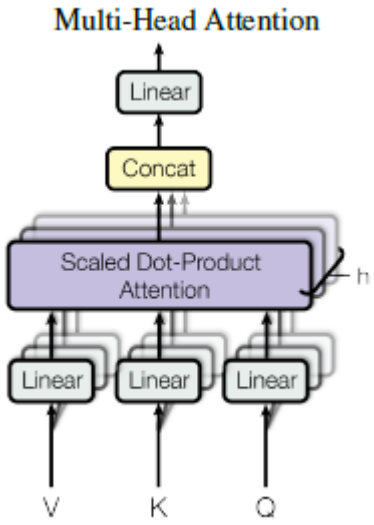

Figura 7: Paralelización del bloque atencional. [6]

El bloque de normalización (*Add and norm*) está en varias partes del algoritmo debido a que la red es tan profunda que la información puede degradarse progresivamente, por ende, en este bloque, se suma la entrada del bloque atencional (*Multi-Head Attention*) y se normaliza.

En la parte de decodificación se realiza un proceso inicial similar ya que los vectores codificados pasan por los bloques *Output Embedding* y *Positional Encoding* los cuales, como se dijo anteriormente, realizan una representación vectorial

y además los organizan antes de pasar por el decodificador atencional que se va a realizar *Nx* veces como indica la Figura 3.

El decodificador atencional se diferencia del codificador atencional en el bloque de atención con enmascaramiento (*Masked Multi-Head Attention*) en el cual se realiza un proceso similar al bloque de atención (*Multi-Head Attention*) con la diferencia que pasa todos los vectores de palabras anteriormente codificados y enmascara con ceros los valores atencionales más bajos para dejar los puntajes atencionales más altos.

Luego de salir del decodificador atencional se pasa a un bloque *Linear*, el cual es una red neuronal a la que se le ingresa el vector producido del decodificador atencional y lo transforma en un vector mucho más grande. La longitud de este vector depende de las palabras con las que fue pre entrenada la red, si esta aprendió 10000 palabras la salida será un vector de 10000 elementos.

Por último, el bloque *softmax* toma cada elemento y lo convierte en probabilidades. Haciendo que la posición con mayor probabilidad sea la palabra correcta.

BERT (pre-training of Deep bidirectional Transformers for language understanding)

BERT es un modelo del procesamiento del lenguaje natural con el cual se busca realizar un análisis de sentimientos a textos. Este modelo fue publicado en el 2018 por Google y está pre entrenado con páginas de Wikipedia como por libros de Google books [3]. Actualmente soporta más de 102 idiomas.

Lo que propone BERT es un codificador que obtiene representaciones bidireccionales a partir de *redes transformer* y se enfoca en entender el lenguaje en general. Es un modelo que viene pre entrenado con la totalidad de Wikipedia y book corpus y además se pueden agregar capas adicionales que permiten que el modelo se especialice en una tarea en particular. Con este modelo se usa una *red transformer* para obtener una representación numérica que permita su correcta interpretación y a esta red se le pueden usar entre 12 y 24 codificadores que se verán reflejados en la velocidad y eficacia del procesamiento.

Antes de pasar el texto por las *redes transformer* se debe tener una representación de entradas que representen a una única oración o parejas de oraciones en secuencias de tokens. Como se ve la figura 8, el primer token de cada secuencia debe ser un token especial de clasificación *CLS* y para las separaciones entre oraciones se utiliza el token especial *SEP*. Posteriormente se encuentran los *embeddings* de cada palabra, se le suma la posición y además se le suma un segmento que indica a que frase pertenece cada palabra.

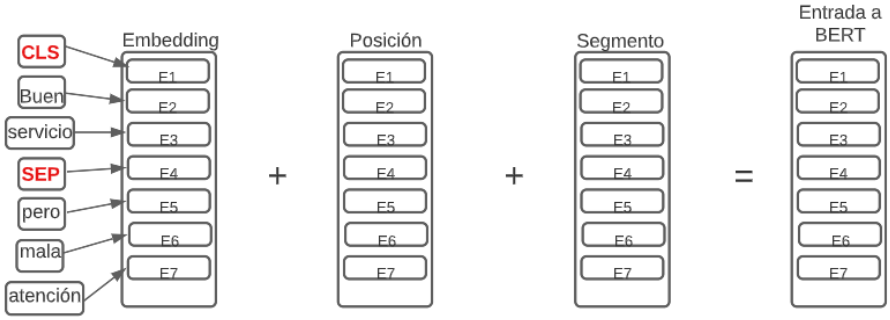

Figura 8: Representación de entradas a BERT.

Para el proceso de preentrenamiento se debe tener en cuenta que *BERT* analiza el texto de manera bidireccional, es decir que aprende a codificar cada palabra teniendo en cuenta todo su contexto, para esto realiza la tarea *nex sentence prediction (NSP).*

Muchas de las funciones que puede realizar *BERT* son preguntas y respuestas (*QA: Question and answer*) o inferencia de lenguaje natural que se basa en la relación que tienen dos oraciones. En el proceso de preentrenamiento se entrena en la tarea de predicción de la siguiente oración. El 50% de los casos, cuando se eligen dos oraciones, la segunda oración (después del token *SEP*) es realmente la oración consecutiva y se indica con la etiqueta *"IsNext"* mientras que el resto de los casos es la etiqueta *"NoNext"* que indica que la siguiente oración no es la que seguía.

Para que el modelo cumpla con funciones específicas se debe pasar por la etapa de afinación. En esta etapa, al final del modelo pre entrenado, se agregan dos pequeñas capas: una red neuronal y una capa *softmax* para obtener valores cuantitativos. En esta etapa se deben ingresar, si se necesitan, parámetros propios del problema que no fueron tenidos en cuenta en la etapa de preentrenamiento.

BETO:

En los últimos años el campo del procesamiento de lenguaje natural ha ido creciendo desde que se publicaron estudios sobre la autoatención (*self-atention*) y las *redes transformer*. Con esto y con modelos, ya previamente explicados como *BERT,* se han estado creando modelos pre entrenados para diferentes idiomas [7]. En un inicio *BERT* se publicó como un modelo pre entrenado que usaba *redes transformer* para dos idiomas diferentes, el inglés y el mandarín. Luego se publicó un modelo pre entrenado para todos los idiomas, *mBERT*, el cual incluye simultáneamente 100 idiomas incluyendo el idioma español [8]. Por último, se creó *BETO*. Este modelo pre entrenado nace con la necesidad de crear un modelo pre entrenado para uno de los idiomas más hablados en el mundo, además de este modelo nacieron modelos para el ruso, portugués, francés, etc [7]. Este modelo tiene la misma arquitectura que *BERT*, pero se pre entrenó completamente usando todo el corpus sin etiquetar de Wikipedia en español, además de todas las fuentes del Proyecto OPUS que estuviesen en español [9].

Como se puede observar en la tabla 1, *BETO* resultó tener un rendimiento superior a *mBERT* en la mayoría de las tareas estudiadas y comparables. En la tabla 1 se muestra el rendimiento de diferentes modelos entre *mBERT* y *BETO cased*, que quiere decir que se usó *BETO* y *mBERT* con un *embedding* que diferencia las mayúsculas en las palabras.

Tabla 1: Comparación entre los modelos pre entrenado mBERT y BETO. [8]

Modelo	XNLI	PAWS-X	NER	POS	MLDoc
mBERT	78.50[a]	89.00[b]	87.38[a]	97.10[a]	**95.70[a]**
BETO cased	**82.01**	**89.05**	**88.43**	**98.97***	95.60

En la tabla 1 se encuentran los resultados de exactitud promedio de los dos modelos pre entrenados, este estudio fue realizado a partir de los valores de exactitud al realizar las siguientes tareas [8].

Natural Language Inference XNLI: Este es el conjunto de datos Multi Genre Natural Language Interference (MNLI) es un conjunto de pares de frases. La primera oración se denomina premisa y la segunda hipótesis, la tarea consiste en predecir si la premisa implica la hipótesis.

Paraphrasing PAWS-X: La tarea consiste en determinar si dos oraciones son semánticamente equivalentes.

Name Entity Recognition NER: Esta tarea consiste en determinar en una frase nombres de personas u organizaciones, lugares, tiempo y cantidades.

Part-of-Speech POS: con esta tarea solo se estudia una parte gramatical de las frases y tiene como objetivo encontrar las figuras gramaticales embebidas en la oración.

Document Classification MLDoc: Con esta tarea se busca clasificar las frases o documentos entre CCAT (Corporativo/Industrial), ECAT (Economía), GCAT (Gobierno/Social), y MCAT (marcas).

METODOLOGÍA:

Para implementar el modelo de NPL se va a realizar la metodología propuesta por las publicaciones de *Collarte González, Procesamiento del lenguaje natural con BERT: Análisis de sentimientos en tuits* [3], y *Spanish pre-trained bert model and evaluation data* [8] de *Canete* y compañía.

Estas metodologías proponen usar un modelo pre entrenado basado en *redes transformer*, llamado *M*, para luego afinarlo con una data propia y así tener un modelo para dar solución a un problema específico [8].

Antes de comenzar el preentrenamiento y afinación del modelo se debe realizar una depuración de los datos:

Depuración y base de datos:

La empresa OCXCI, la cual trabaja para C.I HERMECO S.A, utiliza como base de datos un modelo relacional el cual se encuentra ubicado en Google Cloud Platform. Esta base de datos esta codificada en formato UFT-8 y fue creada mediante lenguaje de consulta estructurado (SQL) administrado por PostgreSQL.

Las respuestas a las preguntas son almacenadas en una tabla de la base de datos de la empresa C.I HERMECO S.A en la cual almacenan las respuestas a las preguntas realizadas luego de cada compra, además de otros canales que se usan para dicho fin.

Los canales de que actualmente tiene la empresa C.I HERMECO S.A en la base de datos de OCXCI S.A.S se encuentran ubicados en la tabla *survey*. A cada uno de estos canales se le asignan preguntas que varían respecto a los usuarios que van a consumir esos canales y son almacenadas en la tabla *survey_item* y sus respectivas opciones en la tabla *survey_item_option.*

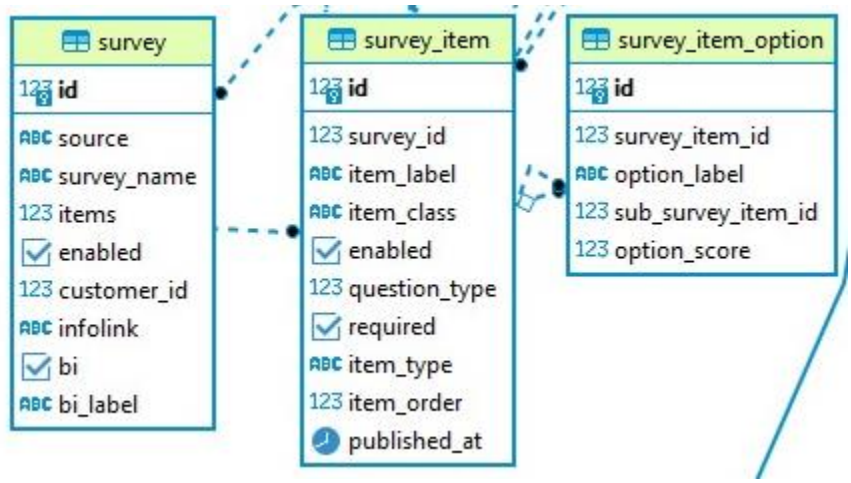

Figura 9. Diagrama relacional entre las tablas survey, survey_item y survey_item_option.

En la figura 9 se observa el diagrama de relación entre las tablas *survey, survey_item y survey_item_option*, este indica que todas tablas tienen como llave primaria los *id* y también se observa que todas las tablas se relacionan mediante una llave foránea las cuales son respectivamente, para la tabla *survey_item_option* la columna *survey_item_id*, y para la tabla *survey_item* la columna *survey_id*.

Las respuestas de los usuarios a dichas preguntas son almacenadas en la tabla *user_answer* y está relacionada con la tabla *analyze_text_sentiment* debido a que en esta tabla se almacenan las respuestas después de haber pasado por el

API de Google Natural Language. Esta última tabla contiene el comentario (columna *text*) además de un score (columna *score*) que indica la polaridad del comentario.

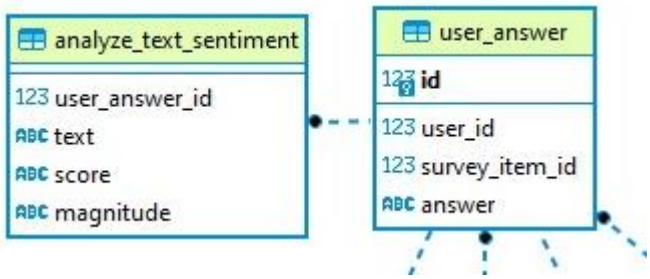

Figura 10. Diagrama relacional entre las tablas analyze_text_sentiment y user_answer.

En la figura 10 se encuentra el esquema relacional de las tablas *analyze_text_sentiment* y *user_answer*, estas se relacionan mediante la columna *user_answer_id*.

Los canales que se van a usar de la tabla *survey* son los canales en los que se interactúa con el cliente, los cuales son tiendas propias de la marca OFFCORSS (marca de C.I HERMECO S.A), outlets de la marca OFFCORSS y compras realizadas por la página web de la marca OFFCORSS.

De estos canales se tomaron solo aquellas preguntas abiertas en las cuales el usuario opinara de los productos de la marca OFFCORSS, de sus tiendas y de su página web.

Las preguntas seleccionadas por la empresa OCXCI S.A.S para realizar el análisis de sentimientos son aquellas preguntas de tipo Text Area, la cuales son preguntas de tipo abiertas y no son de respuesta obligatoria.

Las preguntas que se seleccionaron de la tabla *survey_item* fueron las siguientes:

- *"Déjanos un comentario."*
- *"¿Qué aspectos deberíamos mejorar?"*
- *"¿Por qué te gusto la prenda que compraste?"*
- *"Si pudieras mejorar algo de nuestros productos: ¿Qué seria?"*
- *"¿Qué tendría que pasar para que compres más seguido en OFFCORSS?"*

Estas preguntas, al ser de respuesta abierta, no tenían opciones y por ende no es necesario usar la relación entre las tablas *survey_item* y *survey_item_option*.

Para realizar la depuración de los datos se tomaron las respuestas de los usuarios entre el primero de mayo del 2022 y el 30 de julio del 2022.

En una primera instancia se tomaron todos los datos entre el primero de mayo del 2022 y el 30 de julio del 2022 usando SQL.

Para esta consulta realizó una unión entre las tablas anteriormente explicadas además que se realizó un etiquetado según los umbrales entregados por Google Natural Lenguage indicados en la figura 11.

Figura 11: Umbrales positivo, neutro y negativo del API de Natural Lenguage de Google

Dicha consulta dio como resultado un total de 4284 respuestas que se dividieron entre comentarios positivos, neutros y negativos.

Al realizar el depurado de los datos se encontró que algunas preguntas de tipo Text Area estaban generando ruido debido a que las preguntas no eran neutras y por el contrario ya tenían un sentimiento implícito. Estas preguntas fueron las siguientes:

- *"¿Qué aspectos deberíamos mejorar?"*
- *"¿Por qué te gusto la prenda que compraste?"*

Por ejemplo, a la pregunta "*¿Qué aspectos deberíamos mejorar?",* se obtuvieron las siguientes respuestas:

- *La atención al cliente.*
- *Variedad en mascotas.*
- *Empaques amigables con el medioambiente, menos uso de plástico.*

Todas las respuestas fueron etiquetadas con un score positivo, lo cual debería ser todo lo contrario debido al sentimiento negativo que implícitamente le está dando la pregunta a la respuesta.

Otra de las cosas que se encontró en esa primera consulta fue que se encontraron muchas respuestas positivas con score negativo y muchas respuestas negativas con score positivo, por esto se ajustaron los umbrales para los casos positivos mayores a 0.5, neutros entre 0.5 y -0.5 y negativos para los menores de -0.5.

Finalmente se realizó una segunda consulta teniendo en cuenta los cambios anteriormente dichos y se obtuvieron 3069 respuestas.

Para realizar dicha consulta en lenguaje de consulta estructurado primero se realizó un WHEN CASE con el que se realizaría el etiquetado de los datos entre los valores Negativo, Neutro y Positivo. Luego se realizó una unión de las tablas *analyze_text_sentiment*, *survey*, *survey_item*, *user_answer* y *user*, esta última contiene la información personal del cliente, con el fin de realizar la búsqueda solo en los canales anteriormente dichos y solo teniendo en cuenta las preguntas anteriormente presentadas, el código en SQL es el siguiente:

```
SELECT si.item_label as pregunta, ua.answer as respuesta,
CASE
    WHEN CAST(ats.score AS float) <= - 0.5 THEN 'Negativo'
    WHEN CAST(ats.score AS float) > - 0.5 and  CAST(score AS float) < 0.5 THEN 'Neutro'
    WHEN CAST(ats.score AS float) > 0.5 THEN 'Positivo'
ELSE ats.score END AS sentimiento
FROM analyze_text_sentiment ats
INNER JOIN user_answer ua
ON ats.user_answer_id = ua.id
INNER JOIN survey_item si
ON ua.survey_item_id = si.id
INNER JOIN survey s
ON si.survey_id = s.id
INNER JOIN public.user u
ON ua.user_id = u.id
WHERE s.id in ('53', '54', '50') AND CAST(u.completed_at AS DATE) <= '2022-07-30' AND
CAST(u.completed_at AS DATE) >= '2022-05-01' AND si.id IN ('384', '338', '736', '427')
```

Figura 12: Consulta empleada para extraer las preguntas de la base de datos.

En la figura 12 se puede observar la consulta empleada para extraer los datos de la base de datos, en esta se observa que se realiza un etiquetado con los score que tiene cada palabra, además se realizaron las búsquedas según los esquemas relacionales explicados anteriormente. Por último, solo se consultaron los canales previamente indicados y solo se tomaron las preguntas que no tuvieran un sentimiento implícito como se explicó anteriormente.

Las respuestas de dicha consulta fueron depuradas de la siguiente manera:

Se eliminaron las respuestas que estaban repetidas y cuya longitud fuera muy pequeña, esto para que la base de datos tuviese una gama alta de respuestas positivas, neutras y negativas.

Se cambiaron muchas respuestas que seguían presentando un mal etiquetado de los datos, por ejemplo, una de las respuestas que se repitió con mayor frecuencia fue *"ninguno"* respondiendo a la pregunta *"Déjanos un comentario."*. Dicha respuesta tiene una etiqueta negativa y se cambió por neutra debido a que esta respuesta no indicaba ningún aspecto positivo o negativo hacia la marca.

Por último, se tomaron algunas respuestas que contenían palabras del lenguaje coloquial colombiano para ser etiquetadas por otros miembros de las empresas OCXCI S.A.S y C.I HERMECO S.A, esto con el fin de que el etiquetado de los datos no estuviese basado solo en el criterio de una sola persona. Las palabras del lenguaje coloquial colombiano más recurrente fueron "*chévere", "Chiquis"* y *"Chiquitos".*

La base de datos resultante del depurado tiene un tamaño de 1050 filas y 2 columnas organizadas de manera aleatoria en formato CSV y codificación UFT-8. En la primera fila se encuentra el comentario de los usuarios y en la segunda se encuentran las etiquetas Positivo, Negativo o Neutro.

Dicha base de datos se compone de 350 datos etiquetados de manera positiva, 350 datos etiquetados como neutros y los restantes 350 datos fueron etiquetados como negativos.

Esta base de datos resultante fue entregada a la empresa junto con las indicaciones para realizar otras tablas similares o para acrecentar la base de datos realizada. Además, se entregaron los casos presentados anteriormente en los que se evidencia una alta fuente de ruido en los análisis de datos debido a preguntas con sentimientos implícitos.

Entrenamiento del modelo:

BERT es un modelo pre entrenado de procesamiento de lenguaje natural que usa *redes transformer*. Su arquitectura es similar a la de una *red transformer*, figura 3, con la única diferencia de que tiene una capa final para afinar el modelo a una tarea específica.

Dos de los modelos similares a *BERT* que usan el idioma español es *mBERT* y *BETO*, estos modelos vienen pre entrenados con un corpus sin etiquetar bastante amplio en español. El modelo que se eligió fue *BETO* debido a las publicaciones que lo respaldan a diferencia del modelo *mBERT*.

Para la implementación se hizo uso de la herramienta Google Colab usando la GPU como dispositivo para realizar su ejecución y su implementación fue realizada en el lenguaje de programación Python.

Primero se instaló la librería *transformer*, esta librería provee miles de modelos pre entrenados que se pueden usar para texto, audio y visión.

De esta librería se usaron los métodos *BertModel* y *BertTokenizer* los cuales se usan respectivamente para cargar el modelo *BERT* que se va a usar en la implementación y para darle una representación numérica a cada palabra.

Para ejecutar el modelo se truncarán los datos por palabras, esto para no saturar la memoria RAM y se van a tomar 3 clases, esto indica que los datos van a tener 3 estados: Positivo, negativo y neutro.

Antes de entrenar el modelo completo se carga el modelo pre entrenado BETO usando *from_pretrained* de la librería *BertTokenizer* y el modelo pre entrenado *beto_sentiment_analysis* de la librería *transformer*. Luego se deben remplazar las palabras de la frase con los tokens especiales del modelo pre entrenado usando el método *tokenize* el cual hereda de *BertTokenizer* los tokens de dicho modelo pre entrenado. Por ejemplo:

Para el texto, "*Me gustó mucho la tienda!"* resultan los siguientes tokens:

Frase: Me gustó mucho la tienda!

Tokens: [‘Me’, ‘gustó’, ‘mucho’, ‘la’, ‘tienda’, ‘!’]

Tokens numéricos: [1369, 12223, 1789, 1030, 6517, 1127]

Luego dichas entradas se codifican para realizar la afinación del modelo. La codificación consiste en ingresar a la frase los tokens de inicio, separación y padding y se realiza con el método *encode_plus* que tiene como parámetro de entrada el número máximo de longitud, si el texto se va a truncar y si se va a agregar padding. Para nuestro ejemplo:

A los tokens anteriores se le agregan respectivamente los tokens de inicio, separación y padding:
[‘[CLS]’, ‘Me’, ‘gustó’, ‘mucho’, ‘la’, ‘tienda’, ‘!’, ‘[SEP]’, ‘[PAD]’, ‘[PAD]’]

Estos tokens se le asigna una representación numérica usando tensores:

[4, 1369, 12223, 1789, 1030, 6517, 1127, 5, 1, 1]

Y se realiza una máscara atencional, la cual indica cuales son los caracteres que se van a analizar

[1, 1, 1, 1, 1, 1, 1, 1, 0, 0]

Dicho esto, se comienza a preparar la data para ser entrenada junto con el modelo pre entrenado:

Para leer la data se hace uso de la librería pandas de Python y se realiza un mapeo de las entradas, esto con el fin de tener una representación numérica de los sentimientos. Se le asigno el 0 a los valores negativos, 0.5 para los valores neutros y 1 para los valores positivos.

Luego se prepara la base de datos resultante del depurado de datos y se comienza a codificar para ser entrenada junto al modelo, para esto se construye la siguiente clase:

```
class HermecoDataset(Dataset):
  #Constructor de la clase
  def __init__(self,reviews,labels,tokenizer,max_len):
    self.reviews = reviews
    self.labels = labels
    self.tokenizer = tokenizer
    self.max_len = max_len
  #Metodo para conocer la longitud del comentario
  def __len__(self):
      return len(self.reviews)

  #Metodo para realizar la representación por tokens del comentario
  def __getitem__(self, item):
    review = str(self.reviews[item])
    label = self.labels[item]
    encoding = tokenizer.encode_plus(
        review,
        max_length = self.max_len,
        truncation = True,
        add_special_tokens = True,
        return_token_type_ids = False,
        pad_to_max_length = True,
        return_attention_mask = True,
        return_tensors = 'pt'
        )
    return {
          'review': review,
          'input_ids': encoding['input_ids'].flatten(),
          'attention_mask': encoding['attention_mask'].flatten(),
          'label': torch.tensor(label, dtype=torch.long)
      }
```

Figura 13: Clase HermecoDataset.

En la figura 13 se encuentra el algoritmo usado para realizar la clase *HermecoDataset*, la cual se encarga de preparar el comentario de entrada en un texto óptimo para ser entrenado con el método pre entrenado. Esta clase tiene en el constructor los elementos *reviews* que indica el comentario que se va a analizar, *labels* que es el valor numérico que representa los sentimientos del comentario, *tokenizer* el cual son los tokens del modelo pre entrenado *BETO* para cambiar las palabras a valores numéricos y *max_len* que indica la máxima longitud del comentario y por ende el truncamiento.

Esta clase retorna el comentario, el comentario codificado, su mascara atencional y el sentimiento de forma numérica.

Luego se construye la clase del Modelo *BERT*:

```
class BERTSentimentClassifier(nn.Module):

  def __init__(self, n_classes):
    super(BERTSentimentClassifier, self).__init__()
    self.bert = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
    self.drop = nn.Dropout(p=0.3)
    self.linear = nn.Linear(self.bert.config.hidden_size, n_classes)

  def forward(self, input_ids, attention_mask):
    _, cls_output = self.bert(
        input_ids = input_ids,
        attention_mask = attention_mask
    )
    drop_output = self.drop(cls_output)
    output = self.linear(drop_output)
    return output
```

Figura 14: Clase BERTSentimentClassifier.

En la figura 14 se encuentra el algoritmo que describe la estructura del modelo BERT, esta estructura agrega la red neuronal final con la cual se afina el modelo BERT para realizar el clasificador de sentimientos. Esta clase tiene como parámetro de entrada un modelo basado en redes neuronales y contiene en el constructor el modelo pre entrenado, el valor *drop* que hereda de la librería *pytorch* que indica que las neuronas tienen una probabilidad del 30% de desactivarse para cada iteración y una capa lineal de redes neuronales para la afinación, esta tiene como parámetro las 3 salidas que se desean. También tiene el método forward al cual se le ingresan los tokens y la máscara atencional de los datos previamente depurados.

Por último, tiene como salida el sentimiento de forma numérica del comentario a analizar.

El siguiente paso es realizar el entrenamiento, en este entrenamiento se dividen los datos depurados para realizar el entrenamiento y realizar la evaluación.

Para el entrenamiento se tiene la función *train_model*:

```
def train_model(model, data_loader, loss_fn, optimizer, device, scheduler,
n_examples):
  model = model.train()
  losses = []
  correct_predictions = 0
  for batch in data_loader:
    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    labels = batch['label'].to(device)
    outputs = model(input_ids = input_ids, attention_mask = attention_mask)
    _, preds = torch.max(outputs, dim=1)
    loss = loss_fn(outputs, labels)
    correct_predictions += torch.sum(preds == labels)
    losses.append(loss.item())
    loss.backward()
    nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
    optimizer.step()
    scheduler.step()
    optimizer.zero_grad()
  return correct_predictions.double()/n_examples, np.mean(losses)
```

Figura 15: Función train_model.

La función de la figura 15 se encarga de recibir los datos que se destinaron para el entrenamiento, pasarlos por el modelo *BERT* y compararlos con los valores que se tomaron aleatoriamente para la evaluación.

Para la evaluación se tiene la función *eval_model*:

```
def eval_model(model, data_loader, loss_fn, device, n_examples):
  model = model.eval()
  losses = []
  correct_predictions = 0
  with torch.no_grad():
    for batch in data_loader:
      input_ids = batch['input_ids'].to(device)
      attention_mask = batch['attention_mask'].to(device)
      labels = batch['label'].to(device)
      outputs = model(input_ids = input_ids, attention_mask = attention_mask)
      _, preds = torch.max(outputs, dim=1)
      loss = loss_fn(outputs, labels)
      correct_predictions += torch.sum(preds == labels)
      losses.append(loss.item())
  return correct_predictions.double()/n_examples, np.mean(losses)
```

Figura 18: Función eval_model.

La función de la figura 18 se encarga de evaluar el modelo con los datos que se eligieron para evaluación y compararlos con los datos que se tienen para entrenar el modelo.

Las dos funciones explicadas anteriormente calculan el promedio de datos que fueron entrenados y evaluados de forma de correcta.

El entrenamiento primero se realizó con 750 comentarios tomados aleatoriamente de los datos depurados para realizar una primera implantación de prueba en la cual se realizaron 5 épocas tardando 1:10:00 minutos en el entrenamiento. Luego se realizó una implementación con el total de los datos y además realizando un total de 5 épocas. Esto tardó 1 hora con 47 minutos y registró una exactitud promedio final en el entrenamiento del 97% y en la evaluación del 87%.

Para realizar la prueba con datos de encuestas de usuarios reales se realiza la función *classifySentiment* para codificar los datos según el modelo *BETO* y se envían todos los parámetros a la GPU donde está implementado el modelo como lo indica la figura19.

```
def classifySentiment(review_text):
  encoding_review = tokenizer.encode_plus(
      review_text,
      max_length = MAX_LEN,
      truncation = True,
      add_special_tokens = True,
      return_token_type_ids = False,
      pad_to_max_length = True,
      return_attention_mask = True,
      return_tensors = 'pt'
      )

  input_ids = encoding_review['input_ids'].to(device)
  attention_mask = encoding_review['attention_mask'].to(device)
  output = model(input_ids, attention_mask)
  _, prediction = torch.max(output, dim=1)
  print("\n".join(wrap(review_text)))
  print(prediction)
  if prediction == 1:
    print('Sentimiento predicho: Positivo')
  else:
    print('Sentimiento predicho: Negativo')
```

Figura 19: Función del salida del modelo

Para realizar una primera prueba se tomaron dos comentarios:

El primero "*Es una marca lleva muchos años en el mercado, Muy buena calidad en sus prendas. Lastima la antencion todo mal*" el cual obtuvo una calificación negativa

Y el segundo "*Siempre he usado offcorss, sus prendas son muy chéveres*" el cual obtuvo una calificación positiva.

Métricas de evaluación:

Para evaluar la eficacia del modelo implementado se tomó una base de datos con una estructura similar a la base de datos con la que se afinó el modelo, esta base de datos tiene un total 160 comentarios reales de usuarios y están etiquetados como comentarios negativos y positivos.

Las métricas usadas para evaluar el rendimiento del modelo se han tomado de estudios en los cuales se ha llevado a cabo una comparación entre diferentes algoritmos de análisis de sentimientos [10].

Primero se deben encontrar las matrices de confusión, en las cuales el sentimiento positivo se toma como un caso de éxito y el sentimiento negativo como un caso de no éxito.

Dichas matrices se componen por:

El número de datos que son calificados como verdaderos positivos (TP), es decir, los casos que se prevé que pertenecen a la clase de éxito y realmente pertenecen a clase de éxito.

El número de casos verdaderos negativos (TN), los cuales pertenecen a la clase de datos de no éxito y verdaderamente pertenecen a la clase de no éxito.

El número de casos que son falsos positivos (FP), es decir, los casos en los que se prevé pertenecen a la clase de éxito pero que realmente pertenecen a la clase de no éxito.

Y, por último, el número de casos falsos negativos (FN), en la que pertenecen aquellos comentarios que pertenecen a la clase de no éxito pero que realmente pertenecen a la clase de éxito [11].

Tanto TP como TN son clasificaciones correctas, mientras que FN y FP son clasificaciones incorrectas. A partir de dicha matriz de confusión, el rendimiento del modelo se evalúa con las siguientes métricas: exactitud, precisión, recall y F1 Score [12].

- La exactitud o porcentaje de aciertos en el método es el número total de opiniones correctamente clasificadas como positivas o negativas y proporciona la exactitud predictiva global de los métodos a la hora de clasificar las reseñas en positivas y negativas. Dicha exactitud se calcula como:

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \quad (1)$$

  Para el modelo de *NPL* que se está evaluando, el resultado de exactitud es del 91.8%, lo cual indica que el modelo tiene una alta exactitud a la hora de clasificar un sentimiento como negativo o positivo.

- La precisión en un método se determina a partir de la proporción de reseñas correctamente clasificadas entre rodas las reseñas que se han predicho con un sentimiento positivo.

$$precision = \frac{TP}{TP + FP} \qquad (2)$$

  Para el modelo de NPL afinado con los datos de C.I HERMECO S.A la precisión fue del 87.5%, lo cual indica una buena precisión para los comentarios con sentimiento positivo.

- La recuperación (recall) o sensibilidad es la proporción de verdaderos positivos. Indica la proporción de comentarios reales con un sentimiento positivo que han sido clasificadas correctamente. La recuperación es una medida de la solidez del modelo para predecir correctamente las opiniones que tienen un sentimiento positivo.

$$recall = \frac{TP}{TP + FN} \qquad (3)$$

Para el modelo *NPL* que se está evaluando, el valor de recall es del 91.3%, lo cual indica que el modelo tiene solidez para predecir de forma correcta las opiniones con sentimiento positivo.

- El valor de F1 score proporciona un equilibrio entre las dos clases, además que indica si el método tiene una alta capacidad de predecir sentimientos positivos.

$$F1\ score = 2\frac{precision * recall}{precision + recall} \qquad (4)$$

Para el caso del modelo evaluado su porcentaje F1 score es del 89.4%, lo cual indica que tiene una alta capacidad para predecir sentimientos positivos y que hay un equilibrio entre las clases.

Otra de las métricas que se usa para medir la eficacia de un modelo de inteligencia artificial, o modelo predictor, es el valor del área bajo la curva (AUC), de la curva característica del receptor (ROC). Con esta métrica se puede evaluar la sensibilidad frente a especificidad para un sistema clasificador binario según se varía el umbral de discriminación. Como se explicó en las métricas anteriores, tenemos una matriz 2x2 de opciones que puede tener una muestra. La cual, como se puede observar en la figura 18, son falsos positivos, falsos negativos, verdaderos positivos y verdaderos negativos.

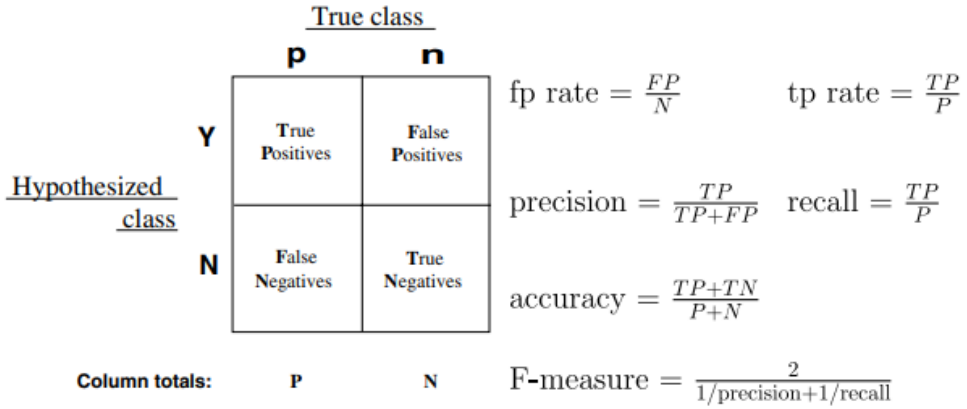

Figura 18: casos posibles en un modelo de predicción. [13]

Con dichos casos posibles se encuentran dos valores:

La razón de verdaderos positivos (VPR) con la cual se mide hasta qué punto un clasificador es capaz de clasificar casos positivos correctamente y la razón de falsos positivos (FPR) con la cual se miden cuantos resultados positivos son incorrectos de entre todos los casos negativos disponibles durante la prueba [13]. Estas razones se calculan como lo indica las ecuaciones 5 y 6.

$$vpr = \frac{Positivos\ clasificados\ correctamente}{Total\ positivos} \quad (5)$$

$$fpr = \frac{Negativos\ clasificados\ correctamente}{Total\ negativos} \quad (6)$$

Con dichos cálculos se grafica la curva característica del receptor:

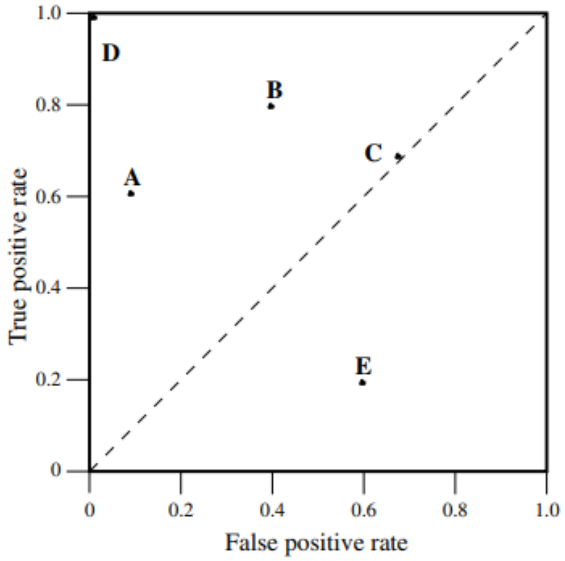

Figura 19: ROC.[13]

En la figura 19 vemos el espacio de la curva ROC. Para que un predictor o clasificador sea perfecto este debe pasar por el punto D como lo indica la figura 19, esto quiere decir que tiene un 100% de probabilidad de que la clasificación o la predicción sea correcta, si por el contrario el clasificador o predictor está en el punto C se dice que este tiene una probabilidad del 50% para clasificar o predecir un dato de forma correcta, ya que esta sobre la diagonal, y por último, si un método de clasificación o predicción está por debajo de la diagonal, como el punto E, se deben invertir sus predicciones o clasificaciones para aprovechar al máximo sus capacidades de predicción o clasificación [13].

La curva característica del modelo que se implementó es la siguiente:

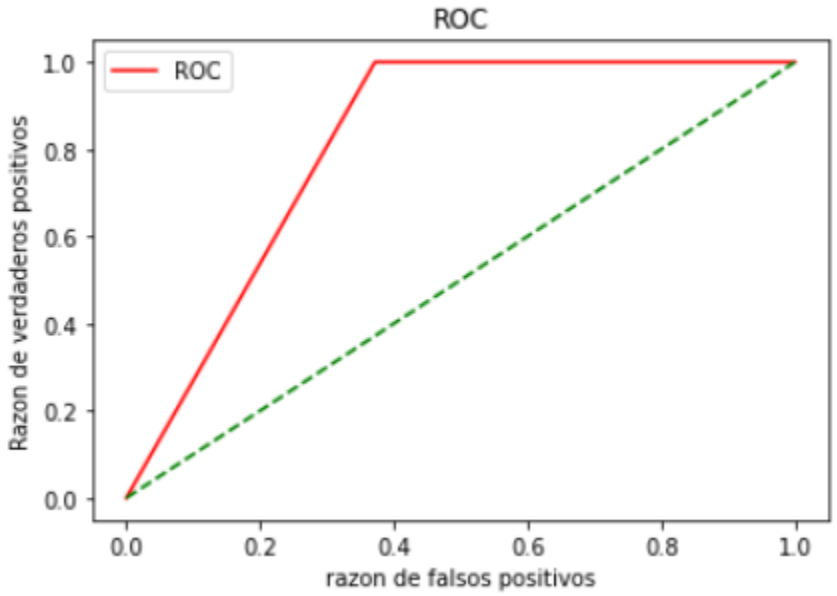

Figura 20: ROC del modelo afinado con los datos de C.I HERMECO S.A.

Como se puede observar en la figura 20 la curva ROC está por encima de la diagonal principal, indicando que el modelo tiene una probabilidad para clasificar un sentimiento mayor al 50%. El valor AUC es de 81.4% lo que indica que el modelo clasifica sentimientos con una probabilidad del 81.4% de que sean correctos.

Todos los cálculos anteriormente expuestos fueron realizados usando la librería de Python llamada *sklearn*, librería con la cual se evalúan modelos de inteligencia artificial.

Resultados y evaluación del modelo:

Los resultados obtenidos fueron entregados con el debido manejo de confidencialidad establecido en el contrato de practicas presentado por la empresa C.I HERMECO S.A y la empresa OCXCI S.A.S

Base de datos: se hizo entrega de dos archivos en formato CSV y codificación UFT-8 en los cuales se encuentran las opiniones, etiquetadas y depuradas, de usuarios reales, que aceptaron el *habeas data*, sobre la marca OFFCORSS para realizar tanto el entrenamiento del modelo *NPL* como para realizar las métricas de evaluación del modelo. Además, se entregaron los códigos en lenguaje *SQL* con los que se consultaron dichos comentarios de usuarios.

Excelente servicio, me encanta la ropa es de la mejor Calidad.	Positivo
Excelente servicio, me encantan los colores…	Positivo
Excelente servicio, muy eficientes y el med…	Positivo

Figura 21: comentarios de la base de datos para el entrenamiento

En la figura 21 se puede observar 3 de los 1050 comentarios que se utilizaron para el entrenamiento del modelo, además se puede ver que el archivo, en formato CSV, tiene dos columnas que se conforman por el comentario y la etiqueta que tiene dicho comentario. Este mismo formato se usó para la tabla con la que se evaluó el modelo.

Se informó a la empresa OCXCI S.A.S las preguntas que estaban generando ruido en la base de datos para que se realizaran de forma oportuna los cambios y no hubiera problema con el modelo que usa actualmente la empresa.

Algoritmo: Se hizo entrega de un archivo en lenguaje Python donde está descrito el algoritmo del modelo BERT, el algoritmo con el que se hace el entrenamiento del modelo BERT junto con su afinación y el algoritmo en el cual se transforman los datos en vectores numéricos atencionales para ingresarlos al modelo.

Además de los algoritmos descritos anteriormente, también se entregaron las funciones que se realizaron para realizar las gráficas de evaluación del modelo y sus resultados explicados de forma detallada.

Dichos algoritmos se encuentran explicados de forma detallada en un archivo Python que puede ser ejecutado mediante Google Colab y fueron compartidos a la empresa OCXCI S.A.S al servicio de C.I HERMECO S.A.

Modelo implementado: el modelo fue implementado haciendo uso de la herramienta de Google llamada Vertx AI, en la cual es posible implementar modelos de inteligencia artificial de forma fácil y rápida en Google Cloud, en esta herramienta se volvió a entrenar el modelo y se descargó el modelo ya entrenado. Luego, haciendo uso de la librería de Python Flask, se realizó un API con la que se consume el modelo de análisis de compra en tiempo real. Dicha API, con el modelo ya entrenado, puede ser implementada en las máquinas virtuales de Google Cloud para tenerlo en producción.

evaluación del modelo: Para realizar la evaluación del modelo se hizo uso de la librería *sklearn* y se calcularon los valores de exactitud, sensibilidad, f1score y AUC descritos en *métricas de evaluación*. Los resultados de dichas métricas fueron superiores al 80% indicando que el modelo es confiable, equilibrado y solido, por ende está listo para estar en producción.

Por último, se realizó la comparación del modelo *BERT* afinado con los datos de C.I HERMECO S.A y el modelo de Google Natural Lenguaje que usa OCXCI S.A.S actualmente para analizar opiniones de compra de la marca OFFCORSS. Dicha comparación se realizó en tiempo real haciendo uso de la API desarrollada para el modelo y de la API de Google con los comentarios de la marca OFFCORSS del día 25 de septiembre del 2022.

Tabla 2: Comparación en el etiquetado de comentarios de los modelos de Google Natural Language y BERT

Comentario	**etiqueta Google**	**Etiqueta BERT**
Me gusta toda la ropa en general	Positivo	Positivo
Tuve un error al pedir la talla y no súpe si podía hacer el cambio	Neutro	Negativo
Excelente calidad y servicio	Positivo	Positivo
Excelentes productos para los niños	Positivo	Positivo
Surtir un poco mas ropa de bebé 0 meses.	Neutro	Negativo
Muy buena	Positivo	Positivo
Excelente muchas gracias	Positivo	Positivo
Gracias por la atención	Positivo	Positivo
Más abilidad mmy atención	Neutro	Negativo
Muy buenas tiendas	Positivo	Positivo
Gracias por la calidad y la preparación de sus empleados, asesoran muy bien, me sentí muy apoyado sobre todo al ser papá primeriso.	Positivo	Positivo

Excelente la atención, sigan así Dios me los Bendiga	Positivo	Positivo
Felicitaciones y mi mayor deseo es que continúen con la innovación y calidad	Positivo	Positivo
Excelente ubicación y variedad	Positivo	Positivo

Como se puede observar en la tabla 2, el modelo *BERT* afinado con los comentarios de marca OFFCORSS tiene resultados similares en el etiquetado de datos frente al modelo que actualmente usa la empresa OCXCI S.A.S.

TRABAJOS FUTUROS:

En esta sección se presentarán los trabajos que se pueden realizar a futuro con las entregas presentadas en la sección anterior y se presentaran por dos líneas diferentes, la de los datos y la del algoritmo del modelo pre entrenado BERT.

En primer lugar, se entregaron dos bases de datos con 1050 entradas para entrenamiento y 160 entradas para pruebas, estas bases de datos pueden ir creciendo con el tiempo haciendo que el método de *NPL* sea cada vez más grande y robusto debido a que el desempeño de los datos depende de la cantidad de datos para el entrenamiento [3].

Además, para la empresa C.I HERMECO S.A, se pueden crear bases de datos para cada uno de los canales en específico (Venta directa, venta outlets y venta página web) para así tener un modelo afinado para cada canal de ventas y tener mayor veracidad en la clasificación del sentimiento.

En segundo lugar, para la empresa OCXCI S.A.S, debido a que se entregó el algoritmo del modelo y el algoritmo para realizar el entrenamiento, se pueden depurar los datos de otros clientes y crear modelos para los diferentes mercados en los cuales la empresa OCXCI S.A.S desea conocer las opiniones de los clientes.

CONCLUSIONES:

- Las bases de datos relacionales administradas por PostgreSQL y creadas con SQL permiten realizar de forma automática la inserción y consulta de datos. Esto hace que la depuración y etiquetado de los datos se pueda realizar de una forma más fácil y rápida mediante el uso de consultas eficientes en SQL. Usando filtros por palabras y cantidad de palabras se realizó una depuración más eficiente y además se lograron tener opiniones diferentes de todos los canales de ventas de la marca OFFCORSS haciendo que el corpus de datos usado para el entrenamiento albergara gran cantidad del espectro de opiniones que tienen los usuarios sobre la marca.
- La cantidad y calidad de los datos usados para el entrenamiento de un modelo de *machine learning* afectan directamente a la predicción que se espera del mismo. Para entrenar un modelo se debe tener una gran cantidad de datos balanceados por número de entradas para obtener una salida acorde a las entradas con las que el modelo se entrenó. Además, se debe tener, en los datos etiquetados de enteramiento y pruebas, todos los posibles tipos de entradas que va a tener el sistema, haciendo, de esta manera, que el modelo entrenado obtenga una salida positiva para cada entrada que obtendrá el sistema.
- Los modelos predictivos pre entrenados de procesamiento de lenguaje natural, como *BERT*, presentan una precisión y eficiencia alta, igualmente son altamente avalados y usados por la comunidad. Estos modelos, al usar *redes transformer* y tener una arquitectura similar, son altamente eficientes y prácticos a la hora de implementarlos. Dichos modelos están pre entrenados en más de 100 idiomas y constantemente se pre entrenan modelos *BERT* para muchos más idiomas, además que constantemente los modelos por idiomas, como *BETO*, se siguen pre entrenando con corpus de datos cada vez más grandes. Sin embargo, al afinar el modelo para realizar análisis de sentimientos, no se puede ignorar que su resultado puede ser afectado por la forma en que están escritos los comentarios, la ortografía y el sarcasmo, añadiendo de esta manera una capa más de dificultad. Así mismo, la forma en la cual se le solicita la opinión al usuario infiere considerablemente en el sentimiento de dicha opinión, porque, para que las respuestas del usuario puedan ser analizadas usando modelos de *NPL*, las preguntas deben ser neutras, es decir, que las preguntas no contengan ningún sentimiento implícito. Aun así, los resultados obtenidos con dichos modelos son principalmente positivos.
- Vertx AI es una herramienta prestada por Google que está integrada al paquete de Google Cloud. Esta herramienta tiene la opción tanto para afinar modelos pre entrenados de Google con set de datos propios como la opción para realizar el entrenamiento de modelos propios, opción que se usó en este proyecto. Con dicha herramienta se realizó un entrenamiento mucho más rápido, haciendo uso de los servidores de Google, y adicionalmente se descargó el modelo ya entrenado para ser almacenado en Google Cloud y consumido por una API desarrollada con la librería FLASK de Python.
- Las métricas de exactitud, sensibilidad, f1score y AUC sirven para medir la eficiencia y fiabilidad del modelo pre entrenado de *NPL* afinado con los datos de C.I HERMECO S.A. Con estas métricas se obtuvieron resultados positivos haciendo uso de un set de datos de comentarios reales. Así mismo, haciendo uso de la API desarrollada para usar el modelo, se comparó el análisis de sentimientos realizado usando el modelo BERT con el análisis de sentimientos haciendo uso del modelo de *NPL* desarrollado por Google, en ambos

casos usando comentarios de usuarios reales en tiempo real, obteniendo resultados positivos. Validando así la fiabilidad que tiene el modelo BERT.

REFERENCIAS BIBLIOGRÁFICAS

[1]. Rouhiainen, L. (2018). Inteligencia artificial. Madrid: Alienta Editorial.

[2]. Lee, J. D. M. C. K., & Toutanova, K. (2018). Pre-training of deep bidirectional transformers for language understanding.

[3]. Collarte González, I. (2021). Procesamiento del lenguaje natural con BERT: Análisis de sentimientos en tuits.

[4]. Hernández, M. B., & Gómez, J. M. (2013). Aplicaciones de procesamiento de lenguaje natural. Revista Politécnica.

[5]. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems.

[6]. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems.

[7]. Canete, J., Chaperon, G., Fuentes, R., Ho, J. H., Kang, H., & Pérez, J. (2020). Spanish pre-trained bert model and evaluation data.

[8]. Wu, S., & Dredze, M. (2019). Beto, bentz, becas: The surprising cross-lingual effectiveness of BERT.

[9]. Quiñones, A. (2021). Análisis de la sintaxis aprendida por BETO, un modelo de lenguaje en español basado en transformers.

[10]. Das, S. R., & Chen, M. Y. (2007). Yahoo! for Amazon: Sentiment extraction from small talk on the web. Management science, 53(9), 1375-1388.

[11]. Shmueli, G., Bruce, P. C., Yahav, I., Patel, N. R., & Lichtendahl Jr, K. C. (2017). Data mining for business analytics: concepts, techniques, and applications in R. John Wiley & Sons.

[12]. Alaparthi, S., & Mishra, M. (2021). BERT: A sentiment analysis odyssey. Journal of Marketing Analytics, 9(2), 118-126.

[13]. Fawcett, T. (2006). An introduction to ROC analysis. Pattern recognition letters, 27(8), 861-874.