PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

APLICACIÓN DE PROCESAMIENTO DE LENGUAJE NATURAL SOBRE UNA ENCUESTA DE SATISFACCIÓN

CARLOS ÁLVAREZ

Tesis para optar al grado de
Master of Science in Engineering

Profesor Supervisor:
PEDRO GAZMURI

Santiago de Chile, Julio 2021

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

APLICACIÓN DE PROCESAMIENTO DE LENGUAJE NATURAL SOBRE UNA ENCUESTA DE SATISFACCIÓN

CARLOS ÁLVAREZ

Miembros del Comité:

PEDRO GAZMURI

DENIS PARRA

ANDRÉS ARTIGAS

ANDRÉS GUESALAGA

Tesis para optar al grado de
Master of Science in Engineering

Santiago de Chile, Julio 2021

Gracias a mis padres por su apoyo en mi educación.

AGRADECIMIENTOS

Doy gracias a mis padres Carlos y Verónica por su apoyo y cariño entregados durante mi educación escolar y universitaria.

En segundo agradezco a la empresa, en especial a Matías y Freddy por facilitar la realización del trabajo y su compromiso con este. También a Andrés por su voluntad de ayudar a revisar los resultados y redacción de la tesis.

Finalmente, agradezco al profesor Pedro Gazmuri por guiarme en el enfoque, redacción y exposición del trabajo realizado.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABSTRACT

Natural Language Processing techniques were used on a customer satisfaction survey of the retail industry in order to obtain relevant information from the answered text that cannot be obtained from traditional satisfaction level indicators. The work focused on the analysis of different models of multi-label text classification and extractive text summarization.

A text classification baseline was built using FastText to model the language. It exposed how different factors in text preprocessing affect the classification performance. The results indicate that the correction of spelling mistakes improves the classification of the texts and data augmentation increases the precision, but decreases the recall. Implementing the general-purpose BERT model substantially improves text classification performance over the baseline. Spell checking is shown to be relevant in classification performance and short texts are classified better.

The extractive text summarization experiments succeed in summarizing relevant facts better than methods proposed in other works and better than summarizing at random. Incorporating text classification information improves the quality of the summary.

The results of the classification of the surveys are compared with the traditional customer satisfaction indices. It is found that customers comment on more negative aspects than positive and that these are not reflected in the satisfaction index.

Keywords: **FastText, BERT, extractive text summarization**.

RESUMEN

Se emplearon técnicas de Procesamiento de Lenguaje Natural sobre una encuesta de satisfacción de la industria del retail con el objetivo de obtener información relevante del texto contestado que no se puede obtener a partir de los indicadores de nivel de satisfacción tradicionales. El trabajo se enfocó en el análisis de diferentes modelos de clasificación *multi-label* de texto y de resumen extractivo de texto.

Se construyó un *baseline* de clasificación de texto en que se utilizó FastText para modelar el lenguaje. Se expone un análisis de cómo distintos factores del preprocesamiento de texto afectan en el rendimiento de clasificación. Los resultados indican que la corrección de faltas de ortografía mejora la clasificación de los textos y el aumento artificial de datos aumenta la precisión, pero disminuye la exhaustividad. La implementación del modelo de propósito general BERT mejora sustancialmente el rendimiento de clasificación de textos respecto del *baseline*. Se muestra que la corrección ortográfica es relevante en el rendimiento de clasificación y que los textos cortos se clasifican de mejor manera.

Los experimentos de resumen extractivo de texto logran resumir los hechos relevantes mejor que métodos propuestos en otros trabajos y mejor que seleccionando al azar. La incorporación de información de clasificación de texto mejora la calidad del resumen extraído.

Se comparan los resultados de la clasificación de las encuestas con los índices de satisfacción tradicionales. Se encuentra que los clientes comentan sobre más aspectos negativos que positivos y que estos no se ven reflejados en el índice de satisfacción.

Palabras Claves: **FastText, BERT, resumen extractivo de texto**.

1. INTRODUCCIÓN

1.1. Motivación

El Procesamiento de Lenguaje Natural (PLN) es un conjunto de técnicas computacionales, motivadas por la teoría, para el análisis automático y representación del lenguaje humano. *General Language Understanding Evaluation benchmark* (GLUE) Wang et al. (2018) es una colección de herramientas para medir el rendimiento de modelos de representación del lenguaje a lo largo de varias tareas del Entendimiento del Lenguaje Natural (ELN), una subárea de investigación de PLN que mediante el concepto de Inteligencia Artificial busca generar comprensión lectora en máquinas. Entre estas tareas se incluyen análisis de sentimiento, comprensión lectora a través de preguntas y respuestas, vinculación de textos y clasificación de textos. GLUE mantiene una tabla de clasificación de modelos de ELN, donde se muestra el *ranking* de estos de acuerdo a su rendimiento en las tareas que mide esta herramienta. Durante el último lustro, el desarrollo de esta área de investigación ha crecido vertiginosamente, y modelos propuestos por distintos autores han superado constantemente los puntajes del *benchmark* GLUE. Solo desde abril de 2018 el estado del arte de PLN, medido como el puntaje del *benchmark* GLUE pasó de 70,0 puntos a 83,9 en mayo de 2019, acercándose al puntaje humano obtenido en el test de 87,1 Nangia & Bowman (2019), en una escala de 100,0.

En esta sección se expondrá una visión general de los cambios de paradigma respecto de la modelación matemática del lenguaje en el campo de investigación de PNL, partiendo por exponer los métodos estadísticos que se desarrollaron en sus inicios, hasta mostrar a grandes rasgos lo que implica el concepto de representación contextualizada de palabras que ha logrado elevar la comprensión lectora en máquinas. De esta manera se quiere explicar la motivación de este trabajo de tesis, que es implementar los recientes avances del campo de investigación de PNL para resolver un problema real que se presenta en la industria del retail.

En los años 2.000, era usual que el trabajo computacional se realizara solo con el conjunto de textos con el que se trabaja, con el objetivo de identificar palabras que se repiten frecuentemente, o tópicos importantes presentes, que pueden ser de utilidad para algún análisis de lo que está plasmado en dichos textos. Esto se realiza utilizando métodos estadísticos como TF-IDF Ramos (1999) o *Latent Dirichlet Allocation* Blei et al. (2003). El análisis de texto utilizando estos modelos se basa en identificar características de este, midiendo la frecuencia de aparición de términos o modelando una distribución que permita identificar los tópicos más importantes.

Posteriormente surgió el concepto de *transfer learning*, que se refiere a extraer información de un conjunto de textos muy grande, por ejemplo la Wikipedia o un conjunto grande de libros, de tal manera de crear una modelación matemática del lenguaje que sea capaz de capturar el significado de las palabras en un espacio vectorial, y luego utilizar este conocimiento extraído sobre un set de datos acotado y específico en algún trabajo particular. Este tipo de modelos se basan en redes neuronales, que son capaces de codificar información de manera abstracta en sus parámetros, y de esta manera almacenan el significado semántico de las palabras del lenguaje. Dado que para realizar extracción de información de un conjunto de textos muy grande se requiere gran poder computacional y tiempo, es común que organizaciones que se dedican a la investigación del PNL entrenen este tipo de modelos, y luego los publiquen para que puedan ser usados por la comunidad en alguna tarea acotada y específica.

El primer modelo con relativo éxito que se construyó basado en redes neuronales es presentado en el trabajo *"A Neural Probabilistic Language Model"* Bengio et al. (2003), que según los autores logra evitar el problema de "maldición de la dimensionalidad". Resumidamente, esto se refiere a que se logra modelar el lenguaje a través de variables aleatorias continuas en un espacio vectorial de dimensión mucho menor en comparación con previos modelos basados en métodos estadísticos, que utilizan variables aleatorias discretas. Por ejemplo, si en un modelo estadístico, en que se trabaja utilizando una distribución conjunta de 10 palabras consecutivas que

aparecen en las oraciones del texto, y la cantidad de palabras distintas en los textos con que se trabaja es de 100.000, existen potencialmente $100.000^{10} - 1 = 10^{50} - 1$ parámetros libres. Por otro lado Bengio et al. (2003) logra modelar el lenguaje utilizando un modelo basado en redes neuronales en que la cantidad de parámetros escala linealmente con el número de palabras distintas del texto, que usualmente es del orden de 10^5 a 10^6. El mayor valor que introdujo este trabajo al desarrollo del campo de investigación de PLN fue proponer una nueva aproximación al modelamiento del lenguaje utilizando vectores de características para cada palabra.

El mayor problema con el modelo propuesto por Bengio et al. (2003) consiste en que el entrenamiento del modelo involucra múltiples multiplicaciones de matrices, y por lo tanto es costoso en términos de tiempo y memoria. Cerca de una década después, Mikolov, Sutskever, et al. (2013) introdujo un modelo de lenguaje distribuido llamado Skip-Gram cuyo tiempo de entrenamiento es lineal, que alcanzó mejor rendimiento que los que se venían haciendo, e hizo que la investigación investigación del PLN crezca exponencialmente. Un modelo de lenguaje distribuido se refiere a un modelo que genera vectores de palabras. Estos vectores almacenan características semánticas y sintácticas del lenguaje en sus componentes. La semántica alude al significado de las expresiones lingüísticas, y la sintaxis tiene que ver con el orden y la relación de las palabras en una oración, por ejemplo la relación singular-plural. En Mikolov, Yih, & Zweig (2013), un trabajo que expande y analiza los alcances logrados en Mikolov, Sutskever, et al. (2013), los autores muestran que a través de desplazamientos vectoriales (suma o traslación en el espacio), los vectores de palabras generados por el modelo son capaces de capturar las propiedades del lenguaje mencionadas.

La forma de trabajar en alguna tarea de PLN con un modelo de lenguaje distribuido consiste en modelar una oración utilizando los vectores de las palabras que la componen. Si se requiere clasificar una determinada oración de acuerdo a un criterio de un problema planteado, el procedimiento consistirá en primero generar un vector de representación de la oración, que puede ser calculado como el promedio de

los vectores de palabras que la componen, y luego introducir el vector de la oración a un modelo clasificador. A lo largo de este trabajo de tesis se explicará este procedimiento con mayor detalle.

Aunque los modelos de lenguaje distribuido fueron un gran salto en la caracterización matemática del lenguaje, estos presentan algunas limitaciones en la representatividad de vectores de palabras, por ejemplo que a cada palabra se le asocia un único vector, siendo que en el lenguaje las palabras pueden tener múltiples significados. Este problema fue abordado a través de un nuevo paradigma de representación del lenguaje, que en lugar de generar vectores de palabras, consiste en generar vectores contextualizados de palabras.

En los últimos años, se desarrollaron los modelos de lenguaje de propósito general, que a diferencia de los modelos de lenguaje distribuido, son capaces de generar vectores de palabras de acuerdo al contexto en que estas se presentan, es decir, en lugar de tener como objetivo capturar el significado de una palabra en un vector, su objetivo es capturar el significado de una palabra en un vector considerando el contexto de todas las palabras que la rodean en una determinada oración o texto. De esta manera una palabra tiene más de una representación posible y esta depende del contexto lingüístico en que aparece.

Los modelos de propósito general normalmente tienen una cantidad de parámetros que va desde del orden de 10^8 y su surgimiento en los últimos años solo ha sido posible debido al crecimiento del poder de cómputo que ha logrado generar la sociedad en su conjunto, haciendo más accesible económicamente la capacidad computacional necesaria para procesarlos. El desarrollo de este tipo de modelos ha sido liderado por grandes organizaciones, por ejemplo Google, que utilizando el concepto de *transfer learning* son capaces de entrenar modelos cuyo entrenamiento es extremadamente costoso y publicarlos para ser utilizados por investigadores que deseen utilizarlos en tareas específicas.

En vista de los avances realizados en el ELN se ha decidido trabajar experimentando con el reciente modelo de propósito general *Bidirectional Encoder Representations from Transformers* (BERT) Devlin et al. (2018), que a la fecha de definir el trabajo que se realizará en esta tesis, había logrado un gran salto en el ELN, medido a través del *benchmark* GLUE, elevando el puntaje en una diferencia absoluta de 7,7 puntos porcentuales respecto de su predecesor.

En este trabajo se quiere acercar el conocimiento desarrollado por la investigación académica a la industria y con la colaboración de una empresa chilena de retail se emplearán los recientes avances en el PLN, comparando y analizando diferentes técnicas en este campo de investigación para solucionar un problema real que se les presenta.

1.2. Aplicación de Clasificación y Resumen de Textos

La empresa realiza una encuesta de satisfacción a sus clientes para medir la lealtad y la satisfacción de estos respecto a sus distintas áreas de negocio. La áreas o ejes de negocio, son aspectos claves en la operación cotidiana de la empresa que tienen que ver con aspectos del servicio que entregan al cliente, como por ejemplo la atención del personal, el orden de la tienda o la calidad del servicio de despacho, entre otros. La lealtad de los clientes se mide con el indicador *Net Promoter Score* (NPS) y la satisfacción respecto a cada eje de negocio se mide a través del *Customer Satisfaction Score* (CSAT). Ambos puntajes se calculan numéricamente a través de las respuestas de los clientes a preguntas de alternativas en una escala de 1 a 10 y de 1 a 5 respectivamente. Además, en esta encuesta la empresa realiza una pregunta de texto abierto para tener el motivo de la evaluación de los clientes respecto de su experiencia de compra. Debido al volumen de encuestas semanales que reciben, es muy costoso resumir toda la información contenida en los textos y generar información relevante y consistente a partir de ella.

En el campo de texto abierto el cliente especifica los aspectos más relevantes para su experiencia y también aporta información adicional que no puede ser recopilada a través de preguntas de alternativas. Por medio de las respuestas escritas la empresa puede comprender la razón del porqué un cliente evaluó con cierta nota su experiencia de compra y así detectar irregularidades e ineficiencias dentro de las operaciones diarias o identificar aspectos específicos que el cliente valora. Actualmente la empresa no es capaz de rescatar toda la información adquirida a través del campo de texto abierto y el personal solo es capaz de resumir algunas respuestas manualmente.

En la información que está contenida en los textos de la encuesta de satisfacción, la empresa busca específicamente 1) encontrar el motivo del porqué un eje del negocio tiene buen o mal rendimiento y 2) entender cómo el nivel de satisfacción de los clientes afecta su proceso de recompra o fidelización. Para buscar una solución a esta necesidad, este trabajo de tesis se concentrará en realizar tareas de clasificación y resumen sobre los textos de la encuesta. Se experimentará implementando diferentes de modelos de clasificación de texto y resumen de texto basado en los recientes avances en el campo de investigación de PLN, realizando un análisis de la relevancia que pueden tener diferentes técnicas de preprocesamiento de texto en el proceso, con la finalidad proponer una herramienta computacional basada en la teoría y la práctica del PLN que resuelva la necesidad expuesta.

1.2.1. Problema y Contexto

La experiencia de compra del cliente es un aspecto clave para generar fidelización de este a la empresa de retail y así establecer una relación a largo plazo Cachero-Martínez & Vázquez (2017). El proceso de compra puede fomentar la creación de un vínculo especial con el consumidor, generando clientes más leales y de esta forma elevar la intención de recompra y disposición a pagar. Existen variadas formas de medir la lealtad o la satisfacción de un cliente, entre las más populares están los indicadores NPS (*Net Promoter Score*), CSAT (*Customer Satisfaction Score*) y CES (*Customer Effort Score*) Dessel (2014), de los cuales la empresa utiliza los

primeros dos. Estos están enfocados en medir el nivel de fidelidad del cliente a la empresa en el largo plazo, el nivel de satisfacción del cliente respecto a cierta área de la empresa y cuán fácil es para el cliente comprar en la tienda, respectivamente.

De acuerdo a Gentile et al. (2007), la experiencia de compra de un cliente en el retail consiste en la percepción de este con respecto a un conjunto de interacciones entre él y la empresa o servicio. Para la empresa es importante medir qué aspectos del negocio tienen buen rendimiento en cuanto a la percepción de los clientes y cuáles no. En consecuencia, realiza una encuesta de satisfacción dentro de las 48 horas posteriores a una compra en un local y esta consiste en preguntar mediante preguntas de alternativas al cliente cómo evalúa los distintos aspectos o ejes del negocio en el proceso de su compra. Con ello, la empresa es capaz de generar los indicadores NPS para medir la lealtad del cliente con la tienda a largo plazo y CSAT para medir el rendimiento general del negocio en cuanto al nivel de satisfacción de los clientes y el rendimiento de cada unos de los 13 ejes de negocio definidos por la empresa. Los 13 ejes y su respectiva definición puede verse en la tabla 1.1.

La obtención de información para el cálculo del puntaje NPS se realiza preguntando a los clientes "¿Con que probabilidad recomendarías la Tienda a tus familiares y amigos?" dando la opción de contestar en una escala de 1 a 10, donde 1 es "muy improbable" y 10 es "lo recomendaría con seguridad". Se categoriza a los clientes que contestaron 9 o 10 como promotores y a los que contestaron 6 o menos como detractores. Luego en la encuesta se le pide al cliente contestar con una alternativa de 1 a 5 qué tan de acuerdo está con una afirmación que indica que estuvo satisfecho con un aspecto del negocio en su compra, donde 1 es "nada satisfecho" y 5 es "muy satisfecho" y esto se pregunta para cada aspecto del negocio. El puntaje final del indicador NPS varía entre -1 y 1 se calcula como el porcentaje de promotores menos el porcentaje de detractores y existe un puntaje CSAT para cada eje de negocio y su valor corresponde al promedio de los puntajes dados por los clientes para un determinado eje.

Tabla 1.1. Ejes de Negocio.

Eje de Negocio	Definición
Al Pagar	Todos los aspectos que se consideran relevantes en el proceso de pago. Entre ellos el tiempo de espera en caja, la actitud y calidad de atención de la cajera, el tiempo de espera para atención.
Atención	Todos los aspectos relevantes en la atención por parte de los vendedores en tienda hacia los clientes. Ejemplos son la actitud de los vendedores, la amabilidad, etc.
Call Center	Todos los aspectos relacionados con la experiencia obtenida a través de un contacto con el Call Center de la empresa.
Despacho	Todos los aspectos relacionados con el proceso de despacho a domicilio de artículos comprados por internet o en un local. Por ejemplo la puntualidad del despacho, el estado en que llega el producto al domicilio, que efectivamente llegue el producto que se compró y no otro, etc.
Facilidad	Todos los aspectos relacionados con el proceso de compra integral. Responde a cuán fácil es para el cliente comprar desde que entra a la tienda hasta que sale de ella.
Precio	Todos los aspectos relacionados con los precios de los productos. Entre ellos la percepción de precios altos o bajos, ofertas, claridad en la señalización del precio, etc.
Producto	Todos los aspectos relacionados con los productos que vende la empresa. Estos son la variedad, el stock, la calidad de los productos, entre otros.
Puntos Cencosud	Todos los aspectos relacionados con la promociones y beneficios para los clientes fidelizados a través del programa de puntos de la empresa de retail.
Retiro	Todos los aspectos relacionados con el proceso retiro en tienda de compras efectuadas por internet. Algunos ejemplos son la calidad de atención de la sección de retiro, el tiempo de espera para el retiro, entre otros.
Servicio al Cliente (SAC)	Todos los aspectos relacionados con la experiencia obtenida en el área el Servicio al Cliente en la tienda.
Sitio Web	Se refiere a todos los aspectos relacionados con la interacción del cliente y el sitio web. Por ejemplo la visibilidad de los productos y promociones, el proceso de pago a través del sitio web y otros.
Tarjeta Cencosud	Todos los aspectos relacionados con el proceso de pago y de beneficios por parte del servicio financiero de la empresa, que entrega una tarjeta bancaria a los clientes.
Tienda Física	Todos los aspectos relacionados con el local físico. Algunos son orden de la tienda, distribución de espacio, visibilidad de señalizaciones, etc.

Dentro de la encuesta de satisfacción se hace la pregunta "¿Cómo evalúas tu experiencia en la tienda?" para medir el nivel de satisfacción de la experiencia de compra en general, donde el cliente debe contestar con nota de 1 a 5, y se puede

considerar como un puntaje CSAT general de la experiencia de compra en su conjunto. A continuación se pregunta "¿Nos contarías por qué evalúas con esa nota?", donde el cliente debe escribir la razón en un campo de texto abierto. El objetivo es tener el puntaje de la evaluación general de la experiencia de compra y el motivo del puntaje, respectivamente. En la tabla 1.2 se muestran algunos ejemplos de las respuestas recibidas. Como se aprecia en la tabla, los comentarios son escritos en un contexto informal y contienen faltas de ortografía y gramática.

Tabla 1.2. Ejemplos de respuestas textuales a la pregunta "¿Nos contarías por qué evalúas con esa nota?" de la encuesta de satisfacción.

Respuestas
"la atención en la tienda por parte del vendedor sentí poco interés, con respuestas muy escuetas respecto del producto, ahora el sistema de despacho, la atención es pésima, para entregar el producto se demoraron 25 minutos desde el momento que al trabajador le entregaron la orden de despacho, súmele a eso el lugar es inhóspito en plena calle, el encargado actúa con una indolencia y desidia que hace sentir que esta haciendo una favor al cliente."
"Los productos con descuento no marcaban el descuento y tuve que esperar para que verificaran y despues llamaran a la jefa de departamento y aplicar el descuento"
"Me gusta la tienda ordenada, y sus trabajadores muy amables,,, pregunta que les hacía me respondían muy amablemente"
"Rapidez en caja, encontré lo que buscaba, excelente opción de envio de boletA al correo asi no se extravían"
"porque mi compra tuvo que ver con canje de puntos, y no tenían una guía en donde pudiera ver a qué optar con los puntos que tenía y ellos en lo personal no tenía claridad de los productos. Me mandaron a que lo consulte yo misma en mi celular.La atención del Señor de la caja estuvo muy buena."
"Tienda hordenada variedades de produ"
"X la atencion"
"la compra fue rápida. fuimos producto de un cambio y la información estuvo bien entregada desde el principio."

Para la empresa es de suma importancia comprender la razón del porqué un cliente evaluó con cierta nota su experiencia de compra, para así poder detectar irregularidades e ineficiencias dentro de las operaciones diarias o identificar aspectos específicos que el cliente valora. Actualmente la manera en que lo hacen es leer los comentarios uno a uno. Debido al gran volumen de encuestas recibidas y a que estas deben ser leídas y resumidas por el personal, la empresa no es capaz de rescatar toda

la información adquirida a través del campo de texto abierto. En este campo el cliente especifica los aspectos más relevantes para su experiencia y también aporta información adicional que no puede ser recopilada a través de los indicadores de lealtad y satisfacción descritos anteriormente.

1.2.2. Solución Computacional

Teniendo en cuenta las necesidades de la empresa, se plantea implementar una solución computacional que resuelva el problema de recolectar la información del campo de texto abierto que es contestado por los clientes a partir de la pregunta "¿Nos contarías por qué evalúas con esa nota?" de manera automatizada. La herramienta sólo tomará en cuenta la respuesta expresada por el cliente, sin considerar las notas que puso a cada eje de negocio, enfocando el análisis en el procesamiento, clasificación y resumen de textos.

El objetivo es explorar, implementar y analizar los recientes avances del campo de investigación de PLN para generar una solución computacional al problema que presenta la empresa, mediante la extracción de información de valor a partir de las respuestas de los clientes sobre su experiencia de compra. Tener este conocimiento podría ayudar a solucionar problemas operacionales con mayor eficacia y generar inteligencia de negocio, entre otras utilidades. Para realizar esto la empresa ha entregado una base de datos que contienen 166.111 respuestas a la encuesta. De aquí en adelante, cada respuesta de los clientes a la pregunta "¿Nos contarías por qué evalúas con esa nota?" se tratará como un "comentario", que se compone de todo el texto que escribió el cliente al responder a la pregunta.

La solución consiste en primero identificar de qué trata cada comentario de los clientes, clasificándolos de acuerdo a sobre qué eje de negocio se expresa el cliente y si es bueno o malo respecto al rendimiento esperado por la empresa. Este problema se puede caracterizar como uno de clasificación *multi-label* de textos. La clasificación *multi-label* de textos se refiere a asociar a cada texto a un subconjunto de todas las posibles etiquetas que se tienen y en este caso se requiere asignar una etiqueta de tipo

(<eje de negocio>, <polaridad>) a cada comentario. Por ejemplo si un comentario es "El vendedor tuvo mala actitud, pero la atención en caja fue rápida", será clasificado con las etiquetas "Atención-Negativo" y "Al Pagar-Positivo".

El proceso general de clasificación texto consiste en 1) preprocesar el texto, 2) vectorización del texto, 3) entrenamiento del modelo clasificador y 4) ajuste de los hiperparámetros del modelo (*fine-tuning*). En este trabajo primero se analizará cómo influyen diferentes técnicas de preprocesamiento de texto en la clasificación de estos utilizando el modelo de lenguaje distribuido FastText Bojanowski et al. (2016) para vectorizar el texto. Este modelo se caracteriza por rapidez de entrenamiento y robustez ante faltas de ortografía. Se establecerá un *baseline* de clasificación de texto, luego se implementará el reciente modelo de propósito general BERT para vectorizar y clasificar textos y se compararán los resultados obtenidos con el *baseline* establecido.

Segundo, se implementará una solución que permita resumir automáticamente un conjunto de comentarios, extrayendo la información más relevante de estos. Se experimentará agrupando comentarios emitidos en un determinado periodo de tiempo o local donde se realizó la compra y se procederá a resumirlos. El objetivo es que se logre entender el motivo de la evaluación obtenida por parte de los clientes a partir de toda la información que se tiene disponible. Este problema se aborda con técnicas de resumen extractivo de texto (*extractive text summarization*) y se experimentará implementando diferentes metodologías propuestas para cumplir este propósito.

Finalmente, se realizará un análisis cuantitativo y cualitativo de la información obtenida a partir de la implementación de la solución computacional propuesta. Este análisis se enfoca en mostrar la utilidad de la herramienta en cuanto a la información nueva que se puede extraer de los comentarios.

1.3. Hipótesis y Objetivos

La hipótesis planteada en este trabajo de tesis es que aplicando los recientes avances en el campo de investigación de PLN y utilizando la información escrita entregada por los clientes al responder la encuesta de satisfacción de la empresa se puede extraer información sobre el nivel de satisfacción y relevancia en la experiencia de compra de cada eje de negocio, que antes no se tenía, que permita tomar decisiones estratégicas y operacionales en el negocio.

Los objetivos específicos del trabajo de tesis son 1) analizar distintas técnicas de preprocesamiento texto y diferentes formas de abordar el problema de clasificación *multi-label* en conjunto con la implementación de un modelo de lenguaje distribuido para establecer un *baseline* de clasificación de texto, 2) emplear y analizar un modelo de lenguaje de propósito general y aprendizaje profundo para la clasificación *multi-label* de textos y compararlo con los resultados obtenidos en el *baseline* realizado, 3) implementar y analizar modelos de resumen extractivo de texto para generar resúmenes de los comentarios de los clientes y 4) mostrar la utilidad de la herramienta a partir de un análisis cuantitativo y cualitativo de la información generada.

1.4. Estructura del Documento

Este documento comienza describiendo la motivación del trabajo de tesis, planteando el problema a resolver, la hipótesis y objetivos de esta. En el Capítulo 2 se presenta una revisión bibliográfica del campo de investigación de PLN y el marco teórico con que se trabaja. En el Capítulo 3 se describe la metodología y experimentos de este trabajo de tesis. En el Capítulo 4 se presentan los resultados de los experimentos y análisis del procesamiento de texto. En el Capítulo 5 se realiza una comparación de las métricas del nivel de satisfacción utilizadas por la empresa con los resultados obtenidos a partir de la implementación del modelo clasificador

sobre la base de datos entregada, exponiendo las aplicaciones de la herramienta generada. Finalmente, en el Capítulo 6 se exponen las conclusiones y trabajo futuro.

2. REVISIÓN BIBLIOGRÁFICA

El este trabajo se implementan y analizan las tareas de PLN de clasificación *multi-label* de texto y de resumen extractivo de texto. Este capítulo tiene por objetivo realizar una revisión bibliográfica que exponga la base teórica y prácticas comunes que se utilizan en el campo de investigación de PLN para llevar a cabo estas tareas. Primero se muestra una breve introducción a la práctica y experimentación que se utilizará para resolver el problema de clasificación y resumen de texto, y a lo largo del capítulo se explicará en detalle la base teórica y práctica que involucra cada procedimiento.

Como se mencionó en el capítulo anterior, el proceso general de clasificación de texto se realiza siguiendo los siguientes pasos: preprocesamiento del texto, vectorización del texto, entrenamiento del modelo clasificador y ajuste de los hiperparámetros del modelo clasificador. El preprocesamiento tiene que ver con normalizar el texto con que se trabaja para lograr de identificar secuencias de caracteres (palabras o subpalabras) que luego se modelarán como vectores. La vectorización se refiere a la transformación de secuencias de caracteres a vectores, que permitirán entrenar un modelo clasificador capaz de asociar cada ejemplo de texto a un subconjunto de etiquetas determinado.

La calidad de un vector palabra se observa a través de un concepto abstracto, que tiene que ver con qué tan bien se han capturado los elementos intrínsecos del lenguaje en los componentes de ese vector comparado con todos los otros vectores de palabras que se han generado. Es un concepto abstracto ya que no se puede observar explícitamente qué componente de un vector de palabra representa un elemento del lenguaje específico, por ejemplo relación de género, significado semántico, elementos sintácticos u otros. Sin embargo, a través de trabajos como el de Mikolov, Sutskever, et al. (2013), mediante desplazamientos vectoriales de los vectores de palabras se puede observar que estos han logrado capturar elementos intrínsecos del lenguaje. Esto se expondrá con mayor detalle a lo largo de este capítulo. Es importante realizar

una vectorización de palabras de calidad ya que esta afecta el rendimiento de la clasificación de los textos, que es determinado por métricas de evaluación específicas.

En este trabajo se utilizarán dos tipos de modelos para lograr la vectorización del texto, el modelo de lenguaje distribuido FastText Bojanowski et al. (2016) y el modelo de lenguaje de propósito general BERT Devlin et al. (2018). El primero se destaca por su rapidez de entrenamiento, lo que facilita realizar varios experimentos. El segundo ha logrado dar un gran salto en el estado del arte del PLN medido a través del *benchmark* GLUE y a diferencia del primero requiere un gran poder computacional para ser implementado. En este capítulo se mostrará la investigación que derivó en formulación matemática de cada uno, la forma en que se implementan y el motivo de porqué BERT cambia el paradigma de la vectorización de texto.

En seguida de la vectorización de texto se entrena un modelo clasificador. Se establecerá un *baseline* de clasificación de texto, utilizando FastText, experimentando en la forma que se preprocesa el texto, en la utilidad del aumento de datos artificiales para entrenar un clasificador y en la forma de abordar el problema de clasificación *multi-label*. Existen 3 categorías de métodos para resolver el problema de clasificación *multi-label* y se experimentará utilizando un algoritmo por cada uno. En la sección 2.4 se exponen la formulación de cada uno de ellos. Luego de establecer el *baseline* de clasificación se experimentará utilizando BERT, analizando diferentes configuraciones de este y exponiendo detalles de su implementación, que dado su novedad no se han encontrado muchos trabajos que lo implementen en aplicaciones para resolver problemas reales y tampoco para procesar lenguaje en español.

Por otro lado, el resumen extractivo de texto se trata de extraer frases textuales de un texto o conjunto de documentos. Existen múltiples maneras de abordar este problema en este trabajo se experimentará utilizando un algoritmo de *clusterización*. El proceso general consiste en preprocesar el texto, vectorizar los comentarios, agruparlos en *clusters* y seleccionar un subconjunto de frases, en base a heurísticas, que en conjunto pierdan la menor cantidad de información posible respecto de todas las demás.

En este capítulo se exponen los siguientes contenidos en orden: 1) una introducción a conceptos fundamenteles de PLN que se utilizarán para describir los procedimientos de este trabajo, 2) una introducción a los conceptos fundamentales de redes neuronales, 3) la formulación del modelo de lenguaje distribuido FastText, 4) los métodos de clasificación *multi-label* empleados para clasificar texto, 4) la formulación del modelo de propósito general BERT y 5) los métodos de resumen extractivo de texto que se usarán para resumir los textos de la encuesta de satisfacción.

2.1. Conceptos Fundamentales de PLN

A continuación se definen algunos conceptos propios de PLN que se utilizan en la modelación matemática del leguaje, clasificación y resumen de texto. *Corpus* se refiere a el conjunto de textos que se utilizan como set de datos para trabajar, y en este trabajo corresponde a todos los comentarios de la encuesta de satisfacción entregados por la empresa.

Antes de modelar matemáticamente cada comentario, es necesario preprocesar el texto para eliminar elementos que generan ruido en la modelación. El preprocesamiento de texto consiste en primero normalizar el texto removiendo caracteres especiales y corregir faltas de ortografía y gramática. Luego de normalizar se procede a *tokenizar* el texto, que se refiere a segmentar los comentarios de acuerdo a un criterio basado en expresiones regulares y también aplicando procesos de limpieza y transformación de texto, de tal manera que se identifiquen secuencias de caracteres llamadas *tokens*. Usualmente los *tokens* corresponden a las palabras o subpalabras que se presentan en los textos que se preprocesaron. Un *token* también puede ser un elemento gramático como un punto o una coma. El vocabulario con que se trabaja se define como el conjunto de todos los *tokens* distintos que se construyeron luego del preprocesamiento.

Un *n-gram* consiste en una secuencia de caracteres de largo n. Por ejemplo la palabra “reloj” se puede segmentar en los siguientes *3-grams*: “rel”, “elo” y “loj”.

El contexto de una palabra corresponde al conjunto de las palabras que la rodean en un determinado texto, por ejemplo, en la oración “la vendedora me atendió amablemente y en la caja no había fila”, el contexto de la palabra “no” de largo 4 está conformado por el set de palabras {“la”, “caja”, “había”, “fila”}, y el contexto de largo 6 de la palabra “amablemente” corresponde a {“vendedora”, “me”, “atendió”, “y”, “en”, “la”}.

Las *stop words* se refieren a palabras sin contenido informativo y que se repiten mucho a lo largo del *corpus*, por ejemplo las palabras “la”, “una” y “que”. Es una práctica común removerlas del vocabulario porque generan ruido en la representatividad de los vectores de palabras.

Una codificación *one-hot* consiste en la asociación con relación uno a uno de los *tokens* del vocabulario de largo V, ordenados en un índice, a vectores en un espacio $\{0,1\}^V$ con un 1 en la posición correspondiente al índice de cada *token* y ceros en el resto de los componentes del vector. En otras palabras, cada *token* del vocabulario se asocia a un único vector canónico de dimensión V.

2.2. Conceptos Fundamentales de Redes Neuronales

En esta sección se exponen conceptos fundamentales para describir redes neuronales, que se utilizarán luego para describir los modelos FastText y BERT. Una red neuronal usualmente se utiliza para clasificar o predecir. Su unidad fundamental se denomina una unidad, que es análogo a una neurona cerebral, y conceptualmente su función es almacenar información durante el flujo del *input* por la red. Las unidades se agrupan en capas, que están conectadas por los pesos de la red. Los pesos son análogos a las dendritas y axones que conectan neuronas y corresponden a los parámetros de un modelo basado en redes neuronales. En general las redes neuronales se entrenan como cualquier modelo clasificador bajo el concepto de aprendizaje supervisado.

Entrenar un modelo de clasificación mediante aprendizaje supervisado consiste hacer que el modelo sea capaz de asociar ejemplos $\{x^{(1)}, ..., x^{(m)}\}$, con $x^{(i)} \in \mathcal{X}\ \forall i$ a etiquetas $\{y^{(1)}, ..., y^{(m)}\}$, con $y^{(i)} \in \mathcal{Y}\ \forall i$. En este trabajo los ejemplos corresponden a la vectorización de comentarios y las etiquetas a las categorías de eje de negocio junto con su polaridad. Existe un espacio hipótesis Θ donde se busca ajustar $f_\theta : x \in \mathcal{X} \rightarrow z \in \mathcal{Y}$, donde z es la predicción hecha por el modelo, de tal manera que se cumple el objetivo de clasificar. El proceso de aprendizaje es iterativo y se define una función de pérdida $L : (z, y) \in \mathcal{Y} \times \mathcal{Y} \rightarrow e \in \mathbb{R}$, donde e es el error de predicción de una observación. También se define una función de costos J que se busca minimizar, y que en general es la suma de la pérdida de todos los ejemplos disponibles $J(\theta) = \sum_i^m L(f_\theta(x_i), y_i)$. En cada iteración de entrenamiento de la red se actualizan los pesos de esta mediante el proceso de *back-propagation*, que corresponde a propagar el gradiente del error por la red, actualizando los pesos de acuerdo a una tasa de aprendizaje. Así se dice que la red aprende.

En la figura 2.1 se observa un ejemplo de una red neuronal de una capa oculta de tipo *feed-forward fully-connected*, que se refiere a que el *input* fluye a través de la red en una dirección y que cada unidad de cada capa está conectada a todas las unidades de la siguiente capa.

En el ejemplo la primera capa corresponde a la capa de *input*, la segunda es la capa oculta y la tercera es la de *output*. Entre cada capa hay conexiones, que son llamadas los pesos de la red y corresponden a los parámetros del modelo. Estos se pueden definir como la matriz de pesos $W^{(1)} \in \mathbb{R}^{6\times4}$ y $W^{(2)} \in \mathbb{R}^{3\times6}$ junto con sus sesgos asociados $b^{(1)} \in \mathbb{R}^{6\times1}$ y $b^{(2)} \in \mathbb{R}^{3\times1}$ respectivamente. Entre cada capa existe una función de activación, que se utiliza para transformar la multiplicación de la matriz de parámetros y la capa correspondiente. Por ejemplo, en una iteración de la red se introduce un *input* $x^{(1)}$ luego se multiplica por los pesos de la red y la capa intermedia h corresponderá al resultado de la función de activación sobre la multiplicación $h = f_h(W^{(1)}x^{(1)} + b^{(1)})$. Análogamente, la capa de *output* se calcula de la misma manera y $z = f_o(W^{(2)}h + b^{(2)})$.

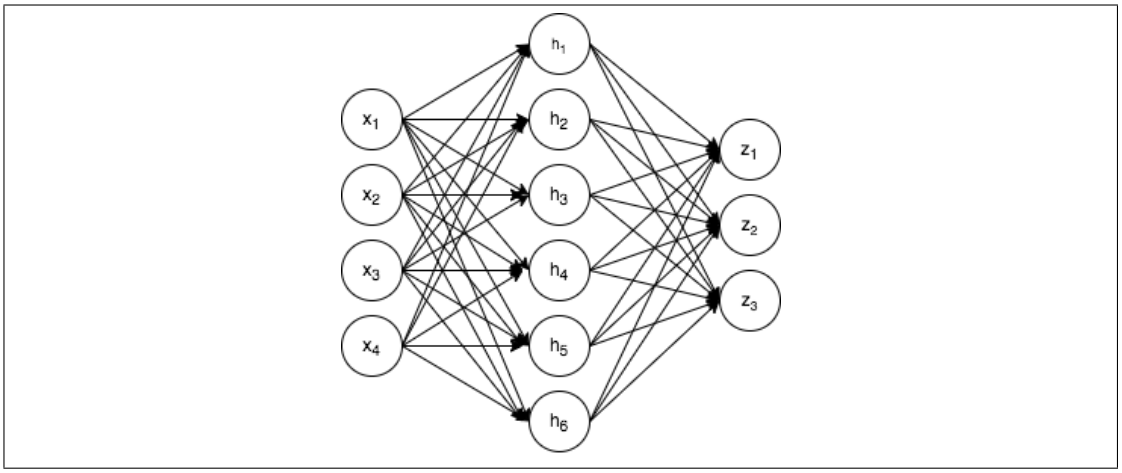


Figura 2.1. Ejemplo red neuronal con una capa oculta. Fuente: Elaboración propia

Las funciones de activación de las capas intermedias en general son lineales $f(x) = x$, o también es común definirlas como la función $ReLU(x) = \max(0, x)$, y esto se hace porque su derivada es poco costosa de calcular. Por otro lado función de activación de la capa de *output* en general es no lineal y define el objetivo de la red, ya que se utiliza en conjunto con la función de pérdida para obtener el error del modelo. Por ejemplo si es necesario que el *output* de la red tenga la forma de una distribución de probabilidad en general se utiliza la función *softmax* $s(x)_i = e^{x_i}/\sum_j e^{x_j}$.

La función de pérdida está relacionada con el objetivo de la red. Si se requiere crear una red que sea un clasificador *multi-label* una opción es utilizar la función sigmoide $f_o(x) = \sigma(x) = 1/(1 + exp(-x))$ como activación de la capa de *output* y utilizar la función de pérdida *Binary Cross Entropy Loss*, que será detallada en la sección 2.4.1. En cada iteración de entrenamiento de la red, una vez calculada la función de pérdida, los pesos de esta se actualizan a través del proceso de *back-propagation* Buscema (1998). Este proceso consiste en calcular el gradiente del error, obtenido a través de la función de costo, y actualizar los pesos de manera que $W^+ = W - \eta \nabla J$, donde W^+ es una matriz de pesos en la siguiente iteración y η es la tasa de aprendizaje. En general se utiliza el descenso del gradiente, sin embargo se pueden utilizar variaciones de este o métodos de optimización más sofisticados. El factor para actualizar los pesos

entre cada capa se calcula a través de la multiplicación de la tasa de aprendizaje η por el gradiente correspondiente a la capa, que se obtiene por la regla de la cadena.

Un problema común en redes muy profundas (con muchas capas) es que el gradiente explota o se hace muy pequeño en las capas iniciales debido a la serie de multiplicación sucedidas por la regla de la cadena, lo que perjudica el aprendizaje de la red y el rendimiento del modelo clasificador. Para mitigar este problema en el modelo BERT se implementan capas especiales como la de normalización *LayerNorm* Lei Ba et al. (2016), cuya funcionalidad es normalizar el vector de una capa interna desplazando cada componente del vector usando el promedio de este y escalando por la desviación estándar y capas de conexión residual He et al. (2015), en que se suma el vector de *input* de una capa interna al vector calculado luego de la función de activación, haciendo que la capa interna se calcule como $h = f_h(Wx + b) + x$.

Los modelos que involucran Aprendizaje Profundo implementan redes neuronales de múltiples capas, cuyo buen funcionamiento se basa en que logran condensar información de una gran cantidad datos en vectores de características. Estos se caracterizan por lograr codificar una representación abstracta de un *input* en la estructura interna de la red, y en general requieren más ejemplos de entrenamiento que modelos de aprendizaje de máquinas tradicionales. Debido a la complejidad de estos modelos, en lugar de describirse por una fórmula matemática, es usual que se describan de acuerdo a la arquitectura de la red y el objetivo que se quiere lograr en cada paso del flujo del *input* por la red.

2.3. Modelo de Lenguaje Distribuido FastText

En esta sección se describe el proceso preprocesamiento y vectorización de texto que son parte de la secuencia de pasos para lograr clasificar los comentarios usando FastText. Una vez entrenado FastText se obtiene una asociación con relación uno a uno de cada *token* del vocabulario a un vector de características. Los vectores de

características (o vectores de palabras) tienen codificado en su estructura interna componentes intrínsecos del lenguaje. Estos se utilizarán para describir vectorialmente cada comentario de la encuesta de satisfacción de tal manera que pueda ser introducido a un modelo de clasificación *multi-label*.

FastText es una extensión del modelo Skip-Gram Mikolov, Yih, & Zweig (2013), y su principal diferencia radica en que Skip-Gram genera vectores de palabras y FastText construye vectores a nivel de caracteres, que luego se utilizan para construir vectores de palabras.

Primero se describe cómo se realizó la etapa preprocesamiento, que implica los pasos de normalización y *tokenización* de texto. Segundo se detalla la formulación de Skip-Gram con el objetivo de lograr un mejor entendimiento de FastText y finalmente se detalla la formulación de FastText.

2.3.1. Preprocesamiento de Texto

El preprocesamiento consiste en normalizar y *tokenizar* el texto para poder indexar las palabras, y luego generar una codificación *one-hot* de los *tokens* formados, que será utilizada para entrenar FastText. El primer paso es normalizar el texto debido a que los comentarios recibidos por la encuesta de satisfacción contienen faltas de ortografía y gramática, caracteres especiales de otros idiomas del dispositivo que se usó para contestar. Estos elementos generan un aumento del tamaño del vocabulario, lo que empeora la representatividad de los vectores de palabras y el desempeño de la clasificación que se realizará posteriormente ya que habrá una mayor cantidad *tokens* únicos y poseerán menor frecuencia de aparición. Por ejemplo, si no se normalizara el texto las palabras “atención” y la palabra mal escrita “atensión” serían representadas como dos palabras distintas, con un vector de palabra distinto asociado a cada una.

La normalización del texto en este trabajo consistió en primero convertir todas las letras de comentarios a minúsculas y reemplazar los caracteres especiales encontrados por caracteres pertenecientes al idioma español si correspondía, o bien eliminarlos.

Luego se procedió a la corrección de faltas de ortografía. Se utilizó la librería *pyspellchecker* Barrus (2018) para realizar una corrección automática del texto. Esta utiliza el algoritmo de Distancia de Levenshtein Levenshtein (1966) para encontrar cadenas de texto creadas a partir de inserciones, eliminaciones, reemplazos y transposiciones de caracteres con una distancia de 2 ediciones o menos de la palabra original. El concepto distancia se refiere a la cantidad de acciones mencionadas realizadas sobre un carácter de la cadena. Por ejemplo, tomando la palabra "atención" como palabra original, la cadena de texto "tención" está a distancia de 1 edición, y la cadena de texto "tensión" está una distancia de 2 ediciones de la palabra original. Una vez calculadas todas las permutaciones posibles con distancia de 2, se eligen las palabras candidatas a corrección dentro de una lista de frecuencia de aparición dentro de un *corpus* de la librería, donde la palabra más probable para reemplazar la falta de ortografía es la que tiene más frecuencia de apariciones.

Se presenta el problema que la librería *pyspellchecker* y otras librerías públicas de corrección automática no contienen todo el vocabulario en español, y en especial el español chileno, que permita corregir eficaz y eficientemente el texto. Además las correcciones automáticas producidas no se adecúan al contexto de una encuesta de satisfacción. Es por esto que se realizó una corrección semiautomática de faltas de ortografía. Primero se construyó un set de 403.391 palabras conocidas del idioma español, uniendo las palabras del lemario de Olea (2013) y de los trabajos de Sosa (2019) y Domínguez (2019). Luego se seleccionó un conjunto de palabras potencialmente corregibles, que consiste en todas las palabras con menos de 10 repeticiones sobre el set de más de 160 mil comentarios de la base de datos entregada por la empresa y que estén presentes dentro de los comentarios del set de entrenamiento. Este criterio se utilizó con el objeto de disminuir la mayor cantidad de ruido dentro del set de datos de entrenamiento, pero considerando todo el vocabulario de la base de datos para que la clasificación automática de texto se vea sesgada de menor manera en ejemplos fuera del set de entrenamiento.

Sobre este set, se hizo una corrección automática con la librería *pyspellchecker*, y además se revisó manualmente que las correcciones sean adecuadas al contexto de la encuesta de satisfacción. Así se corrigieron solo errores ortográficos de palabras que no están en el vocabulario español, y se verificó que la corrección sea adecuada al contexto de la encuesta de satisfacción. La cantidad de palabras corregidas manualmente son 1154.

Luego de normalizar los comentarios, se procede a la *tokenización* de cada uno. FastText construye vectores o *embedings* de palabras que se asocian a cada *token* del vocabulario en lugar de asociarse a las palabras del texto, ya que los *tokens* consisten en palabras o subpalabras del vocabulario y pueden o no incluir secuencias o caracteres especiales como una coma, un signo de exclamación, una composición de palabras u otros elementos que puedan ser modelados vectorialmente como un componente del texto.

Para realizar la *tokenización* se utilizan expresiones regulares, que son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto y posteriormente realizar una segmentación de este. No existe una única forma de *tokenizar* el texto, y un *tokenizador* se diferencia de otro dependiendo de los criterios de expresiones regulares utilizadas y los procesos de limpieza y transformación de texto empleados. Luego de identificar los *tokens* realizando la segmentación de acuerdo a las expresiones regulares se remueven las *stop words* con el objetivo de disminuir el ruido que generan las palabras no informativas del texto durante el entrenamiento de FastText.

En la tabla 2.1 se pueden ver ejemplos de 3 *tokenizadores* distintos, que usan distintas expresiones regulares para generar diferentes *tokenizaciones*. La primera toma en cuenta caracteres especiales “%” y “$” para poder distinguir cadenas de texto relacionadas con precios y porcentajes, la segunda solo toma en cuenta caracteres alfanuméricos y la tercera solo las letras del alfabeto.

Tabla 2.1. Ejemplos de tokenizaciones.

Comentario original	"Una de las poleras marco en caja $9990, pero la prenda decia $14990 y $10490llevaba otra polera de $14990, y por TODA la tienda salia segunda polera de ciertas marcas con 50% de descuento.En la Boleta solo hizo descuento de 2490. AL reclamarle al cajero me dice que la polera costaba $14990, por mas de insistirle q"
Tokenizador 1	['poleras', 'marco', 'caja', '$9990', 'pero', 'prenda', 'decia', '$14990', '$10490', 'llevaba', 'polera', '$14990', 'toda', 'tienda', 'salia', 'segunda', 'polera', 'ciertas', 'marcas', '50%', 'descuento', 'boleta', 'solo', 'hizo', 'descuento', '2490', 'reclamarle', 'cajero', 'dice', 'polera', 'costaba', '$14990', 'mas', 'insistirle']
Tokenizador 2	['poleras', 'marco', 'caja', '9990', 'pero', 'prenda', 'decia', '14990', '10490llevaba', 'polera', '14990', 'toda', 'tienda', 'salia', 'segunda', 'polera', 'ciertas', 'marcas', '50', 'descuento', 'boleta', 'solo', 'hizo', 'descuento', '2490', 'reclamarle', 'cajero', 'dice', 'polera', 'costaba', '14990', 'mas', 'insistirle']
Tokenizador 3	['poleras', 'marco', 'caja', 'pero', 'prenda', 'decia', 'llevaba', 'polera', 'toda', 'tienda', 'salia', 'segunda', 'polera', 'ciertas', 'marcas', 'descuento', 'boleta', 'solo', 'hizo', 'descuento', 'reclamarle', 'cajero', 'dice', 'polera', 'costaba', 'mas', 'insistirle']

Una vez finalizada la tokenización de un comentario de la encuesta, se unen los *tokens* de este usando espacios entre cada uno, conformando un texto que se utilizará para entrenar FastText. Si el comentario original es "Porque todo es hermoso, hay variedad y la atencion muy buena!" un posible ejemplo de texto que se usará en FastText es "todo hermoso , hay variedad atencion buena !", y este puede variar dependiendo de la expresión regular utilizada.

2.3.2. Formulación de Skip-Gram

Una vez *tokenizado* el texto se procede a vectorizarlo con el objetivo de que posteriormente pueda ser clasificado. En este trabajo se utiliza FastText, que corresponde a una extensión del modelo de lenguaje distribuido Skip-Gram Mikolov, Sutskever, et al. (2013) cuyo tiempo de entrenamiento es log-lineal y se destaca por su rapidez de entrenamiento y buen desempeño en comparación con previos modelos. En esta sección primero se muestra porqué se dice que los vectores de palabras almacenan componentes intrínsecos del lenguaje y luego se procede a la detallar la

formulación de Skip-Gram con el objetivo de facilitar la especificación teórica de FastText.

Skip-Gram atrajo gran atención de la comunidad científica al campo de investigación de PLN debido a la alta calidad de sus representaciones vectoriales y a su velocidad de entrenamiento. En Mikolov, Sutskever, et al. (2013), los autores describen que la calidad de los vectores se entiende como su capacidad de generalizar y capturar propiedades intrínsecas del lenguaje de manera abstracta en el espacio en que están definidos. Esto se muestra en el posterior trabajo Mikolov, Yih, & Zweig (2013), que expande y analiza los resultados del primero. En este se indica que los vectores de palabras generados por el modelo Skip-Gram tienen múltiples grados de similaridad y son capaces de reflejar regularidades semánticas y sintácticas en el lenguaje en el espacio, y cada relación entre palabras está caracterizada por un desplazamiento de los vectores. En este caso, el desplazamiento de un vector se refiere a la suma y resta de vectores de palabras o a una traslación de vectores en una determinada magnitud y dirección.

En la figura 2.2, a la izquierda se ilustran proyecciones de vectores que representan las palabras "*man*", "*uncle*" y "*king*", y cuyos desplazamientos en la misma dirección y magnitud en el espacio vectorial donde están definidos resultan en vectores de palabras "*woman*", "*aunt*" y "*queen*" respectivamente, mostrando que se ha logrado capturar la relación de género en el desplazamiento. A la derecha se ilustra que a través del desplazamiento representado por la flecha roja se ha logrado capturar la relación de sintaxis singular/plural.

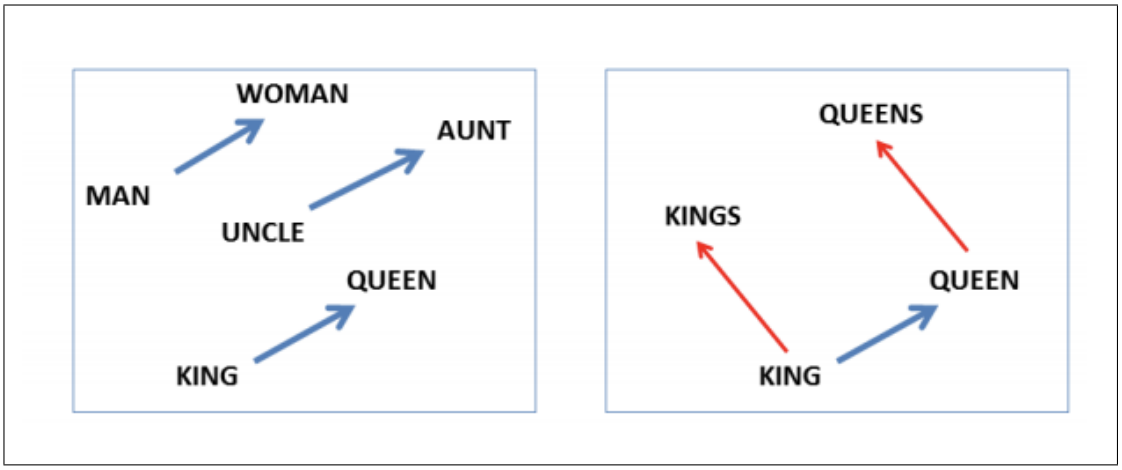


Figura 2.2. A la izquierda tres pares de desplazamientos de vectores ilustrando la relación de género. A la derecha se muestra una proyección distinta mostrando la relación singular/plural de dos palabras. Fuente: Mikolov, Yih, & Zweig (2013)

En Mikolov, Yih, & Zweig (2013) se explica que los vectores exhiben composicionalidad, que quiere decir que la suma de dos vectores resulta en un vector que es la composición de las palabras que representa cada uno. Presenta el ejemplo de que los vectores generados por el modelo de lenguaje distribuido que representan a las palabras "*king*", "*man*", "*woman*" y "*queen*" se relacionan de tal manera que $vector(king) - vector(man) + vector(woman) \approx vector(queen)$, mostrando que los componentes de los vectores almacenan regularidades del lenguaje.

FastText y Skip-Gram modelan vectores de palabras y no vectores de comentarios. En la práctica se necesita la representación vectorial de un comentario que sirva como vector de ejemplo para el modelo clasificador. Debido a la propiedad de composicionalidad de los vectores de palabras se justifica que la representación vectorial de un comentario sea el promedio de todos los vectores de palabras que lo componen. Por ejemplo la representación vectorial del comentario preprocesado "tienda ordenada" será $(vector(tienda) + vector(ordenada))/2$. A continuación se procede a detallar la base teórica y formulación de Skip-Gram.

Inspirado en hipótesis distribucional Harris (1954) que indica que las palabras que ocurren en contextos similares tienden a tener significados similares, el objetivo de entrenamiento de Skip-Gram es aprender representaciones vectoriales de palabras que sean buenas prediciendo las palabras que la rodean. Este tipo de modelos funciona porque en general se entrenan sobre un *corpus* de millones de palabras, por ejemplo toda la Wikipedia, ya que así pueden adquirir un buen nivel generalización y representatividad. En este trabajo se utilizará como *corpus* el set de más de 160 mil comentarios.

El objetivo del entrenamiento de Skip-Gram es que dada una palabra, el modelo logre predecir el contexto de esta. El modelo consiste en una red neuronal de tipo *fully-connected* con una capa oculta, en que el *input* de la red se relaciona a una palabra de una oración en el *corpus*, y se busca que el *output* de la red se relacione al contexto de la palabra de *input* introducida. Si se define el largo del contexto 4, los ejemplos de entrenamiento del modelo consistirán en todas las secuencias de 5 palabras que se presentan en los textos del *corpus*, donde el *input* consistirá en la palabra central y se buscará igualar el *output* a la primera, segunda, cuarta y quinta palabra de la secuencia. A continuación se detallan los componentes de la red, un ejemplo del esquema de entrenamiento y luego se describe cómo se generan los vectores de palabras y la función objetivo del modelo.

La capa de *input* de la red es de tamaño $V \times 1$, V correspondiente al tamaño del vocabulario, la capa oculta es de tamaño $D \times 1$, D arbitrario, con $D < V$. En este trabajo, el tamaño del vocabulario varía dependiendo del *tokenizador* utilizado y es aproximadamente 38.000. También eligió $D = 300$ debido a la recomendación del sitio de Facebook Open Source *fastText.cc* (n.d.). Los pesos que conectan ambas capas se representan por la matriz W de dimensión $D \times V$. La capa de *output* es de tamaño $CV \times 1$, donde C es el largo definido del contexto. Los pesos que unen la capa oculta con la de *output* se representan por matrices $W'_1, W'_2, \ldots, W'_C$ de dimensión $V \times D$ y el vector de *output* de la red corresponde a la concatenación de los vectores resultantes

de la multiplicación entre el vector de la capa oculta h y cada matriz W_i'. En la figura 2.3 se visualiza un esquema de la red neuronal que compone el modelo.

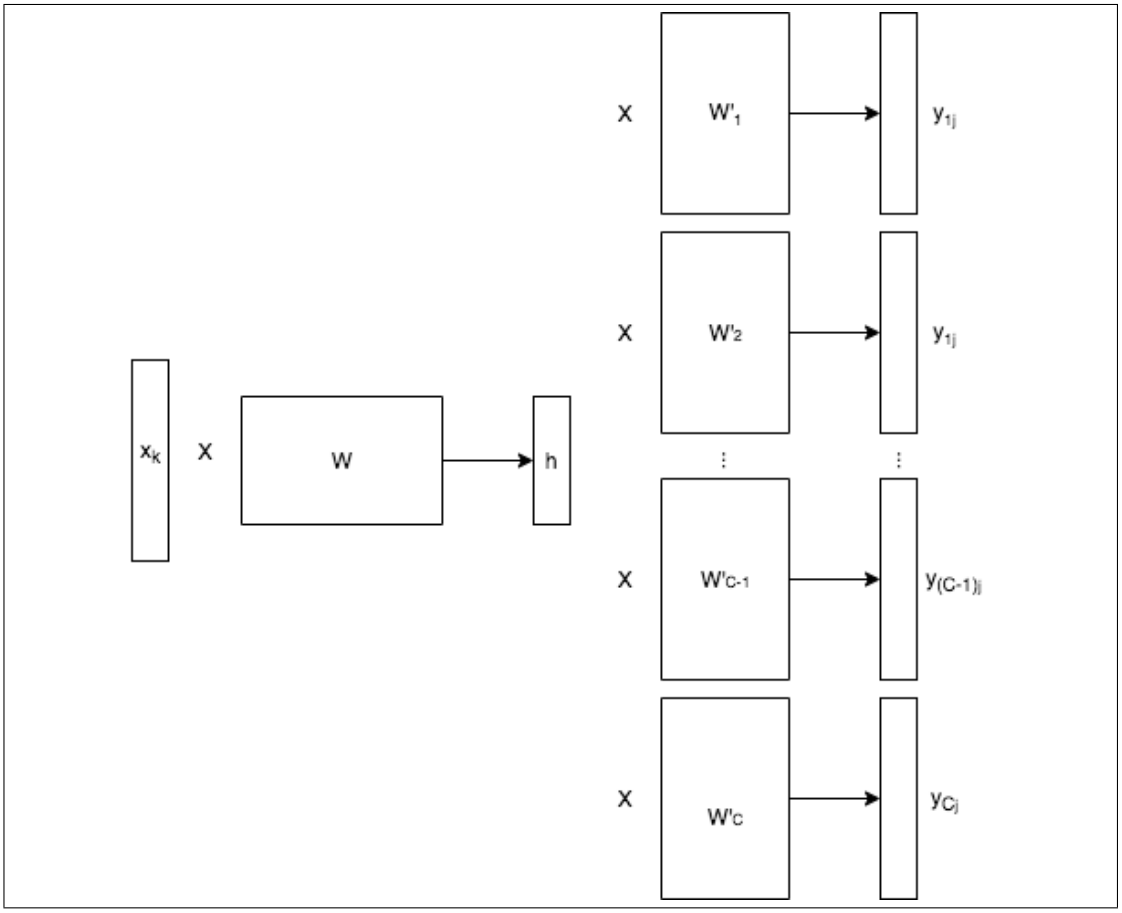


Figura 2.3. Modelo Skip-Gram. Fuente: Elaboración propia

A modo de ejemplo se describirá un esquema de una iteración del entrenamiento de la red. Antes de entrenar el modelo se construye el vocabulario con que se trabaja, que corresponde a todos los *tokens* distintos en el *corpus* preprocesado, a cada *token* distinto se le asocia un índice entre 1 y V de acuerdo a su orden de aparición en el *corpus*, y también se les asocia un vector con codificación *one-hot* con un 1 en la posición de su índice y 0 en todas las otras posiciones. Luego, como ejemplo de entrenamiento se tiene el comentario "Necesité información sobre la prenda que compré y me la dieron muy satisfactoriamente", el texto preprocesado corresponde a "necesite informacion sobre prenda compre dieron satisfactoriamente", y a

representación *one-hot* de dimensión $V \times 1$ de cada *token* del texto corresponderán a los vectores canónicos de $\mathbb{R}^V$ $e_1, e_2, ..., e_7$ respectivamente. Se ha definido que el largo del contexto es 4, y se construye y_1, de dimensión $4V \times 1$, que corresponde a la concatenación de los vectores e_1, e_2, e_4 y e_5. El primer ejemplo de entrenamiento del modelo será el par (e_3, y_1). El vector e_3 será introducido como *input* a la red, y se define una función de pérdida de tal manera que la red pueda predecir y_1 a partir del *input* e_3. Luego se computa el error entre el *output* generado y el vector y_1, y se procede a actualizar los pesos de la red con el algoritmo de *back-propagation*. El segundo ejemplo de entrenamiento de la red consistirá en el par (e_4, y_2), donde y_2 será la concatenación de los vectores e_2, e_3, e_5 y e_6.

Una vez entrenado el modelo, los vectores de palabras generados por Skip-Gram corresponderán a las columnas de la matriz W de dimensión $D \times V$ que representa los pesos de la red entre la capa *input* y la capa oculta, y existe una relación uno a uno entre cada *token* y la cantidad de vectores generados. Es decir, después de entrenar Skip-Gram, este no se utiliza como modelo predictor como sería lo usual, en cambio, se extrae la matriz W de dimensión correspondiente a los pesos de la red que unen la capa de *input* y la capa oculta, que representa la codificación interna que ha generado la red y donde se ha almacenado de manera abstracta la información intrínseca del lenguaje. La representación vectorial del token con índice i en el vocabulario corresponderá a la columna i de W, ya que la multiplicación de su codificación *one-hot* con W, retornará $W_{(\cdot,i)}$ de dimensión $D \times 1$.

Luego de la presentación del esquema de la red se procede a detallar la formulación matemática. Formalmente, dado una secuencia de palabras de entrenamiento $w_1, w_2, ..., w_T$ el objetivo del modelo es maximizar la log-probabilidad

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{2.1}$$

donde $C = 2c$ es el tamaño del contexto de entrenamiento, y en general $c \in \{2, 3, 4, 5\}$.

La función de activación de primera capa es lineal y dado que el *input* tiene codificación *one-hot*, durante el entrenamiento, simplemente se copian los pesos de la k-ésima fila de W a la capa oculta h, ya que dada la codificación *one-hot* del k-ésimo *token* simbolizado como x_k, se tiene que $Wx_k = W_{(\cdot,k)} = h$. Los pesos entre la capa oculta y la de *output* se simbolizan por las matrices W_i' de $V \times D$, para $i \in \{1, ..., C\}$, y la función de activación también es lineal.

Basado en el trabajo de Rong (2014) se procede a definir el "vector de *input*" y el "vector de *output*" con el objetivo de simplificar la formulación. Para la k-ésima palabra del vocabulario, se define como "vector de *input*" la k-ésima columna de W, es decir $v_{w_I} := Wx_k = W_{(\cdot,k)} = h$. Se define v'_{ij} como el "vector de *output*" de la j-ésima palabra en el vocabulario, en la posición i del contexto y es equivalente a la j-ésima fila de W_i'. La definición de estos vectores son útiles para definir la función objetivo.

La formulación del término $p(w_{t+j}|w_t)$ en la ecuación 2.5 originalmente se define usando la función *softmax*

$$p(w_O|w_I)_i = \frac{\exp(v'^{\top}_{w_O} v_{w_I})}{\sum_{w=1}^{V} \exp(v'^{\top}_{w} v_{w_I})} \tag{2.2}$$

donde i corresponde a la posición del contexto, v_w y v'_w son las representaciones de "*input*" y "*output*" de w. Como función de pérdida se utiliza la función de pérdida de Entropía Cruzada, definida como

$$L(z, y) = -\sum_{i}^{C} \sum_{j}^{V} y_{i,j} \log z_{i,j} \tag{2.3}$$

En conjunto, la activación *softmax* con la pérdida de Entropía Cruzada se utilizan para que el modelo prediga solamente una palabra por posición del contexto y que pueda aprender actualizando sus pesos mediante *back-propagation*. Sin embargo, la gran desventaja es que para calcular el error se requiere multiplicar todos los parámetros de las matrices asociadas a las palabras del contexto por el vector de la capa oculta y es costoso en términos de tiempo computacional. Es por esto que en

Mikolov, Sutskever, et al. (2013) los autores sugieren el uso de la función de pérdida de Muestreo Negativo en lugar de utilizar la función *softmax* junto con la pérdida de Entropía Cruzada. El uso de muestreo negativo disminuye el tiempo de entrenamiento y aumenta la exactitud del modelo, medido como la calidad de la regularidad sintáctica y semántica mostrada en tareas específicas.

Los autores presentan el muestreo negativo como una versión simplificada de la Estimación de Contraste de Ruido, introducida por Gutmann & Hyvärinen (2012), y que consiste en seleccionar una muestra de palabras (de 5 a 20) que no corresponden al contexto para actualizar los pesos de la red en cada iteración, en lugar de actualizar todos los pesos como se hace con la función de pérdida de Entropía Cruzada. El muestreo se hace de acuerdo a una distribución arbitraria P_n, y los autores sugieren utilizar $P_n(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=0}^{n} f(w_j)^{\frac{3}{4}}}$ ya que se obtuvo empíricamente mejores resultados, donde $f(w_i)$ indica la frecuencia de aparición de la palabra w_i en el *corpus*.

La selección de muestras negativas tiene por objetivo reducir la cantidad de multiplicaciones que se requieren para calcular el *output* de la red, y actualizar los pesos asociados a palabras que no corresponden a la palabra correcta, de manera que el modelo aprenda a distinguir qué palabra es adecuada al contexto y qué palabras no. Entonces se transforma el objetivo del modelo a una log-verosimilitud negativa que se desea minimizar y se define como

$$\log(1 + e^{-({v'}_{w_O}^{\top} v_{w_I})}) + \sum_{n \in N_{t,c}} \log(1 + e^{({v'}_{w_n}^{\top} v_{w_I})}) \tag{2.4}$$

donde ${v'}_{w_O}^{\top}$ es el "vector de *output*" de la palabra correcta del contexto, v_{w_I} es el "vector de *input*" y $N_{t,c}$ es el conjunto de muestras negativas seleccionadas en cada iteración.

En la práctica, en cada iteración, la red multiplicará el "vector de *input*" por un número pequeño de "vectores de *output*", y no todas las matrices asociadas al contexto. Por ejemplo si el número de muestras negativas es 10, por cada palabra del contexto que se está calculando, para calcular el *output* de la red se requiere hacer 11 multiplicaciones, donde una es la multiplicación asociada a la palabra correcta, y 10

son las asociadas a palabras seleccionadas al azar que no corresponden a la palabra correcta. Por otro lado, usando la Entropía Cruzada se requerían V multiplicaciones por cada palabra del contexto. Luego el proceso de *back-propagation* actualiza los pesos de la red utilizando el gradiente asociado a los vectores utilizados para calcular ese *output* y de esta manera el tiempo computacional para entrenar el modelo logra ser log-lineal.

2.3.3. Formulación de FastText

Hasta ahora se ha detallado la forma de *tokenizar* el texto y cómo se modela vectorialmente con Skip-Gram. FastText es una extensión de este y se define a partir de una red neuronal de arquitectura similar. La principal diferencia entre los dos es que FastText aborda el problema generando una vectorización a nivel de los caracteres y un vector de palabra es formado por la suma de los vectores de caracteres que lo componen. En esta subsección se describe la formulación del modelo FastText utilizado para vectorizar el texto en el *baseline* de clasificación de este trabajo. Primero se muestra la transformación de la función objetivo de Skip-Gram a una notación simplificada con el objetivo de entender la lógica detrás de FastText, luego se muestra como se identifican las secuencias de caracteres que estarán asociadas a vectores de caracteres y finalmente se muestra la formulación del objetivo del modelo.

Dado un *corpus* de entrenamiento representado por la secuencia de palabras $w_1, ..., w_T$ y de vocabulario con tamaño V, el objetivo es aprender la representación vectorial de cada palabra definida por su índice $i \in \{1, ..., V\}$. Similar a Skip-Gram el objetivo del modelo FastText es maximizar log-verosimilitud

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{2.5}$$

Se denota la función de pérdida logística $l : x \to \log(1 + e^{-x})$, se define una función de puntaje s que mapea los pares (palabra, contexto) a puntajes en $\mathbb{R}$, para un contexto elegido en la posición c, y se reescribe la función de Muestreo Negativo definida en la ecuación 2.4, siendo ahora nuevo objetivo del modelo y planteado como

una log-verosimilitud negativa que se desea minimizar

$$\sum_{t=1}^{T}\left[\sum_{c\in\mathcal{C}_t} l(s(w_t,w_c)) + \sum_{w_n\in N_{t,c}} l(-s(w_t,w_n))\right] \tag{2.6}$$

donde $\mathcal{C}_t$ es el set de índices de palabras al rededor de w_t. En el modelo Skip-Gram se define el puntaje

$$s(w_t,w_c) = v_{w_t}^{\top} v'_{w_c} \tag{2.7}$$

siendo v_{w_t} el "vector de *input*" y v'_{w_c} el "vector de *output*" definidos en la sección anterior.

El modelo Skip-Gram de Mikolov, Sutskever, et al. (2013) ignora la estructura interna de las palabras, por lo que los autores de FastText introducen un modelo de subpalabras, donde cada palabra w es representada por un conjunto de *n-grams* $\mathcal{G}_w$. Para cada *token* w del vocabulario, el conjunto $\mathcal{G}_w$ se construye segmentándolo en las secuencias de caracteres consecutivos de largo n que lo componen (*n-grams*). A continuación se describe cómo se identifican los *n-grams* asociados a cada palabra y que posteriormente serán vectorizados.

Durante el proceso de identificación de *n-grams* que componen a w se añaden símbolos especiales de límite "<" y ">" al principio y final de cada palabra, lo que permite distinguir prefijos y sufijos entre otras secuencias de caracteres. Por ejemplo, la palabra "cajera" y eligiendo $n = 3$ será subdividida en los siguientes *n-grams* de 3 caracteres (*3-gram*) "<ca", "caj", "aje", "jer", "era" y "ra>". Además, el conjunto de *n-grams* asociado a "cajera" considerará la secuencia especial "<cajera>". Finalmente, el conjunto de *n-grams* asociado a "cajera" será $\mathcal{G}_{cajera}$ ={<cajera>, <ca, caj, ...}. En la práctica, para construir este conjunto de *n-grams* asociados a w, los autores postulan extraer todos los *n-grams* para n mayor o igual a 3 y menor o igual que 6.

Al final del entrenamiento de Skip-Gram se obtienen V vectores de palabras, un vector asociado a cada *token* del vocabulario de tamaño V obtenido del preprocesamiento del *corpus* con que se trabajó. En cambio al final del entrenamiento

de FastText, se obtendrán G vectores de subpalabras, cada uno asociado a cada *n-gram* distinto obtenido del proceso descrito en el párrafo anterior aplicado a todas las palabras que componen el vocabulario y la representación de un *token* w del vocabulario se hará a través de la suma de los vectores asociados a cada *n-gram* que lo compone.

Para realizar lo mencionado, se construye un diccionario de *n-grams* de tamaño G y cada palabra w se relaciona a su set de *n-grams* correspondiente $\mathcal{G}_w \subset \{1, ..., G\}$. Luego a cada *n-gram* g se le asocia un vector de representación z_g y se reemplaza la función de puntaje s de Skip-Gram definida en 2.7 por la suma del producto punto entre los "vectores de *input*" de los *n-grams* correspondientes a $\mathcal{G}_w$ y el "vector de *output*", como se expone a continuación

$$s(w, c) = \sum_{g \in \mathcal{G}_w} z_g^T v'_c \tag{2.8}$$

La función de puntaje s es incorporada en la función objetivo del modelo, y de esta manera se logra utilizar representaciones compartidas a través de las palabras, obteniendo representaciones confiables de palabras que no están presentes en el vocabulario de entrenamiento. Además se genera una mejor robustez de la calidad de los vectores respecto de texto con faltas de ortografía, como lo es el de esta encuesta.

En cuanto a la arquitectura de la red neuronal, no se encontró literatura que la detalle, sin embargo a partir de la función de puntaje s se puede inferir que es similar a Skip-Gram, con la diferencia que la matriz W que conecta la capa de *input* con la capa oculta es de dimensión $D \times G$, y las matrices W'_i no cambian respecto del modelo anterior. Ahora, dada una palabra w, el *input* introducido al modelo no tendrá codificación *one-hot*, sino que tendrá un 1 en cada índice correspondiente a cada *n-gram* incluido en $\mathcal{G}_w$. La función de activación entre la capa de *input* y la capa oculta también será lineal de manera que $h = Wx_w = \sum_{g \in \mathcal{G}_w} W_{(\cdot,g)}$, es igual a la suma de las columnas de W correspondientes a los *n-grams* que componen la palabra w. Finalmente la función de activación entre la capa de oculta y la de *output* estará dada por la función de Muestreo Negativo.

La implementación de FastText en este trabajo se realizó utilizando de la librería *fasttext* Bojanowski et al. (2016) de Python. Se utilizaron los parámetros de entrenamiento predefinidos en la librería donde tamaño del contexto es 10, largo del *n-gram* mínimo 3, largo del *n-gram* máximo 6, número de muestras negativas 5 y la función de pérdida es Muestreo Negativo. El único parámetro que se cambió fue la dimensión de los vectores de salida y se eligió $D = 300$.

2.4. Clasificación Multi-Label de Texto

En este trabajo se pretende realizar un *baseline* de clasificación utilizando FastText para vectorizar el texto y analizar cómo el preprocesamiento del texto afecta el resultado de la clasificación. El proceso general de clasificación texto consiste en preprocesar el texto (*tokenización*), vectorización de este, entrenamiento del modelo clasificador y ajuste de los hiperparámetros del modelo clasificador. Se mostró cómo se realiza la *tokenización* y vectorización y a continuación se expone en qué consiste el proceso de clasificación de este.

Se desea clasificar los comentarios de acuerdo a los de ejes de negocio que tratan y si es bueno o malo respecto del rendimiento esperado por cada uno de ellos. La empresa mide 13 aspectos del negocio, lo que en total generaría en total 26 posibles categorías a las que podría estar asociado un comentario, de la forma (<eje de negocio>, positivo) o (<eje de negocio>, negativo). El objetivo es automatizar la clasificación de texto entrenando un modelo clasificador que sea capaz de asociar un comentario al eje de negocio del que trata, con su respectiva polaridad, con el criterio que el personal de la empresa usaría para clasificarlo.

El entrenamiento de un modelo clasificador se realiza mediante aprendizaje supervisado, a diferencia del entrenamiento del modelo de lenguaje distribuido FastText, que se realiza de manera no supervisada. Esto se refiere que el modelo aprende a partir de un set de datos de entrenamiento en que se ha asociado manualmente los comentarios a sus categorías correspondientes. En conjunto con el

equipo que creó la encuesta de satisfacción de la empresa se construyó un set datos de entrenamiento etiquetando aproximadamente 5.400 comentarios manualmente para entrenar el modelo.

El problema de asignación de etiquetas (<eje de negocio>, <polaridad>) a comentarios de la encuesta de satisfacción se puede modelar como un problema de clasificación *multi-label*, en que se permite que múltiples o cero etiquetas puedan ser asignadas a un ejemplo, a diferencia de la clasificación *multi-clase* en que un ejemplo tiene asociada una sola etiqueta.

En el problema de clasificación *multi-label* se define un espacio $\mathcal{X} \subseteq \mathbb{R}^D$ donde están definidos los ejemplos que se desean clasificar y un set de etiquetas $L = \{\lambda_1, ..., \lambda_k\}$. Se dispone de un set de ejemplos $E = \{(x^1, y^1), ..., (x^M, y^M) | x^i \in \mathcal{X}, y^i \in \{0,1\}^k \ \forall i \in 1, ..., M\}$ donde M es la cantidad de ejemplos disponibles y cada ejemplo x está asociado a un vector y de dimensión $|L| = k$, e $y_j = 1$ indica que la j-ésima etiqueta es relevante para el ejemplo, e $y_j = 0$ indica que la j-ésima etiqueta no es relevante. La tarea es aprender una función $h : \mathcal{X} \rightarrow \{0,1\}^k$ que asigne un subconjunto de L a cada ejemplo entregado, tal que maximice un criterio de calidad q que recompensa a modelos con alta precisión predictiva y baja complejidad.

En el *baseline* de clasificación *multi-label*, para vectorizar el texto se utiliza el modelo de lenguaje distribuido FastText ya que tiene menor tiempo de ejecución en comparación con los modelos de lenguaje de propósito general que alcanzan el actual estado del arte en clasificación de textos y esto permite realizar una mayor cantidad de experimentos. Se experimentará implementando 3 diferentes algoritmos de clasificación *multi-label*, cada uno correspondiente a un conjunto de métodos distintos para afrontar el problema de clasificación *multi-label*.

Existen 3 categorías de métodos para resolver el problema de clasificación *multi-label*. La primera, Adaptación de Algoritmos son métodos que adaptan, extienden y personalizan métodos existentes de aprendizaje de máquinas. La segunda

categoría es Transformación del Problema, que consiste en métodos que transforman el problema de aprendizaje *multi-label* a problemas de clasificación de una sola etiqueta o a un problema de regresión. La tercera es Metodos de Ensamblaje, cuyos métodos se desarrollan sobre los de transformación de problema o de adaptación de algoritmos, ensamblando varios clasificadores, que en conjunto toman una decisión para la clasificación final. En esta etapa se experimentará utilizando un algoritmo por cada forma de abordar el problema, los que son *Multi Layer Perceptron Classifier*, *Classifier Chains* y *Random K-Labelsets*, respectivamente.

2.4.1. Multi-Layer Perceptron Classifier

Multi-Layer Perceptron Classifier (MLP) consiste en una red neuronal *feed-forward*, en que en la capa de *output* existe una unidad por cada clase. El clasificador implementado en este trabajo se basó en los expuestos en los trabajos de Zhang & Zhou (2006) y Nam et al. (2013). En ambos se utiliza una red neuronal con una capa oculta y la capa de *output* se usa para producir puntajes de predicción para las etiquetas en el intervalo $(0, 1)$ y estos se convierten en categorías binarias usando una función de *threshold*. A diferencia de los modelos presentados en los trabajos mencionados, el clasificador implementado en este consta de dos capas ocultas ya que experimentos preliminares, en que se configuraron múltiples opciones de tamaños de capas ocultas y funciones de activación para cada capa, se determinó que los resultados eran levemente mejores.

Se requiere predecir un sub conjunto de $L = \{\lambda_1, ..., \lambda_k\}$ de etiquetas relevantes dado un vector de *input* de dimensión D. El clasificador es una red neuronal de 4 capas, en que la capa de *input* $x \in \mathbb{R}^{D\times 1}$ y es conectada a la primera capa oculta de F unidades $h_1 \in \mathbb{R}^{F\times 1}$, a través de la matriz de pesos $W^{(1)} \in \mathbb{R}^{F\times D}$ y sesgos $b^{(1)} \in \mathbb{R}^{F\times 1}$. Esta se conecta a la segunda capa oculta h_2 también de F unidades mediante la matriz de pesos $W^{(2)} \in \mathbb{R}^{F\times F}$ y sesgos $b^{(2)} \in \mathbb{R}^{F\times 1}$. Finalmente la capa h_2 se conecta a las unidades de *output* $o \in \mathbb{R}^{|L|\times 1}$ a través de los pesos $W^{(3)} \in \mathbb{R}^{|L|\times F}$ y sesgos $b^{(3)} \in \mathbb{R}^{|L|\times 1}$. El clasificador $f_\Theta(x)$ es una red neuronal de tipo *feed-forward*,

cuya forma vectorial es

$$f_\Theta(x) = f_o(W^{(3)} f_{h_2}(W^{(2)} f_{h_1}(W^{(1)}x + b^{(1)}) + b^{(2)}) + b^{(3)}) \tag{2.9}$$

donde $\Theta = \{W^{(1)}, b^{(21}, W^{(2)}, b^{(2)}, W^{(3)}, b^{(3)}\}$ y f_{h_1}, f_{h_2} y f_o son las funciones de activación entre cada capa. El objetivo es encontrar un $\theta \in \Theta$ que minimice la función de pérdida $J(\Theta; x, y)$, que mide el error entre la predicción y los valores reales.

En la etapa de *baseline* se utilizará la función de pérdida *Focal Loss* introducida por Lin et al. (2017) que es una extensión de la Entropía Cruzada Binaria (*Binary Cross-Entropy Loss*), una función de pérdida que se adecua al problema de *multi-label ranking* y que apunta a nivelar el desbalance de clases.

La función de pérdida *Binary Cross-Entropy Loss* se define como

$$J_{CE}(\Theta; x, y) = -\sum_i (y_i \log p_i) + (1 - y_i) \log(1 - p_i) \tag{2.10}$$

donde $y_i \in \{0, 1\}$ es el objetivo de clasificación, y $p_i \in (0, 1)$ es la predicción de si la etiqueta λ_i es relevante para el ejemplo. Cuando el ejemplo no es relevante ($y_i = 0$) la función en cuenta el término de la derecha y es mayor cuando p_i es cercano a 1 y cuando el ejemplo es relevante ($y_i = 1$) la función solo toma en cuenta el valor de la izquierda y es mayor cuando la predicción es cercana a 0.

Por otro lado, *Focal Loss* está diseñada para la detección de objetos en un escenario con un desbalance de clases extremo. Lin et al. (2017) proponen un factor modulador que reajusta la función de pérdida *Binary Cross-Entropy Loss* dándole mayor importancia a los ejemplos "difíciles", cuando $p_i \approx 0.5$ y restándole importancia a los ejemplos más "fáciles", es decir los que de acuerdo al modelo tienen una probabilidad $p_i \approx 1$ cuando $y_i = 1$ y $p_i \approx 0$ cuando $y_i = 0$. El factor modulador se define como $(1 - p_i)$ para la clase $y_i = 1$ y p_i para la clase $y_i = 0$. Además se incluyen dos hiperparámetros ajustables. Estos son el parámetro de enfoque $\gamma \geq 0$ y un factor de balance que es $\alpha \in [0, 1]$ para la clase $y_i = 1$ y es $(1 - \alpha)$ para la clase $y_i = 0$ que apunta a regular el desbalance de clases

disminuyendo el valor de la pérdida aportada cuando la predicción es correcta. La función de pérdida *Focal Loss* se define como

$$J_{FL}(\Theta; x, y) = -\sum_{l} \alpha(1-p_i)^{\gamma}(y_i \log p_i) + (1-\alpha)p_i^{\gamma}(1-y_i)\log(1-p_i) \quad (2.11)$$

Con respecto a los componentes de la red neuronal, la función de activación de las capas ocultas será $f_{h_1}(x) = f_{h_2}(x) = ReLU(x) = \max(0, x)$, ya que ha mostrado tener una mejor generalización y rendimiento computacional en investigaciones en el área de *deep learnning* Glorot et al. (2011). Se utilizará la función sigmoide como función de activación en la capa de *output*, quedando así $f_o(x) = \sigma(x) = 1/(1 + exp(-x))$. El algoritmo de optimización utilizado en el proceso de back-propagation es Adam Kingma & Ba (2014) debido a su velocidad y estabilidad Reddi et al. (2018). Se utiliza la constante 0.5 como función de *threshold* para determinar si la etiqueta λ_i es relevante al ejemplo correspondiente de acuerdo a la probabilidad p_i determinada por el modelo.

2.4.2. Classifier Chains

Classifier Chains (CC) es un método propuesto por Read et al. (2009), en que se construyen $C_1, ..., C_k, k = |L|$ clasificadores binarios unidos simulando una cadena, donde cada uno resuelve un problema de *Binary Relevance* asociado a la etiqueta $\lambda_j \in L$. *Binary Relevance* es un método de clasificación *multi-label* caracterizado como método de Transformación del Problema que consiste en construir un clasificador binario para cada etiqueta posible y la predicción de etiquetas asociada a cada ejemplo se hace de manera independiente. En cambio, en CC cada eslabón de la cadena $C_1, ..., C_k$ de clasificadores binarios se entrena de manera consecutiva, incluyendo la información generada por el eslabón anterior. Cada clasificador C_j es responsable de aprender y predecir la asociación binaria de la etiqueta λ_j dado el espacio de características, ampliado por las predicciones de los clasificadores binarios anteriores $C_1, ..., C_{j-1}$. Es decir, el clasificador C_j debe predecir la pertenencia de la etiqueta λ_j a un ejemplo x dado, utilizando el vector a $(x_1, ..., x_D, \hat{y}_1, ..., \hat{y}_{j-1})$, siendo

D la dimensión del espacio de características. El proceso de clasificación empieza en el eslabón C_1 se propaga a través de la cadena: C_1 determina $P(y_1|x)$ y cada clasificador siguiente determina $P(y_j|x, y_1, ..., y_{j-1})$. El proceso de entrenamiento y clasificación se muestra en los algoritmos 1 y 2, respectivamente.

Algoritmo 1: Fase de entrenamiento de CC para un set de datos E y set de labels L. Fuente: Read et al., 2009.

Datos: $E = \{(x_1, y_1), ..., (x_M, y_M)\}$

para $j \in 1...k$ **hacer**
  $E' \leftarrow \{\}$;
  para $(x, y) \in E$ **hacer**
    $E' \leftarrow E' \cup ((x, y_1, ..., y_{j-1}), y_j)$;
  // Entrenar C_j para predecir mediante un algoritmo clasificador binario
  $C_j : E' \rightarrow y_j \in \{0, 1\}$;

Algoritmo 2: Fase de predicción de CC para una instancia de test x. Fuente: Read et al., 2009.

Datos: (x)

$Y \leftarrow \{\}$;
para $j \in 1...k$ **hacer**
  $Y \leftarrow Y \cup (l_j \leftarrow C_j : (x, l_1, ..., l_{j-1}))$;
devolver (x, Y) *(El ejemplo clasificado)*

Se utilizó el modelo *Support Vector Machine* (SVM) Evgeniou & Pontil (2001) para construir los clasificadores $C_1, ..., C_k$, utilizando la función de Kernel Radial Basis Function (RBF) en que $K(x, x') = \exp(-\gamma||x - x'||^2)$ y γ es un hiperparámetro definido por el modelador.

2.4.3. Random K Labelsets

Random *k-labelsets* es un método propuesto por Tsoumakas & Vlahavas (2007) de tipo Métodos de Ensamblaje. El clasificador *multi-label* se modela como un ensamblaje de clasificadores, y cada clasificador del ensamblaje considera un pequeño subset de etiquetas al azar para su entrenamiento. Estos son clasificadores *multi-clase* (donde solo se asocia una etiqueta a cada ejemplo) y se utiliza el *powerset* del subset correspondiente como el conjunto de etiquetas posibles que este puede asignar. De esta forma, el algoritmo toma en cuenta la correlación de etiquetas usando varios clasificadores de una clase, que son aplicados en tareas con un número de etiquetas pequeño y manejable.

Sea $L = \{\lambda_1, ..., \lambda_K\}$ el conjunto de todas las etiquetas, con $|L| = K$. Un *k-labelset* se define como un subconjunto del set de etiquetas totales con k elementos, es decir, $Y \subseteq L$ es un *k-labelset*, con $k = |Y|$. Se define L^k como el set que contiene todos los distintos *k-labelsets* en L, donde el tamaño de L^k está dado por $\binom{K}{k}$.

Label Powerset (LP) es un clasificador dentro de la categoría Transformación del Problema, en que se construye un solo clasificador *multi-clase*, donde las etiquetas se transforman en todas las combinaciones vistas en el set de entrenamiento, y en lugar de asignar varias etiquetas a un ejemplo, el clasificador asigna una etiqueta dentro de las 2^L combinaciones de etiquetas posibles. RAKEL construye iterativamente un ensamblaje de m clasificadores LP. En cada iteración, $i = 1, ..., m$, selecciona al azar

un *k-labelset* Y_i de L^k sin reemplazo. Luego se entrena un clasificador LP $h_i : X \rightarrow PowerSet(Y_i)$.

Algoritmo 3: Fase de producción del ensamblaje en RAKEL. Fuente: Tsoumakas & Vlahavas, 2007.

Datos: Número de nodos m, tamaño k de los *k-labelset*, set de etiquetas L, set de entrenamiento E

Resultado: Un ensamblaje de clasificadores h_i y sus correspondientes *k-labelsets* Y_i

```
R ← L^k ;
para i ∈ 1... min(m, |L^k|) hacer
    Y_i ← un k-labelset seleccionado al alzar de R;
    entrenar un clasificador LP h_i : X → PowerSet(Y_i) en D;
    R ← R\{Y_i};
```

El número de iteraciones m y la elección de k son parámetros definidos por el modelador, y $k \in \{2, 3, 4\}$ en general. La clasificación *multi-label* de una nueva instancia x se realiza de la siguiente manera. Cada modelo h_i toma una decisión binaria $h_i(x, \lambda_j)$ para cada etiqueta λ_j en el correspondiente *k-labelsets* Y_i. Luego, RAKEL calcula la decisión promedio para cada etiqueta λ_j en L y asigna la etiqueta correspondiente si el promedio supera a un *threshold* $t = 0.5$, es decir que la mitad o más de los clasificadores hayan decidido asignar la etiqueta λ_j al ejemplo correspondiente. De la misma manera que se hizo en CC, se utilizará el algoritmos SVM para construir los clasificadores binarios correspondientes.

2.5. Modelo de Propósito General BERT

En esta sección se muestra la base teórica utilizada para la segunda etapa de este trabajo que es clasificación *multi-label* de texto mediante Aprendizaje Profundo. En esta etapa se vectorizará el texto con el modelo lenguaje de propósito general BERT (*Bidirectional Encoder Representations from Transformers*) Devlin et al. (2018). La principal diferencia con FastText es que BERT logra generar vectores

contextualizados de palabras. Esto se refiere a que el vector de una palabra cambia de acuerdo al contexto en que esta se presenta y esto se logra mediante un cambio de paradigma en el Entendimiento del Lenguaje Natural, en que se pasa de tener representaciones vectoriales con relación 1 a 1 por cada *token* a generar vectores de palabras dinámicamente cada vez que se presentan en una frase distinta.

La metodología de clasificación de texto con BERT es distinta de la presentada anteriormente y se diferencia en que la vectorización y clasificación del texto se hacen en un mismo proceso. Se dice que BERT es un modelo de propósito general ya que se utiliza en diferentes tareas de PLN y para definir el uso que se le quiere dar solo se debe conectar una capa especializada en una tarea específica sobre la capa de *output* de BERT. Para clasificar texto, sobre la capa de *output* de BERT se añade el clasificador basado en redes neuronales y durante el entrenamiento se modifican los pesos de BERT y los del clasificador simultáneamente.

El entrenamiento de BERT consta de 2 fases, el preentrenamiento y la fase de ajuste de parámetros (*fine-tuning*). En el preentrenamiento el modelo es realizado de manera no supervisada utilizando un *corpus* grande y es extremadamente costoso en términos de tiempo y poder computacional. En comparación con el tiempo de entrenamiento de FastText, que demora minutos en un computador convencional, el preentrenamiento de BERT podría durar semanas y es por esto que se requiere *hardware* especializado para procesarlo. Dado que el preentrenamiento de BERT es costoso, el equipo de Google Research ha liberado varios modelos BERT preentrenados bajo distintas condiciones, con *corpus* de lenguajes diferentes y están disponibles para ser utilizados por la comunidad. En este trabajo se utilizó el modelo preentrenado BERT Base Multilingual Uncased implementado en la librería *transformers* Wolf et al. (2019) en Pytorch Paszke et al. (2019) por la compañía Hugging Face.

Por otro lado, el ajuste de parámetros (*fine-tuning*), donde el modelo es inicializado con los parámetros resultantes del preentrenamiento, tiene por objetivo ajustar los parámetros de BERT a la tarea que se quiere realizar y se hace de manera

supervisada. Este proceso es menos costoso en comparación con el preentrenamiento, sin embargo también requiere *hardware* especializado. Para utilizar BERT en la tarea de clasificación de texto se debe conectar un clasificador sobre la capa de *output*.

BERT se caracteriza por ser un modelo Aprendizaje Profundo, es decir, es una red neuronal de múltiples capaz, y sus buenos resultados en cuanto a la captura de propiedades intrínsecas del lenguaje se deben a la arquitectura de la red. La forma en que están conectadas las capas de la red no es común y cada microestructura dentro de la red tiene un objetivo específico. El modelo BERT se explica de manera conceptual ya que su alta complejidad dificulta detallarlo matemáticamente.

En esta sección primero se expondrá la forma de preprocesar el texto para convertirlo al *input* que será introducido al modelo. Luego se presenta el modelo *Transformer*, que es el modelo base que implementa BERT múltiples veces. En seguida se expone en qué consiste la fase de preentrenamiento de BERT y finalmente se muestra la forma de entrenarlo con el objetivo de clasificar texto.

2.5.1. Preprocesamiento de texto

Al igual que la *tokenización* que se realiza para entrenar FastText es necesario realizar un proceso de limpieza y normalización del texto. En el caso de BERT este proceso se realizó de la misma manera. Se eliminaron caracteres especiales que no corresponden al español y se corrigieron las faltas de ortografía con el mismo método descrito en 2.3.1.

BERT procesa frases en lugar de palabras y por esto el *input* asociado a un ejemplo de entrenamiento es llamado la "secuencia de *input*" y tiene forma matricial. Además, cada columna de la matriz de *input* que utiliza BERT se compone de la suma de 3 vectores. El primero es un vector de asociado a los *tokens* del vocabulario, el segundo es un vector de segmento y el tercero un vector de posicionamiento. En la figura 2.8 se muestra un esquema del *input* del modelo. A continuación se detalla la forma de

tokenizar el texto y luego en qué consisten los vectores asociados a *tokens*, los vectores de segmento y los vectores de posicionamiento.

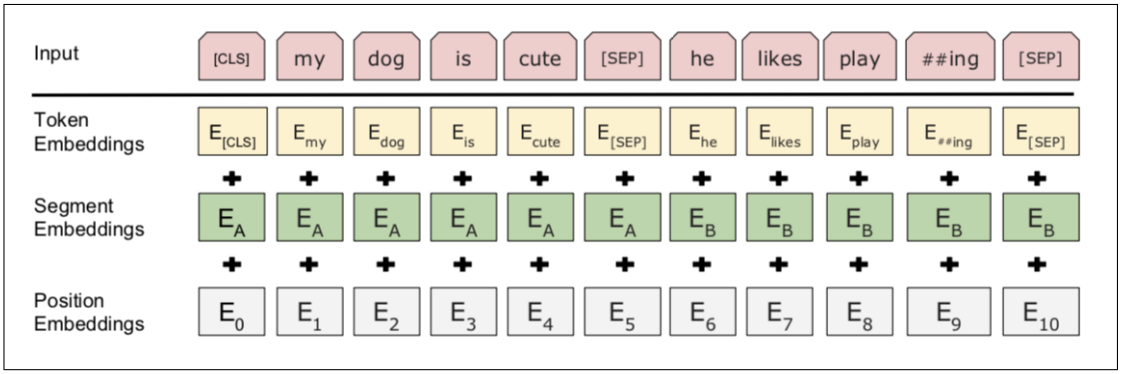


Figura 2.4. Representación de *input* de BERT. Fuente: Devlin et al. (2018)

La forma de *tokenizar* el texto cambia respecto de FastText. Los *tokens* utilizados por BERT corresponden a subpalabras del vocabulario y provienen de la *tokenización* WordPiece Wu et al. (2016). Este es un algoritmo avaro, que dado un número deseado de *tokens*, este los define a partir de caracteres individuales y va añadiendo *tokens* iterativamente combinando las unidades hasta llegar al número deseado, de tal manera que maximiza la verosimilitud del set de entrenamiento cuando se añade un nuevo *token*.

Dado que se utiliza un modelo preentrenado que viene con un set de *tokens* determinado, no es posible implementar este algoritmo sobre el *corpus* de la encuesta de satisfacción, y se utiliza el *tokenizador* implementado en la librería *transformers*. La *tokenización* de subpalabras añade un identificador especial antes de sufijos, de tal manera que cuando se divide una palabra en 2 o más *tokens* el modelo logra identificar que en conjunto se refieren a la palabra original. Ejemplos de la *tokenización* realizada se presentan en la tabla 2.2. En el trabajo original los autores determinan usar 30.000 *tokens*, que logra que el set de *tokens* sea menor que el tamaño vocabulario y también hace posible representar palabras que estén fuera del vocabulario original.

Tabla 2.2. Ejemplos de *tokenización* en subpalabras

Comentario preprocesado	*tokenización*
la atencion fue buena y el producto comprado es excelente	['la', 'atencion', 'fue', 'buena', 'y', 'el', 'producto', 'compra', '##do', 'es', 'excelente']
atencion rapida y expedita	['atencion', 'rapida', 'y', 'ex', '##ped', '##ita']

Como ya se mencionó, BERT procesa frases que corresponden a una secuencia de *tokens* y la secuencia de *input* siempre debe ser del mismo largo, independiente del largo de cada comentario. El largo determinado de la secuencia de *input* en este trabajo es 200. Se eligió 200 y no más por motivos de restricción de capacidad computacional. Para construir la secuencia de *input*, cada comentario se descompone en la secuencia de *tokens* de subpalabras correspondientes y si la esta tiene largo menor a 200, se añade un *token* especial [PAD] hasta completar 200. Si un comentario tiene más de 200 *tokens* asociados se trunca.

Además al inicio de cada secuencia de *input* es necesario incluir un *token* especial [CLS], cuya función es ser el vector de representación de la frase en tareas de clasificación. También cada secuencia de *tokens* provenientes de las subpalabras de un comentario se termina con el *token* especial [SEP]. En la tabla 2.3 se expone un ejemplo de un comentario de la encuesta transformado a una secuencia de *input* para introducir a BERT.

Tabla 2.3. Ejemplo de secuencia de *input* de BERT

Comentario preprocesado	Secuencia de *input*
la atencion fue buena y el producto comprado es excelente	[[CLS], 'la', 'atencion', 'fue', 'buena', 'y', 'el', 'producto', 'compra', '##do', 'es', 'excelente', [SEP], [PAD], [PAD], ...]

Cada *token* del vocabulario es indexado a un vector de dimensión del tamaño de las capas internas de BERT, que en este trabajo corresponde a 768 determinado por el modelo público BERT Base Multilingual Uncased que se usó. En la fase de preentrenamiento de BERT estos vectores son inicializados de manera aleatoria, y en la fase de ajuste de parámetros se usan los provenientes del modelo público preentrenado. Es decir, los vectores de *input* asociados a *tokens* son parámetros del modelo que se ajustan durante el entrenamiento.

El segundo vector del que se compone una columna de la matriz de *input* de BERT es el vector de segmento. Este vector es utilizado con un objetivo específico en la fase de preentrenamiento de BERT y puede tener dos valores, uno es el valor llamado *embedding* `A` y el otro valor es llamado *embedding* `B`. El objetivo de porqué se utiliza este vector se explicará en la sección correspondiente. En la fase de ajuste de parámetros, para clasificar texto se utiliza el *embedding* `A` en todas las posiciones de la secuencia de *input*.

El tercer vector es el de posicionamiento. La estructura interna del modelo utiliza un mecanismo de atención que, mediante los vectores de posicionamiento, permite que el modelo preste "atención" al contexto de cada palabra para determinar los vectores contextualizados de palabras. El mecanismo de atención se explicará en mayor detalle en la siguiente subsección. Existe un vector de posicionamiento asociado cada posición que puede tener un *token* en la secuencia de *input*, es decir que se utilizan 200 vectores de posicionamiento $\{P_0, P_1, P_2, ..., P_{199}\}$. Al igual que los vectores de *tokens* y de segmento, los vectores de posicionamiento se inicializan aleatoriamente en la fase de preentrenamiento, y en la fase de ajuste de parámetros se utilizan los vectores provenientes del modelo público preentrenado.

Como resumen, la primera columna de la matriz de *input* asociada a un comentario será igual a la suma del vector asociado al *token* [`CLS`], el vector `A` y el vector P_0. La segunda columna será la suma del vector asociado al primer *token* de la secuencia de un comentario, el vector `A` y el vector P_1, y continúa así hasta completar las 200 columnas de la matriz.

En la práctica, en el preentrenamiento y en el ajuste de parámetros se procesan varios ejemplos en paralelo, y el *input* corresponderá a un tensor de dimensiones $BS \times S \times D$, donde BS corresponde al tamaño de lote, S corresponde al largo de la secuencia de *input* y D el tamaño de las capas internas del modelo. Los autores realizan el preentrenamiento de BERT con secuencias de *input* de largo 512 y tamaño de lote 256, y la fase de ajuste de parámetros la realizan con un tamaño de lote de hasta 32.

El largo de la secuencia de *input* de este trabajo se determinó en 200 debido a restricciones de procesamiento computacional que tienen que ver con el tope del uso de memoria utilizado por el dispositivo para entrenar la red, que en este trabajo fue una GPU Nvidia Geforce GTX Titan X de 12GB GDDR5 y arquitectura 384 Bit. Se determinó largo de secuencia 200 debido a que los comentarios de mayor largo no representan más del 2% del total de comentarios del set de entrenamiento, y esto permite tener un tamaño máximo 24 comentarios por lote.

Una vez procesados los comentarios del set de entrenamiento en listas de *tokens*, el *input* del modelo en cada iteración del entrenamiento corresponderá a un tensor de dimensiones $BS \times 200 \times 768$, donde BS corresponde al tamaño de lote, 200 corresponde al largo de la secuencia de *input*, y 768 el tamaño de las capas internas del modelo.

2.5.2. Transformer

La implementación de BERT se basa en múltiples capas del codificador del *Transformer* Vaswani et al. (2017), un modelo de aprendizaje profundo utilizado para realizar la tarea de PLN de Traducción de Máquinas, y que revolucionó la forma en que se traduce texto debido al mecanismo de atención que emplea. A continuación se describe conceptualmente el objetivo principal de la arquitectura del codificador del *Transformer*, luego se presenta un ejemplo práctico de cómo el *Transformer* logra implementar el mecanismo de atención y posteriormente se presentan los elementos de la red que hacen posible implementar la atención con el objetivo de explicar la lógica detrás de BERT.

El *Transformer* tiene por objetivo traducir una secuencia de Tipo A a una de Tipo B, que usualmente es traducir un idioma a otro. Los modelos de Traducción de Máquina usan una arquitectura de codificador-decodificador, generalmente basada en redes neuronales recurrentes, que se caracterizan por alimentar al modelo con una secuencia de *input*, uno a la vez. El *Transformer* es revolucionario porque elimina toda la estructura recurrente, basándose solo en un mecanismo de autoatención (*self-attention*) que permite procesar todo el *input* en paralelo. En esta sección se pretende explicar conceptualmente en qué consiste el mecanismo de autoatención.

En una arquitectura codificador-decodificador el codificador tiene por objetivo mapear una secuencia de *input* de vectores $(x_1, ..., x_n)$ a una secuencia de representaciones continuas $z = (z_1, ..., z_n)$ que serán consumidas por el decodificador de manera que pueda generar un *output* $(y_1, ..., y_n)$. El codificador del *Transformer* almacena de manera abstracta la información respecto del significado y posicionamiento de las palabras en múltiples proyecciones de un espacio vectorial, que luego será consumida por el decodificador. Esto es análogo a como FastText condensa información perteneciente a un espacio de dimensión del tamaño del vocabulario V en un espacio de tamaño reducido D y que posteriormente es consumida por un modelo clasificador. En el caso de BERT, este implementa múltiples veces la codificación del *Transformer* almacenando de manera abstracta gran cantidad de información respecto de componentes intrínsecos del lenguaje, y luego mediante una capa de *output* especial puede ser utilizada para una tarea específica.

En el *Transformer*, la codificación de la información se hace mediante un mecanismo de atención que construye 3 vectores para poder ejecutarse. Estos son un vector de consulta q, un vector de clave (*key*) k y un vector de valor v, cada uno, de manera abstracta, tiene un objetivo específico. La idea fundamental detrás del mecanismo es que este pondera la atención que se debe asignar a un elemento de la secuencia utilizando el vector de consulta y de clave y luego a través del vector de valor se extrae la información requerida.

Dada la complejidad de la estructura de la red y operaciones matriciales del modelo, primero se mostrará un esquema simplificado del mecanismo de atención que se emplea a través de un ejemplo y luego se procederá a detallar conceptualmente cómo se compone el codificador. Por ejemplo, se quiere traducir la secuencia en inglés "*The dog is barking*" a español "El perro está ladrando". Cada elemento de la secuencia de *input* del modelo es un *embedding* de palabra, por ejemplo podría ser un vector de palabra generado por FastText.

En el proceso de codificación, la secuencia de *input* se procesa en paralelo, y a cada vector de palabra se le suma un vector de posicionamiento de acuerdo a su posición en la oración para así lograr identificar el orden de las palabras. Por ende en este ejemplo el *input* del modelo corresponderá a los vectores $x_1 = (x'_1 + p_1), ..., x_4 = (x'_4 + p_4)$ que representan los vectores de palabras de los *tokens* "*The*", "*dog*", "*is*" y "*barking*" sumados al vector de posicionamiento. En este esquema cada *input* se introduce como un vector fila, entonces el *input* del modelo corresponde a una matriz de $N \times D$ donde N corresponde al largo de la secuencia y D a la dimensión del vector de palabra.

Luego se implementa el mecanismo de atención que consiste en que a partir de cada palabra de *input* se crea un vector de consulta, uno de clave y uno de valor (*query, key, value*). Es decir el modelo contiene 3 matrices de parámetros W^Q, W^K y W^V que multiplicadas con los vectores de *input* resultan en q_i, k_i y v_i, para $i \in \{1, 2, 3, 4\}$. Para cada palabra *input*, el modelo desea asignarle un puntaje de atención respecto de todas las otras palabras de la secuencia, incluida ella misma.

El modelo multiplicará el vector de consulta y de clave de cada token de *input* procesado, y usará la función *softmax* sobre ellos de tal manera que le asigna un peso a la atención puesta a cada palabra, y la suma de todos los pesos será igual a 1. Por ejemplo, la atención que le presta la palabra "*barking*" a la palabra "*dog*" está determinada por la multiplicación de $q_4 \cdot k_2$, y es ponderada respecto de la atención puesta a las palabras restantes "*The*", "*is*" y "*barking*" ejecutando la función *softmax* sobre $q_4 \cdot k_i$ para $i \in 1, ..., 4$.

Luego para extraer información se pondera este resultado de la función *softmax* por el vector de valor de cada palabra y de esta manera crea una codificación interna $z_i, i \in 1, ..., 4$. Es decir, la codificación interna de la palabra "*barking*" z_4 estará dada por $\alpha_1 v_1 + ... + \alpha_4 v_4$, donde $\alpha_i \in \mathbb{R}$ corresponde a $\frac{\exp(q_4 \cdot k_i)}{\sum_j \exp(q_4 \cdot k_j)}$ y es la atención ponderada respecto de todas las palabras de la secuencia multiplicada por los vectores de valor de cada una. En forma matricial siendo $Q = [q_1 \; ... \; q_4]$, y análogamente para K y V, se realiza $softmax(QK^T)V = Z$, donde cada fila corresponde a la codificación interna que se definió antes como z_i.

Una vez que toda la secuencia de *input* fue consumida simultáneamente por el codificador, se empieza a generar la secuencia de *output*, que corresponde a la traducción. Sin entrar en mayores detalles, ya que el decodificador no es relevante para BERT, en el paso de la traducción el vector de consulta está asociado al texto en español y el vector clave y vector valor están asociados al texto en inglés. De esta manera se logra determinar en qué palabras del inglés se tiene que fijar el modelo para generar la traducción de la siguiente palabra en español. En la figura 2.5 se aprecia un esquema del modelo.

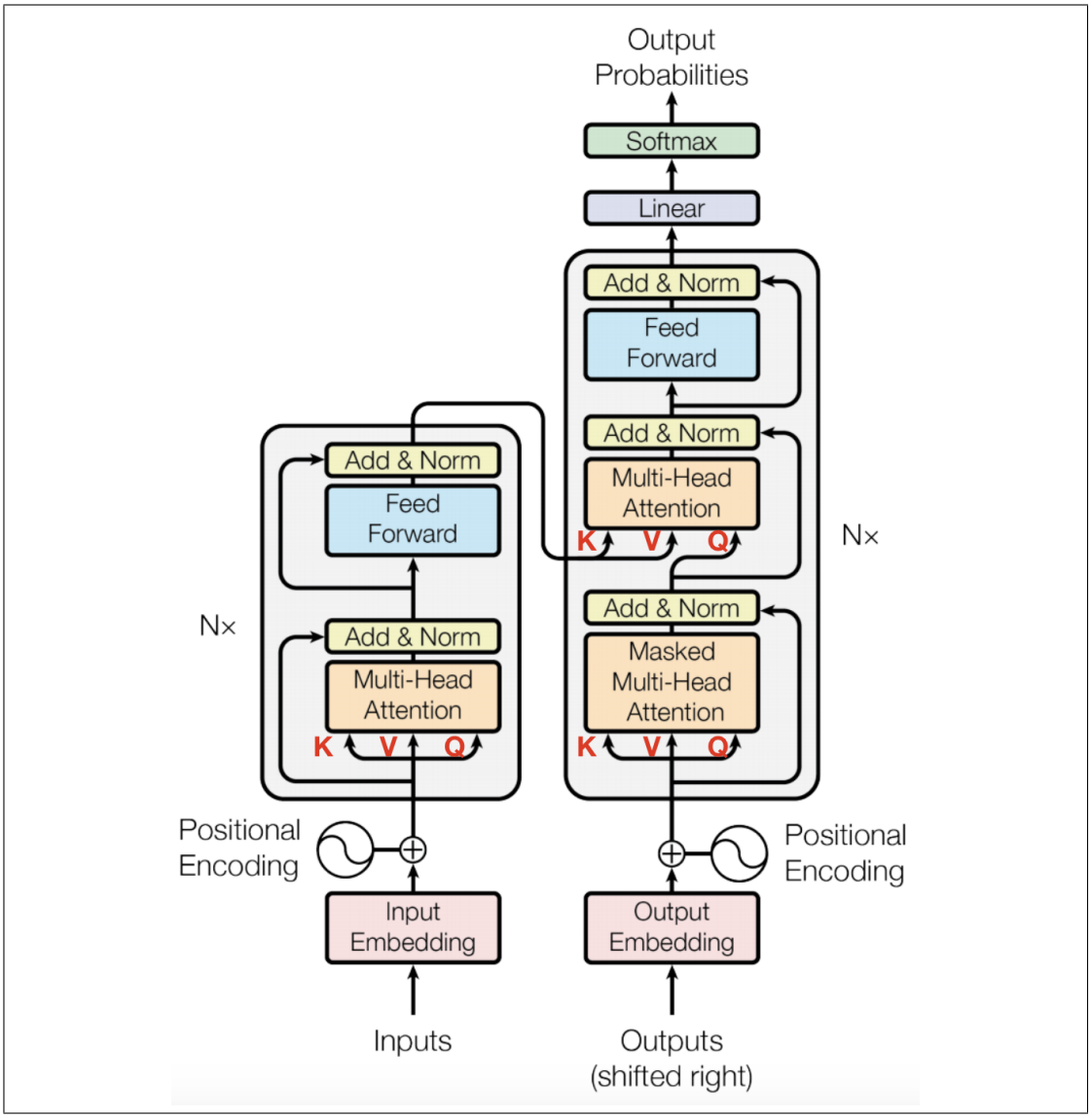


Figura 2.5. Arquitectura del modelo Transformer . Fuente: Vaswani et al. (2017)

BERT solo utiliza el codificador del *Transformer*. Este se compone en una red neuronal de 2 subcapas. La primera es un mecanismo de autoatención y la segunda es una *position-wise fully-connected feed-forward network*. La subcapa de autoatención implementa dos mecanismos atención, la primera es la atención particular llamada *Scaled Dot-Product Attention*. El modelo *Transformer* está altamente optimizado mediante operaciones de álgebra lineal que mitigan el problema de explosión o de

desvanecimiento del gradiente en el proceso de *back-propagation*, que se mencionó en la sección 2.2.

Scaled Dot-Product Attention es el mecanismo que se ejemplificó. Esta atención consiste en transformar cada vector de *input* en 3 vectores mediante la multiplicación de 3 matrices distintas que conforman los parámetros de la red. De manera abstracta, la concatenación de los vectores en matrices determinan una matriz de consulta Q, una matriz de claves (*keys*) K y una matriz de valores V. El mecanismo de atención consiste en multiplicar el vector de consulta q_i de cada palabra con el vector de clave k_i de las demás palabras y utilizar la función *softmax* para determinar la relevancia que cada una tiene respecto de las otras la oración. Luego el resultado se multiplica por la matriz de valores V determinando la codificación de cada palabra en cada columna de la matriz resultante Z'. La fórmula del mecanismo de atención es la siguiente

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V = Z' \tag{2.12}$$

donde d_k corresponde a la dimensión del vector de clave, y la división de la matriz por $\sqrt{d_k}$ tiene por objetivo aumentar la convergencia del gradiente en el proceso de *back propagation* cuando se ajustan los pesos de la red.

Sobre el *output* de la subcapa de autoatención se implementa una capa *feed-forward*, en que se procesan los vectores z'_i correspondientes a las filas de Z' en paralelo y de forma independiente, resultando la codificación en una matriz Z. En la figura 2.6 se presenta un esquema del procesamiento del *input* paralelizado. El *input* siempre es una secuencia de largo fijo al igual que en BERT.

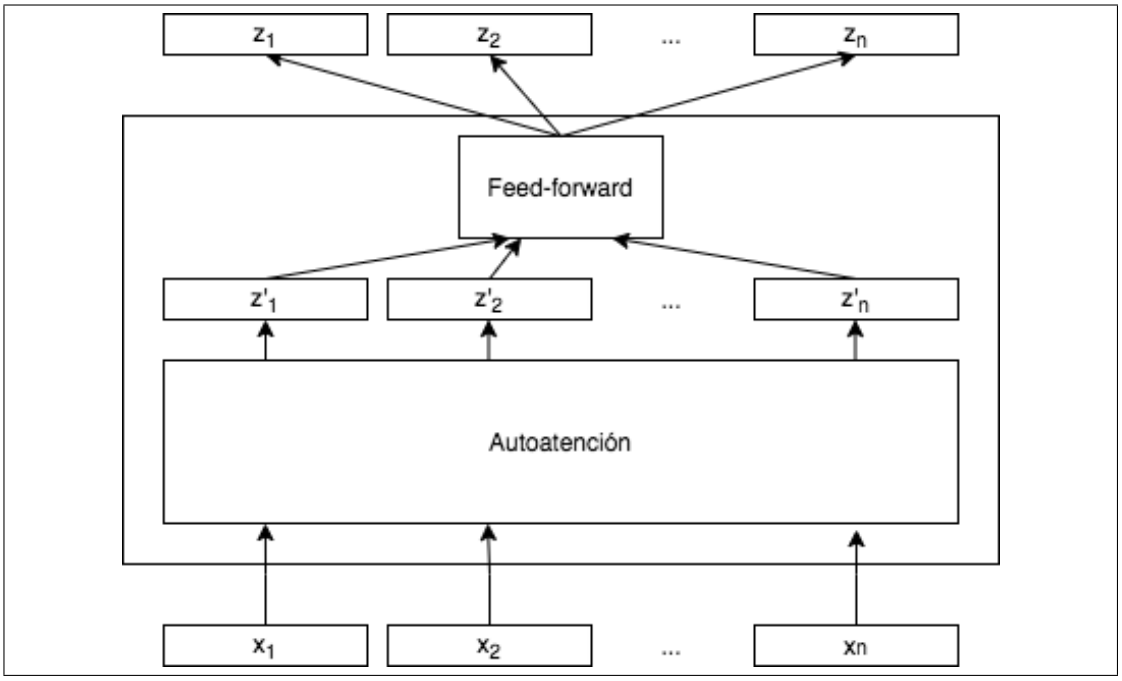


Figura 2.6. Paralelización del Procesamiento en el Transformer. La red *feed-forward* es la misma. Fuente: elaboración propia

En el esquema de la figura 2.5 se observa que se emplea una conexión residual He et al. (2015) entre la dos subcapas, esto quiere decir que se suma el *input* al vector de salida del cada subcapa del codificador. Además, se utiliza una capa de normalización Lei Ba et al. (2016), con el objetivo que el proceso de entrenamiento y optimización sean más rápido debido a que aumenta la convergencia del gradiente. Así el *output* de cada subcapa es $LayerNorm(x + Sublayer(x))$, donde $Sublayer(x)$ es la función implementada por la misma subcapa.

En la misma figura 2.5, se observa que se suma un vector de posicionamiento al *input* antes de introducirlo al modelo. Este consiste en una función trigonométrica con distintos valores dependiendo de la posición del *token* que se está ingresando.

El segundo tipo de atención es la Atención Multicabezal (*Multi-Head Attention*), que consiste en emplear múltiples procesos de atención a la vez, donde se proyectan las consultas, claves y valores h veces en diferentes proyecciones lineales. Se realiza

la atención particular h veces en paralelo. Es decir, los parámetros de un codificador *Transformer* incluyen las matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, con las que se construyen las matrices matrices $(Q^i, K^i, V^i), i \in \{1, ..., h\}$ y que luego de implementar la atención particular resultan las matrices $Z'_1, ..., Z'_h$. Aquí d_{model} corresponde a la dimensión de las capas internas del modelo, d_k y d_v son la dimensión de los vectores clave y valor respectivamente. Finalmente, se obtiene la codificación final a partir de la concatenación de $Z'_1, ..., Z'_h$ multiplicada por la matriz de parámetros $W^O \in \mathbb{R}^{hd_v \times d_{model}}$. En la figura 2.7 se muestra un esquema de los dos tipos de atenciones descritos. El cálculo de la atención mutlicabezal es de la siguiente forma

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{2.13}$$

donde $head_i$ corresponde a $Attention(QW_i^Q, KW_i^K, VW_i^V)$.

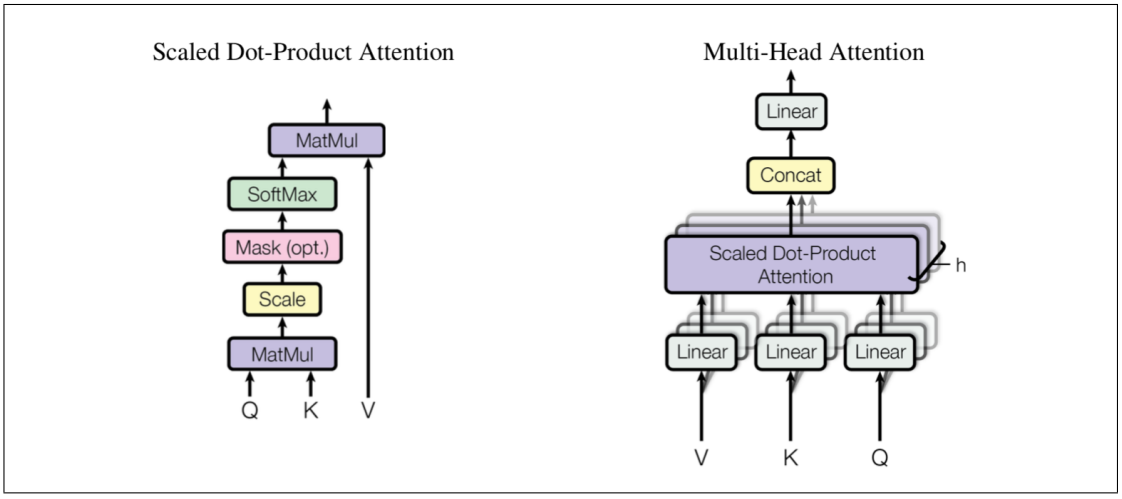


Figura 2.7. (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. Fuente: Vaswani et al. (2017)

El *Transformer* fue revolucionario con respecto a previos modelos debido a que permite procesar el *input* de naturaleza secuencial en paralelo, ya que con modelos de arquitectura recurrente se requería procesar un *token* de *input* a la vez. El procesamiento en paralelo disminuye el tiempo requerido de entrenamiento, y permite

codificar el contexto de las palabras de mejor manera. Además, el primer mecanismo de atención presentado permite al modelo enfocarse en más de una posición a la vez y el segundo permite proyectar las representaciones de palabras en múltiples espacios, donde múltiples tuplas de matrices (Q, K, V) expanden exponencialmente las posibilidades del espacio de características en que se puede fijar el modelo para realizar la traducción, o en caso de BERT para caracterizar los componentes intrínsecos del lenguaje.

2.5.3. Preentrenamieto de BERT

Una vez descrito en qué consiste el *Transformer* se procede a detallar cómo se realiza el preentrenamiento de BERT Devlin et al. (2018). Este modelo implementa múltiples capas del codificador del *Transformer*, que tienen por objetivo codificar de manera abstracta la información respecto del significado y posicionamiento de las palabras del lenguaje en múltiples proyecciones de un espacio vectorial. Dado que el preentrenamieto de BERT es extremadamente costoso en términos de capacidad computacional, los autores han liberado varios modelos preentrenados en un *corpus* muy grande de textos y son públicos para ser empleados en una tarea particular de PLN.

En primera instancia se liberaron 2 modelos BERT, $BERT_{BASE}$ y $BERT_{LARGE}$. En este trabajo se utiliza $BERT_{BASE}$, que consta de 110 millones de parámetros. La red se compone de $L = 12$ capas, o bloques de *Transformers*, el tamaño de los estados ocultos es $H = 768$ y cada *Transformer* tiene $A = 12$ cabezales de atención. Hasta el momento de iniciar esta tesis $BERT_{LARGE}$ que tiene 340 millones de parámetros era el modelo número 1 en el *benchamark* GLUE, sin embargo no se utilizó en este trabajo debido a restricciones de poder computacional.

Como ya se mencionó BERT es un modelo de lenguaje de propósito general cuya implementación se divide en dos fases: el preentrenamiento y el ajuste de parámetros (*fine-tuning*). Durante el preentrenamiento el modelo es entrenado de manera no supervisada utilizando *corpus* de más de 3 mil millones de palabras. Aquí se realiza

la tarea de Modelo de Enmascaramiento de Lenguaje (*Masked Language Model*) (MLM) y Predicción de la Siguiente Frase (*Next Sentence Prediction*) (NSP). Primero se detallarán los componentes del *input* que recibe BERT, y luego se mostrará en qué consisten los dos objetivos del preentrenamiento de BERT.

BERT recibe oraciones como ejemplos de entrenamiento. Una oración está asociada a una secuencia de *input*, que es representada por una matriz de *input* y cada vector columna se compone de la suma de 3 vectores, los que son el vector asociado a un *token*, el vector de segmento y el vector de posicionamiento. Cuando se entrena BERT por primera vez, todos los vectores de *input* son inicializados al azar y conforman parámetros del modelo. El largo de la secuencia de *input* utilizada en el preentrenamiento de $BERT_{BASE}$ es de 512 y la dimensión de las capas internas del modelo es 768, y por ende la dimensión de los vectores de palabras resultantes también. Por lo tanto cada ejemplo de *input* en el preentrenamiento corresponderá a una matriz de 512×768.

El vocabulario se construye a través de la *tokenización* WordPiece, que se detalló en la sección 2.5.1, y cada *token* asociado a una subpalabra tiene asociado un vector. Además existen 4 *tokens* especiales que cumplen un objetivo específico y que tienen asociados un vector. Estos son `[CLS]`, `[SEP]`, `[PAD]` y `[MASK]`. `[CLS]` tiene por objetivo determinar el vector de "clase", que generalmente es usado como vector de representación de una oración. En el preentrenamiento y en tareas relacionadas con preguntas y respuestas `[SEP]` es usado para separar dos frases distintas que son introducidas como *input* en BERT y en otras tareas es usado para marcar el término de una oración. `[PAD]` es utilizado para rellenar las secuencias que tienen largo menor a 512 y `[MASK]` tiene por objetivo ocultar una palabra el la tarea de preentrenamiento MLM y cuando se realiza el ajuste de parámetros de BERT este no se utiliza.

Existen dos vectores de segmento, estos corresponden al vector `A` y el vector `B`. En el preentrenamiento de BERT estos están relacionados con la tarea NSP y en el ajuste de parámetros de BERT se utilizan cuando se realizan tareas de PLN relacionadas con preguntas y respuestas. Para la tarea de clasificación de texto solo se usa el vector `A`.

Por último existen 512 vectores de posicionamiento y cada uno está relacionado a una posición. A diferencia del modelo *Transformer* que usa una función trigonométrica para diferenciar las posiciones de los *tokens*, BERT entrena estos vectores desde cero y constituyen parámetros del modelo. En la sección 2.5.1 en la figura 2.8 se presenta un esquema del *input* de BERT.

A continuación se procederá a explicar cómo las tareas de MLM y NSP logran generar representaciones de vectores contextualizados. Dada la complejidad del modelo, se detallará conceptualmente en qué consiste cada proceso.

El objetivo de la tarea de MLM es que el modelo aprenda a predecir palabras de acuerdo al contexto en que aparecen. Para esto BERT oculta (enmascara) el 15% de los *tokens* de una secuencia *input* al azar, reemplazándolos por el *token* especial [`MASK`]. Por ejemplo, la secuencia [*my dog is hairy*] se transforma en [*my dog is* [`MASK`]], y de esta manera se evita que el modelo pueda "ver" la palabra que está tratando de predecir.

En la práctica, los autores determinan que del 15% de *tokens* a reemplazar, el 80% de las veces se reemplaza por el *token* especial [`MASK`], el 10% de las veces por el mismo *token*, y el otro 10% por un *token* al azar. La característica fundamental que eleva el rendimiento del modelo es que se procesa el contexto de la izquierda y derecha de cada *token* simultáneamente, y no de forma independiente, a diferencia de previos modelos de propósito general.

En el preentrenamiento se usa la función de pérdida *Cross Entropy Loss* para comparar *output* generado con una codificación *one-hot* asociada a los *tokens* del vocabulario. Luego, la actualización de los pesos a través de *back-propagation* se realiza solo actualizando los pesos que están conectados con la predicción de las palabras enmascaradas. De esta manera, el modelo ajusta los pesos de la red haciendo que el modelo sea capaz de predecir 1 o más palabras de acuerdo a su contexto.

La segunda tarea es NSP, que apunta a entrenar un modelo que sea capaz de entender las relaciones entre oraciones, que resulta especialmente útil para realizar la tarea de PLN de preguntas y respuestas. Esta tarea se realiza en conjunto con MLM y

para realizarla se introducen 2 frases como ejemplo de entrenamiento que no necesariamente son oraciones del lenguaje sino que puede ser una separación arbitraria de una. El 50% de las veces la segunda frase corresponde a la continuación de la primera, y el 50% de las veces no. En la secuencia de *input* de largo 512 se usa el *token* `[SEP]` entre cada frase y el vector de segmento asociado a cada *token* de la primera oración corresponderá al vector `A`, y si la segunda oración es la continuación de la primera, también corresponderán al vector `A`. En caso de que la segunda oración no sea a la continuación de la primera, los vectores de segmento asociado a los *tokens* de la segunda oración serán el vector `B`.

En el preentrenamiento, una vez que la secuencia de *input* es procesada, el *output* de BERT corresponde a una secuencia de vectores C, $T_1, ..., T_N$, $T_{[SEP]}$ y $T'_1, ..., T'_M$ asociados al *token* `[CLS]`, los *tokens* de la primera oración, el *token* `[SEP]` y *tokens* de la segunda oración, respectivamente. En la figura 2.8 se muestra un esquema de la fase de preentrenamiento de BERT.

Para realizar la tarea de NSP los autores utilizan el vector *output* que correspondiente a la posición del *token* de *input* `[CLS]` como valor binario para determinar si la segunda oración era la continuación de la primera o no. El objetivo es que el modelo logre comprender la relación entre dos frases y empíricamente se mostró que su aplicación mejora los resultados obtenidos en el *benchmark* GLUE.

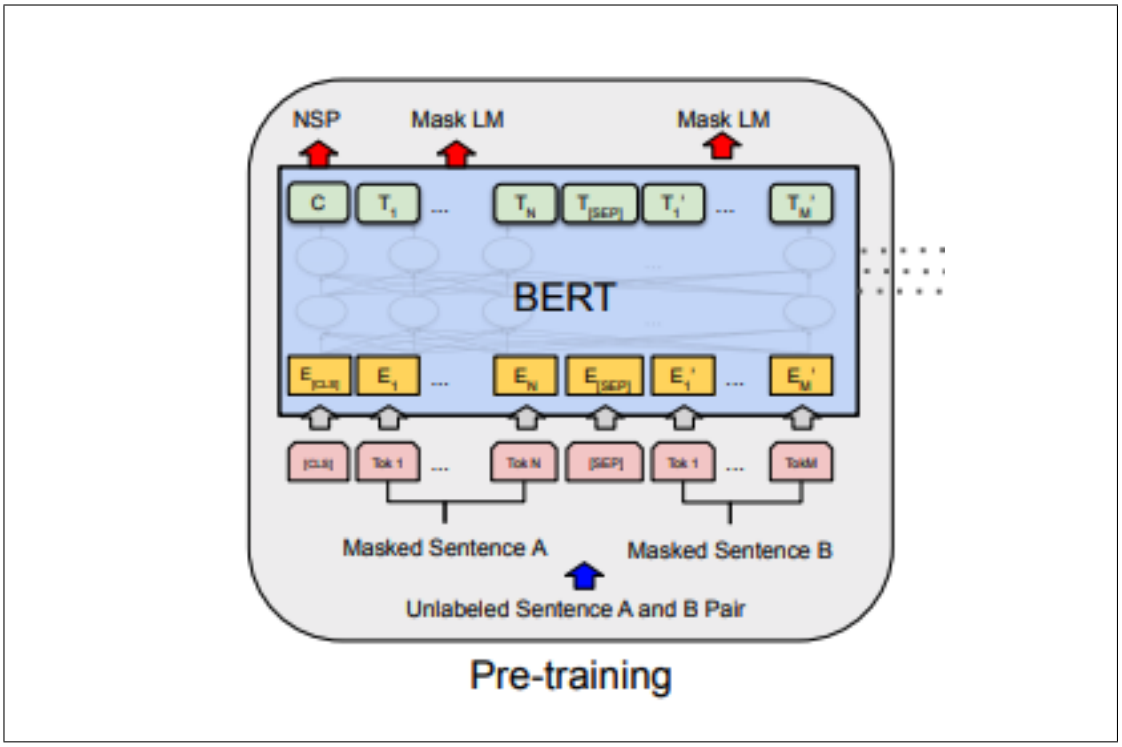


Figura 2.8. Preentrenamiento de BERT. Fuente: Devlin et al. (2018)

2.5.4. Ajuste de Parámetros y Clasificación de Texto con BERT

Una vez completado el preentrenamiento de BERT se deben ajustar los parámetros del modelo para realizar una tarea específica. El ajuste de parámetros mejora de gran manera el rendimiento del modelo medido en las métricas de evaluación de la tarea que se esta realizando.

Para realizar clasificación de texto se debe conectar un clasificador basado en redes neuronales sobre la capa de *output* de BERT. Luego se entrena el modelo con ejemplos de comentarios bajo el concepto de aprendizaje supervisado, con una función objetivo que logre actualizar los pesos de BERT de acuerdo a la tarea que se requiere realizar mediante el proceso de *back-propagation*, y con una tasa de aprendizaje pequeña. La tasa de aprendizaje se refiere al escalar que determina la magnitud del paso que se da en la dirección del gradiente cuando se actualizan los pesos de la red en el proceso de aprendizaje.

En este trabajo el clasificador *multi-label* conectado sobre BERT será el clasificador MLP descrito en 2.4.1, con función de pérdida *Focal Loss*. Esto significa que el clasificador recibe los vectores de *output* de color verde del modelo BERT ilustrados en la figura 2.8. El vector o vectores que se traspasan al clasificador desde BERT corresponden a la representación vectorial del comentario y en lugar de representar cada comentario como el promedio de todos los vectores de palabras $T_1, ..., T_n$, los autores recomiendan usar el vector C obtenido como *output* del modelo para representar el comentario como se ilustra en la figura 2.9.

En la práctica, una vez *tokenizados* los ejemplos del set de entrenamiento, como se mostró en 2.5.1, los autores recomiendan introducirlos al modelos por lotes de 16 o 32 ejemplos a la vez, y con una tasa de aprendizaje de $5 \cdot 10^{-5}, 3 \cdot 10^{-5}$ o $2 \cdot 10^{-5}$ para el algoritmo optimización de Adam Loshchilov & Hutter (2017), que actualiza los pesos de la red en el proceso de *back-propagation*. En términos de tiempo computacional el ajuste de parámetros es del orden 10^2 veces menos costoso que el preentrenamiento.

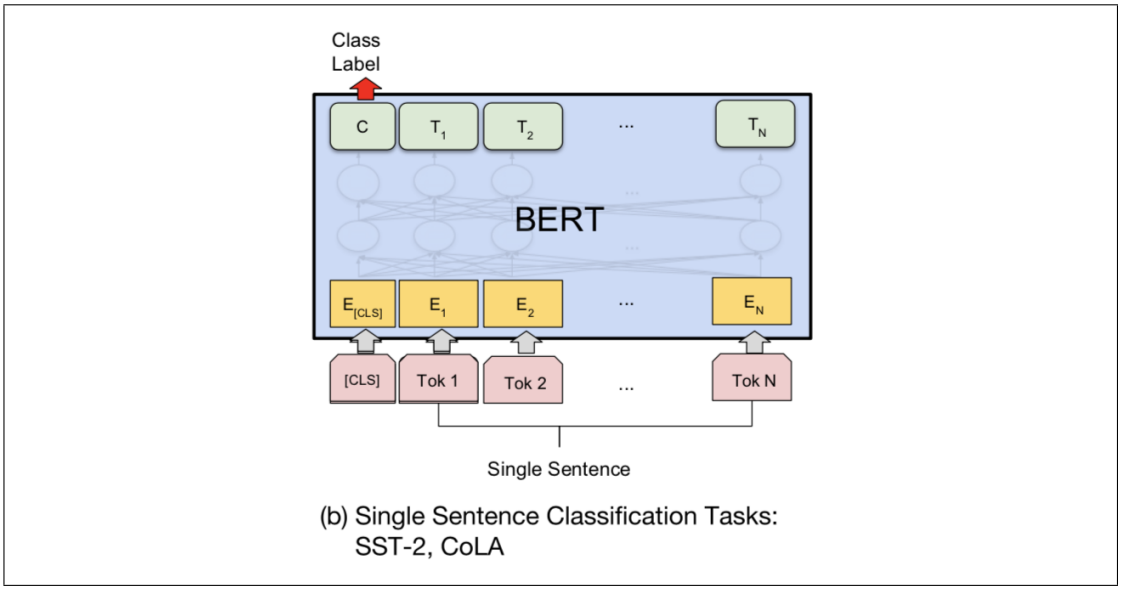


Figura 2.9. Tarea de clasificación de una sola secuencia. Fuente: Devlin et al. (2018)

2.6. Resumen Extractivo de Texto

Resumir comentarios de la encuesta de satisfacción corresponde a la tercera parte de este trabajo de tesis. Luego de entrenar un modelo de clasificación de texto se procede a implementar el resumen automático de texto sobre conjuntos de comentarios agrupados por periodo de tiempo y local en que se realizó la compra a la que está asociado, con el fin de obtener la información más relevante en un texto condensado. Un resumen puede ser definido como un texto que contiene información importante de varios textos, y que es de menor extensión que los textos originales del conjunto a resumir.

En el campo de investigación de PLN, los tipos de resumen se pueden categorizar de dos maneras de acuerdo a la forma en que el *output* es generado. El primer tipo es *abstractive text summarization*, que consiste en generación de texto a partir texto que compone el documento o documentos que se entregan, y el resultado se caracteriza por asemejarse a una redacción humana. Por otro lado *extractive text summarization* es un proceso en que el sistema copia o extrae partes del documento, o documentos, textualmente y genera un resumen no necesariamente cohesionado, sino que a partir de las frases más representativas.

En este trabajo se aplicará el segundo tipo de resumen ya que los resúmenes abstractos requieren mucho más trabajo respecto a la comprensión del lenguaje para lograr la generación de texto, lo que desviaría el enfoque del estudio, que es obtener los hechos puntuales que los clientes consideraron para calificar su experiencia de compra. Los resúmenes se harán respecto de los comentarios agrupados por un determinado periodo de tiempo y local, por lo que el problema se transforma en resumir una cantidad acotada de comentarios. Los conjuntos de comentarios agrupados por periodo de tiempo y local tienen un largo promedio de $45,2$ comentarios, desviación estándar de $29,8$, y máximo de 451.

2.6.1. Resumen Extractivo de Texto Basado en K-Means

De acuerdo a la revisión de métodos de resumen extractivo de texto de Rajasekaran & Varalakshmi (2018), un resumen se genera mediante un proceso que primero asigna un puntaje a diferentes oraciones en el documento a resumir y luego se seleccionan las oraciones que compondrán el resumen de acuerdo a un criterio basado en el puntaje asignado. Los métodos utilizados para asignar puntaje pueden basarse en métodos estadísticos, heurísticas, métodos basados en grafos o *clusterización*, entre otros. En este trabajo se implementarán diferentes variaciones de un método basado en *clusters*. El algoritmo agrupa representaciones vectoriales de los textos en torno a múltiples centroides, basándose en las características de su representación vectorial y luego se seleccionan las frases que compondrán el resumen mediante la asignación de puntaje de acuerdo a la cercanía a cada centroide.

La metodología de resumen de texto en este trabajo de tesis se basará en los trabajos "*Towards automatic extractive text summarization of A-133 Single Audit reports with machine learning*" Chou et al. (2019) y "*Leveraging BERT for Extractive Text Summarization on Lectures*" D. Miller (2019). Ambos utilizan el algoritmo de *clusterización* K-means para agrupar las representaciones vectoriales de las frases del texto en *clusters* y seleccionar las oraciones que compondrán el resumen de acuerdo a la distancia de su representación vectorial respecto del centroide de cada uno.

El algoritmo de formación de *clusters* K-means se formula de la siguiente manera. Dado un número de observaciones N, el algoritmo K-means construye una partición de las N observaciones separándolas en k *clusters*, k determinado por el usuario, de tal manera que minimiza la suma total de los cuadrados de la distancia de los vectores dentro de cada *cluster* con respecto a su centroide. Este se entrena de manera no supervisada, por lo que el modelador solo debe elegir el hiperparámetro k del modelo. El algoritmo primero selecciona k centroides, $a_1, ..., a_k$ al azar dentro del espacio vectorial, y en cada iteración, cada observación X es asignada al *cluster* $C_j, j \in \{1, ..., k\}$ de acuerdo al centroide más cercano y si existe una observación equidistante entre 2 o más centroides la observación se asigna al azar entre estos,

como se expone en la ecuación 2.14.

$$X \in C_j \text{ si } \|X - a_j\|^2 \leq \left\|X - a_i\right\|^2 \ , \ 1 \leq i \leq k \tag{2.14}$$

Luego se recalcula el centroide de cada *cluster* como la media de todas las observaciones asignadas a cada uno. En seguida se vuelven a reasignar las observaciones a cada *cluster* de acuerdo a la cercanía al centroide correspondiente. Este proceso se repite hasta que la nueva asignación de observaciones no cambia respecto de la iteración previa.

No existe un método que determine una mejor elección de k y en este trabajo se utilizará el puntaje de la silueta (*silhouette score*). Este método consiste en calcular la distancia media a entre una observación y todas las observaciones del *cluster* y la distancia media b de la observación con todas las observaciones del *cluster* más cercano, determinando el coeficiente de silueta s para cada observación como

$$s = \frac{b - a}{\max(a, b)} \tag{2.15}$$

El coeficiente de silueta varía entre -1 y 1 y el k elegido es tal que el promedio del coeficiente de silueta de todas las observaciones sea mayor.

El trabajo de Chou et al. (2019) detalla la estrategia utilizada para resumir auditorías realizadas a instituciones federales de Estados Unidos y esta requiere de varios pasos manuales y heurísticas en el proceso para generar los resúmenes. El objetivo es resumir cada auditoría extrayendo un número determinado de oraciones, que en conjunto tengan la información más relevante del texto y sean menos que la cantidad de oraciones del texto original. El texto de una auditoría, que se compone de varios párrafos, se preprocesa identificando cada una de las oraciones que lo componen y luego se generan representaciones vectoriales de cada una. En total el set de datos se compone de aproximadamente 19 mil auditorías, las que se dividieron en 452.071 oraciones vectorizadas.

En seguida proceden a agrupar todas las oraciones de todos los textos en *clusters* de acuerdo al algoritmo K-means. El hiperparámetro k es elegido mediante el método del codo (*elbow method*), que consiste en graficar la varianza *intracluster* para distintos valores de k y elegir visualmente el valor de k donde se forme un "codo" en el gráfico. Luego, los autores revisan las oraciones de los *clusters* formados para identificar los tópicos de cada uno y la uniformidad del contenido. Ellos describen que a pesar que la información de cada *cluster* no era uniforme, pudieron distinguir *clusters* con contenido que no informativo, y las oraciones asociados a estos fueron descartados para ser incluidos en el resumen.

Los autores definen la cantidad de frases que compondrán un resumen a través de una heurística. Esta consiste en una función discreta que de acuerdo a la cantidad de oraciones originales en la auditoría a resumir, el resumen tendrá una cantidad de oraciones que varía desde aproximadamente el 50% de la cantidad original, cuando el texto es de 15 frases o menos y aproximadamente el 5% para textos de más de 250 frases.

Finalmente, definen 2 estrategias para elegir las frases que irán en los resúmenes de cada auditoría y ambas tienen en común que consisten en ordenar las oraciones de acuerdo a un puntaje determinado por la distancia coseno entre las observaciones y los centroides de los *clusters*. La primera es ordenar las oraciones de acuerdo a la menor distancia a cualquiera de los centroides y seleccionar las oraciones del resumen respecto del orden de aparición en esta lista ordenada. La segunda es ordenar las oraciones de acuerdo a la distancia de su centroide asociado y seleccionar 1 oración por cada *cluster* formado hasta que se complete el número deseado de oraciones que tendrá el resumen. Los autores determinan que la segunda estrategia es es levemente mejor que la primera.

Ahora se describe la metodología presentada en D. Miller (2019). El objetivo de este trabajo consiste en la generación de resúmenes de textos cuyo contenido son clases académicas y está orientado a estudiantes que desean resumir una cátedra. A diferencia del trabajo anterior, en que se agrupaban todas las oraciones de múltiples los textos para

formar *clusters*, en este trabajo el set de datos solo se compone de 1 documento cada vez, que corresponde a una clase, y la *clusterización* se realiza solo con las oraciones del texto del estudiante. Primero se preprocesa el texto separándolo en las oraciones que lo componen y luego se proceden a vectorizarlas utilizando el modelo público preentrenado BERT.

Como el usuario del algoritmo son estudiantes, los autores definen que la cantidad de oraciones que compondrán el resumen generado queda determinada por el hiperparámetro k, que lo dejan a elección del usuario. Luego, para formar el resumen se elige una oración de cada uno de los k *clusters*, que corresponde a la que tiene menor distancia euclidiana al centroide.

En base a la metodología de los trabajos presentados se definen 4 experimentos a realizar en este trabajo, los que se exponen en la sección 3.4.

3. METODOLOGÍA Y EXPERIMENTOS

En este capítulo se describe la construcción del set de datos de entrenamiento, los experimentos definidos para realizar la etapa de *baseline* de clasificación de texto, la experimentación de clasificación de texto con BERT y los experimentos determinados para realizar el resumen extractivo de texto. También se incluirán las métricas de evaluación para cada uno de los modelos con que se experimentó.

3.1. Construcción del set de datos

Como se mencionó en el capítulo anterior, se tienen 166.111 comentarios recibidos en las encuestas de satisfacción y se requiere construir un set de datos de entrenamiento con comentarios clasificados manualmente para poder entrenar un modelo de clasificación mediante aprendizaje supervisado. Se construyó un set de datos de entrenamiento que consta de 5.402 comentarios con sus categorías respectivas asociadas. Se tomó la decisión de etiquetar tal cantidad de comentarios de acuerdo a los recursos humanos que se disponían y a que la experiencia del tesista, en base a la revisión de múltiples trabajos de PLN, indica que sería suficiente para alcanzar un nivel de generalización de la información aceptable. El proceso de etiquetamiento de comentarios consistió en tomar una muestra del set de datos completo de más de 160 mil encuestas, leer cada comentario de la muestra y etiquetarlo manualmente de acuerdo a la definición de cada eje de negocio, que se muestra en la tabla 1.1.

El etiquetamiento lo desarrolló el tesista en conjunto con el equipo que creó la encuesta de satisfacción de la empresa. Este proceso fue un desarrollo iterativo, en que constantemente surgían dudas de comprensión lectora que hacían difícil determinar a qué categoría pertenecía un comentario debido a la ambigüedad de estos. Las dudas se resolvieron en conjunto con el equipo de la empresa, y el proceso tomó más de 600 horas hombre desde que se comenzó a etiquetar hasta terminar. En la tabla 3.1, se

muestran algunos ejemplos de los comentarios recibidos en la encuesta junto con sus categorías respectivas asociadas.

Durante el etiquetamiento de comentarios se tomó la decisión de etiquetar con polaridad positiva a comentarios que solo nombraban un aspecto del negocio sin mencionar explícitamente algo positivo, en lugar de etiquetarlo como neutro o no etiquetarlo. Por ejemplo los comentarios "Muchos productos" y "La atención de los vendedores" se etiquetaron con polaridad positiva. El motivo de este criterio adoptado es que la gran mayoría de los clientes que emitieron este tipo de comentarios evaluaron su experiencia de compra con nota 4 o 5 en escala de 1 a 5.

La muestra de comentarios fue hecha al azar, con el criterio de que al menos el 90% de los comentarios deben tener un largo mayor a 5 palabras y al menos un 50% debe ser mayor a 15 palabras. Esto se hizo para que en el set de datos de entrenamiento existan comentarios suficientemente largos para el modelo sea capaz analizar textos que contengan contenido respecto de más de un eje de negocio y que podría ser difícil de extraer. También se evitó seleccionar comentarios que clientes contestaron sin intención de comunicar algo, por ejemplo comentarios con menos de 3 palabras y con secuencias de letras que no son legibles como una palabra en español u otro idioma. El set de datos completo presenta un promedio de $13,64$ palabras por comentario, con una desviación estándar de $16,74$ y en el set de datos etiquetados el promedio de palabras por comentario es $21,72$, con desviación estándar de $22,06$.

Existen 13 ejes de negocio que la empresa desea evaluar, sin embargo el set de datos entregado corresponde solo a la evaluación de la empresa a clientes que compran en locales. La empresa hace una encuesta distinta para los clientes que compran a través de internet y es por esto que de los 13 ejes iniciales que se desean evaluar solo se tiene suficiente información de los siguientes: 1) Al Pagar, 2) Atención, 3) Producto, 4) Tienda Física, 5) Precio, 6) Despacho, 7) Facilidad, 8) Puntos Cencosud, 9) Tarjeta Cencosud y 10) Servicio al Cliente (SAC). Se descartan los ejes "Retiro en Tienda", "Sitio Web" y "Call Center" debido a que la empresa captura esta información a través de la encuesta de compra en internet y en el set de

Tabla 3.1. Ejemplos de comentarios etiquetados.

Comentarios	**Categorías**
“la atención en la tienda por parte del vendedor sentí poco interés, con respuestas muy escuetas respecto del producto, ahora el sistema de despacho, la atención es pésima, para entregar el producto se demoraron 25 minutos desde el momento que al trabajador le entregaron la orden de despacho, súmele a eso el lugar es inhóspito en plena calle, el encargado actúa con una indolencia y desidia que hace sentir que esta haciendo una favor al cliente.”	Atención Negativo, Retiro Negativo.
“Los productos con descuento no marcaban el descuento y tuve que esperar para que verificaran y despues llamaran a la jefa de departamento y aplicar el descuento”	Precio Negativo.
“Me gusta la tienda ordenada, y sus trabajadores muy amables,,, pregunta que les hacía me respondían muy amablemente”	Tienda Física Positivo, Atención Positivo.
“Rapidez en caja, encontré lo que buscaba, excelente opción de envio de boletA al correo asi no se extravían”	Al Pagar Positivo, Producto Positivo, Otros Positivo.
“porque mi compra tuvo que ver con canje de puntos, y no tenían una guía en donde pudiera ver a qué optar con los puntos que tenía y ellos en lo personal no tenía claridad de los productos. Me mandaron a que lo consulte yo misma en mi celular.La atención del Señor de la caja estuvo muy buena.”	Puntos Cencosud Negativo, Al Pagar Positivo.
“Tienda hordenada variedades de produ”	Tienda Física Positivo, Producto Positivo.
“X la atencion”	Atención Positivo.
“la compra fue rápida. fuimos producto de un cambio y la información estuvo bien entregada desde el principio.”	Facilidad Positivo.

datos no existen comentarios suficientes para modelarlos de manera representativa. Además se agregó la categoría 11) “Otros” que abarca todos los comentarios un cliente emite sobre algún aspecto del proceso de compra que no corresponde a ninguna de las categorías mencionadas anteriormente. Estos tratan sobre la falta de bolsas en el local, sobre situaciones con guardias de seguridad, de disponibilidad de papel de regalo y más. También, los comentarios cuyo contenido no correspondía a

ninguna de las categorías anteriores y tampoco encajaban en la categoría “Otros” no se les asignó ninguna etiqueta.

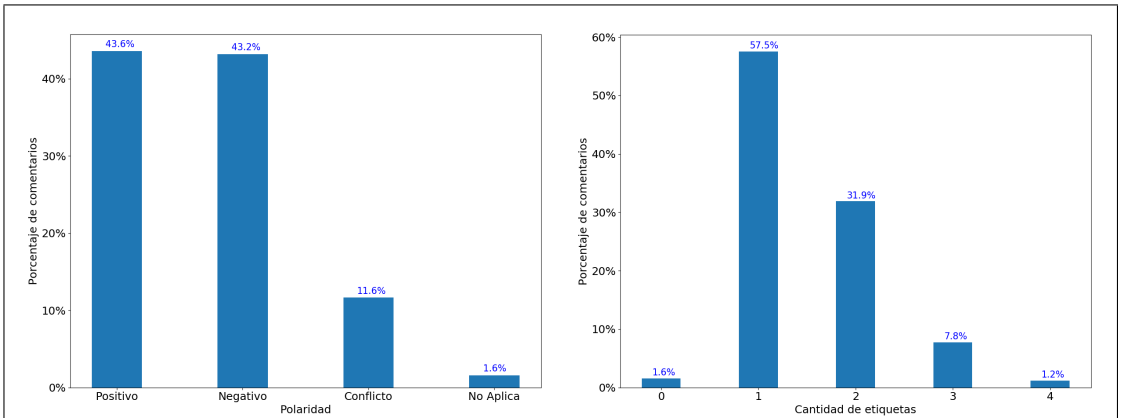
En la categorización de comentarios, a cada uno se le asigna un set de etiquetas o categorías de acuerdo a lo que contestó el cliente. Cada uno de estos ejes se transforma en una etiqueta que se le añade a un comentario si este habla del respectivo eje y esta puede ser positiva o negativa. De las 22 categorías posibles, se descartan las categorías “Despacho-Positivo”, “Facilidad-Negativo”, “Tarjeta Cencosud-Positivo”, “Puntos Cencosud-Positivo” y “Otros-Positivo” ya que no existen suficientes comentarios para lograr caracterizarlas o modelarlas correctamente. El criterio utilizado para descartarlas fue que tuvieran presencia en menos del 1% de los comentarios, es decir que existan menos de 54 comentarios etiquetados de esa manera en el set de datos de entrenamiento. En total se han etiquetado 17 categorías sobre los comentarios, las que son 1) Al Pagar-Positivo, 2) Al Pagar-Negativo, 3) Atención-Positivo, 4) Atención-Negativo, 5) Despacho-Negativo, 6) Facilidad-Positivo, 7) Otros-Negativo 8), Precio-Positivo, 9) Precio-Negativo, 10) Producto-Positivo, 11) Producto-Negativo, 12) Puntos Cencosud-Negativo, 13) SAC-Positivo, 14) SAC-Negativo, 15) Tarjeta Cencosud-Negativo, 16) Tienda Física-Positivo y 17) Tienda Física-Negativo.

Una vez finalizado el etiquetamiento las etiquetas asignadas por comentario varían entre 0 y 4. En la tabla 3.2 se muestran la cantidad de etiquetas que existen por categoría. El promedio cantidad de etiquetas por comentario y las distintas combinaciones de etiquetas son medidas usualmente empleadas para resumir las propiedades del set de entrenamiento, tienen valores de $1,47$ y 296 respectivamente.

En la figura 3.1(a) se muestra la polaridad de las evaluaciones de los clientes y se aprecia que un 11,6% de los comentarios emitidos presentan una polaridad conflictiva respecto a su experiencia, es decir que consideraron un aspecto del negocio positivamente y otro negativamente en la misma evaluación. El la figura 3.1(b) se muestra cantidad de etiquetas por comentario, ósea el número de tópicos sobre los que escribe cada cliente en una evaluación.

Tabla 3.2. Cantidad de etiquetas por categoría. En total son 5.402 comentarios etiquetados.

	Positivo	Negativo	Total
Atención	1706	1293	2999
Despacho	0	61	61
Precio	551	256	807
SAC	59	85	144
Al Pagar	546	948	1494
Tarjeta Cencosud	0	85	85
Puntos Cencosud	0	54	54
Producto	762	456	1218
Facilidad	231	0	231
Tienda Física	341	367	708
Otros	0	160	160



(a) Polaridad de comentarios (b) Número de Tópicos por comentario

Figura 3.1. Polaridad y número de tópicos por comentario.

El set de datos presenta un extremo desbalance respecto a la cantidad de etiquetas por categoría. La mayoría de los comentarios tratan respecto de la atención brindada por vendedores en el local, del proceso de pago y productos, y en mucha menor proporción se habla de los otros aspectos del negocio. En el figura 3.2 se muestra el porcentaje de comentarios que está asociado a cada categoría. Además, algunas categorías presentan correlación, marcando una tendencia a una correlación positiva entre comentarios que tienen etiquetas con la misma polaridad y correlación negativa

en etiquetas con polaridad distinta. En la figura 3.3 se muestra un análisis gráfico de la correlación entre las categorías. Por último, en la figura 3.4 se muestra la distribución del número de palabras por comentario y se observa que la mayor cantidad de comentarios tiene menos de 25 palabras.

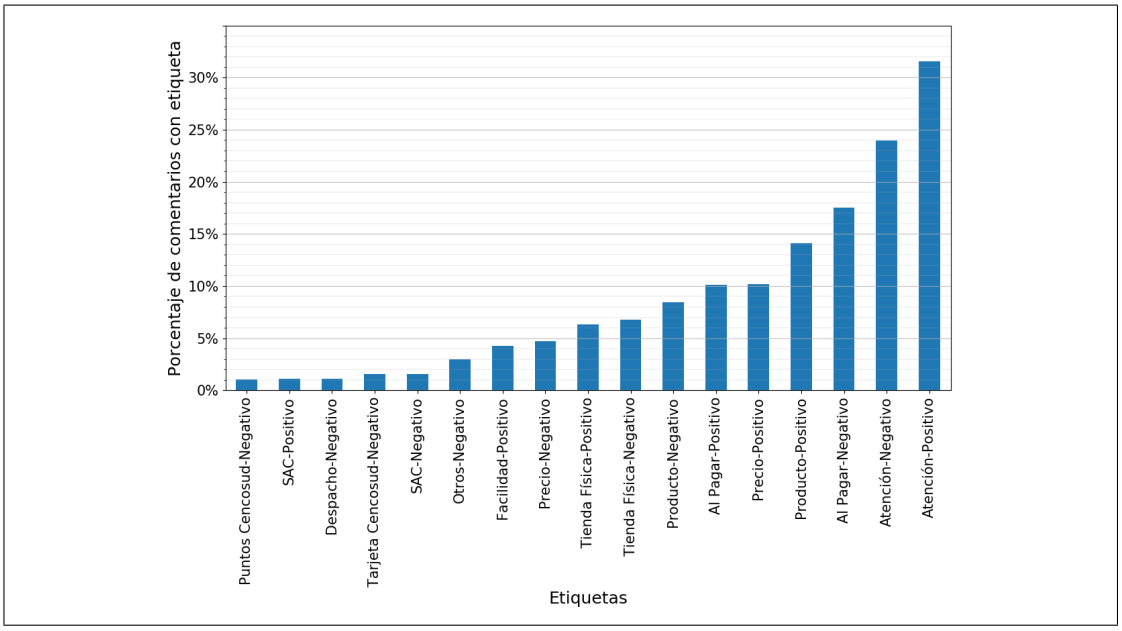


Figura 3.2. Distribución de etiquetas.

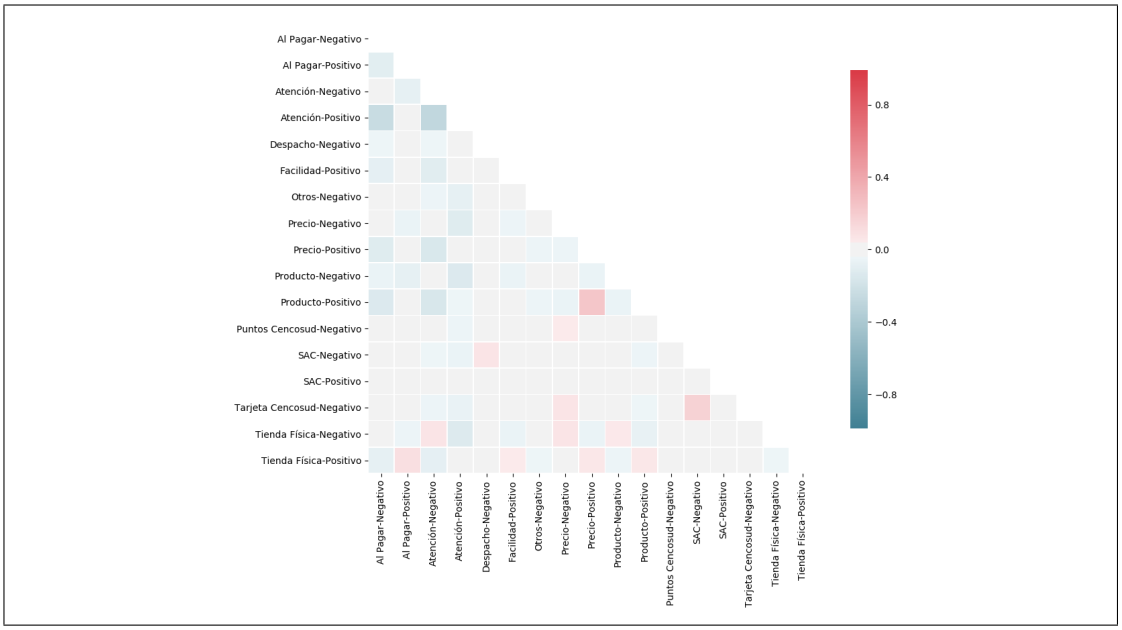


Figura 3.3. Correlación de las categorías.

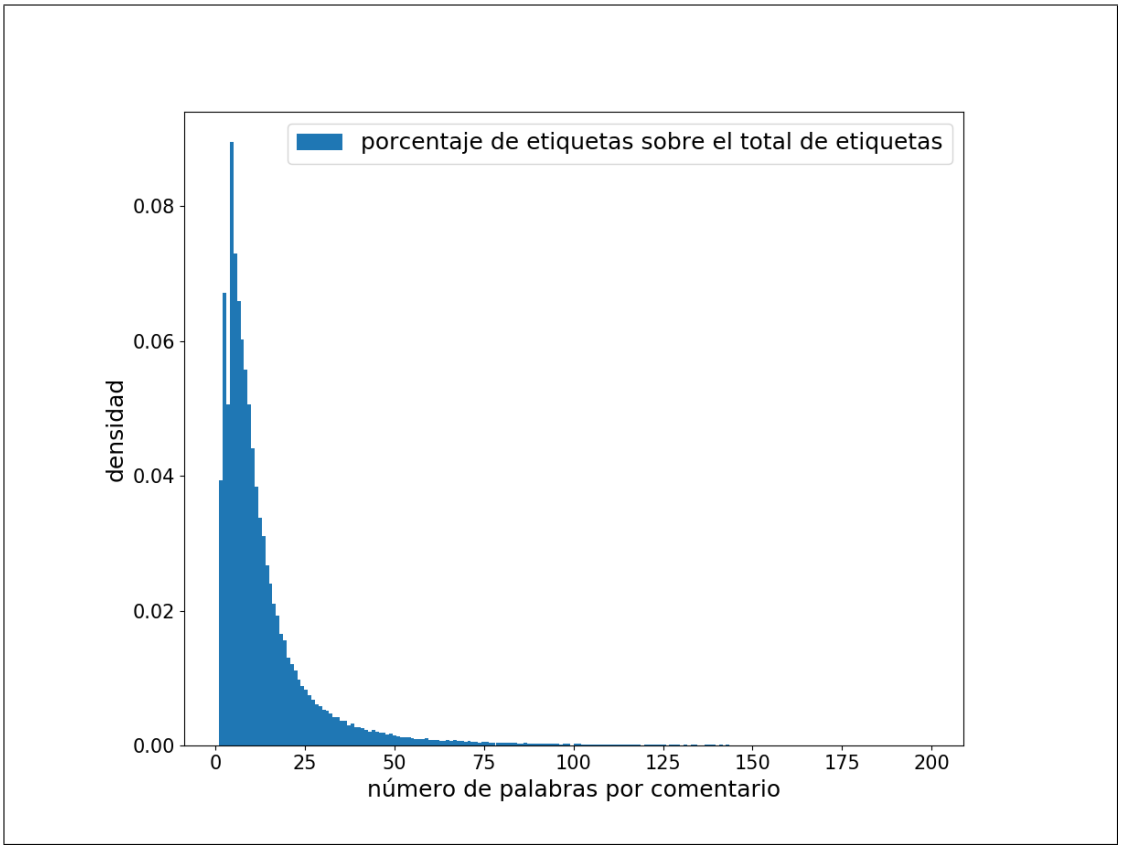


Figura 3.4. Densidad de palabras por comentario.

3.2. Baseline de Clasificación de Multi-Label de Texto

El *baseline* de clasificación de texto definido en este trabajo tiene por primer objetivo comparar diferentes factores en el preprocesamiento del texto y analizar cómo afectan el rendimiento final de la clasificación. El segundo objetivo es analizar el efecto de aumentar los datos de entrenamiento de manera artificial para mitigar el problema del desbalance de datos por categoría en el set de entrenamiento. El tercer objetivo es comparar el rendimiento de clasificación obtenido usando diferentes modelos clasificación *multi-label*, pertenecientes a las 3 categorías que distinguen la forma de abordar el problema de clasificación *multi-label* mencionadas en el capítulo anterior. Por último se quiere establecer métricas de evaluación que midan rendimiento de clasificación *multi-label* de texto, modelando el texto

matemáticamente usando FastText, y así poder comparar estas métricas establecidas con el resultado de clasificación obtenido usando BERT.

La clasificación de texto consistirá en entrenar modelos de clasificación *multi-label* mediante aprendizaje supervisado utilizando el set de datos etiquetados y usando validación cruzada para obtener los estimadores de las métricas de evaluación del rendimiento de clasificación de cada modelo.

La validación cruzada consiste en generar 5 particiones de similar tamaño del set de datos, se utilizan 4 de estas como set de entrenamiento para entrenar el modelo de clasificación *multi-label* y la restante se utiliza como set de *test* para calcular las métricas de rendimiento del modelo entrenado. Se usa el modelo para predecir los ejemplos del set de *test* y se comparan las predicciones con las etiquetas reales. Este proceso se repite 5 veces, cambiando las combinaciones de particiones de set de entrenamiento y *test* de tal manera que el set de *test* en cada iteración sea una partición distinta en cada iteración. Finalmente se crea un estimador de las métricas de rendimiento promediando los 5 resultados obtenidos a partir del entrenamiento del modelo con los 5 sets de entrenamiento conformados, uno en cada iteración de la validación cruzada. Dado que particionar el set de datos al azar podría generar un desbalance de la proporción de las etiquetas set de entrenamiento y *test* en cada combinación de las particiones, la división del set de datos en particiones se hará implementado el algoritmo de *Iterative Stratification* Sechidis et al. (2011), cuyo objetivo es generar una partición del set de datos en que la proporción de etiquetas de cada categoría varía de forma mínima en cada subset. El algoritmo presentado por los autores se puede ver en el Anexo.

Primero, se analizan 3 factores relacionados con la forma de *tokenizar* el texto en el *pipeline* de procesamiento. Se implementarán 9 formas de *tokenizar*, y se pretende analizar el efecto que tiene en la clasificación el uso de distintas expresiones regulares, la implementación de corrección de palabras y la remoción de acentos gráficos de los *tokens*.

Se construyeron 9 set de comentarios preprocesados, cuyas *tokenizaciones* son distintas de acuerdo a los procesos de transformación de texto que se aplicaron. El objetivo es analizar sus efectos en el resultado de clasificación. Primero se definen 3 expresiones regulares que segmentarán el texto considerando distintos factores que pueden influir en la *tokenización*. La primera, toma en cuenta caracteres especiales "%" y "$" para poder distinguir cadenas de texto relacionadas con precios y porcentajes, la segunda solo toma en cuenta caracteres alfanuméricos y la tercera solo las letras del alfabeto. Luego a cada una de las 3 *tokenizaciones* se les aplicó el método de corrección de palabras indicado en la sección 2.3.1. Finalmente, a cada uno de los 3 sets de comentarios corregidos se le aplicó la remoción de acentos gráficos. En la siguiente lista se exponen las *tokenizaciones* que se emplearán para entrenar los modelos de lenguaje distribuidos en el *baseline* de clasificación.

1) Transformación de texto a minúscula → *regex* que incluye caracteres especiales "$", "?", "!", "%" y caracteres alfanuméricos con el objetivo de detectar precios, porcentajes de descuento, preguntas y exclamaciones → remoción de *stop words*.
2) Transformación de texto a minúscula → *regex* de caracteres alfanuméricos → remoción de *stop words*.
3) Transformación de texto a minúscula → *regex* de caracteres de las letras del alfabeto → remoción de *stop words*.
4) Transformación de texto a minúscula → *regex* que incluye caracteres especiales "$", "?", "!", "%" y caracteres alfanuméricos con el objetivo de detectar precios, porcentajes de descuento, preguntas y exclamaciones → corrección de palabras mal escritas → remoción de *stop words*.
5) Transformación de texto a minúscula → *regex* de caracteres alfanuméricos → corrección de palabras mal escritas → remoción de *stop words*.
6) Transformación de texto a minúscula → *regex* de caracteres de las letras del alfabeto → corrección de palabras mal escritas → remoción de *stop words*.
7) Transformación de texto a minúscula → *regex* que incluye caracteres especiales "$", "?", "!", "%" y caracteres alfanuméricos con el objetivo de

detectar precios, porcentajes de descuento, preguntas y exclamaciones → corrección de palabras mal escritas → remoción de *stop words* → eliminación de acentos gráficos.

8) Transformación de texto a minúscula → *regex* de caracteres alfanuméricos → corrección de palabras mal escritas → remoción de *stop words* → eliminación de acentos gráficos.
9) Transformación de texto a minúscula → *regex* de caracteres de las letras del alfabeto → corrección de palabras mal escritas → remoción de *stop words* → eliminación de acentos gráficos.

En la práctica se entrena un modelo FastText con cada uno de los 9 sets de comentarios *tokenizados*. Luego los comentarios del set de entrenamiento se vectorizan utilizando los 9 modelos FastText entrenados.

Segundo se analiza el efecto que tiene la implementación de técnicas de aumento artificial de datos (*data augmentation*), que tienen por objetivo generar datos de entrenamiento artificiales para combatir el desbalance de clases en el set de datos que puede empeorar el rendimiento de la clasificación. Basado en el trabajo "Easy Data Augmentation" Wei & Zou (2019) se implementarán las técnicas de *data augmentation* para campos de texto sobre los comentarios que pertenecen a categorías de tamaño reducido.

(i) Reemplazo de sinónimos: se reemplazan n palabras del comentario por un sinónimo al azar.
(ii) Inserción al azar: se inserta en una posición al azar un sinónimo de una palabra al azar n veces.
(iii) Intercambio al azar: se intercambian las posiciones de 2 palabras al azar n veces.
(iv) Eliminación al azar: cada palabra del comentario se elimina con probabilidad p.

Las categorías de tamaño reducido se definieron como las que tienen menos de 10% de presencia respecto del total de etiquetas presentes en el set de datos. Estas se pueden ver en la figura 3.2. Esto generará 4 comentarios artificiales por cada ejemplo del set de entrenamiento utilizado para entrenar un modelo de clasificación, y se pretende evaluar el efecto que tiene en las métricas de evaluación de la clasificación final.

Tercero, se implementaron 3 algoritmos de clasificación *multi-label*, los que son *Multi Layer Perceptron Classifier*, *Classifier Chains* y *Random K-Labelsets*, y se procederá a entrenarlos, con cada uno de los 9 set de vectores obtenidos de acuerdo a las *tokenizaciones* distintas, y además se entrenarán implementando y no implementando "Easy Data Augmentation" (EDA). Luego se evaluará su rendimiento de clasificación de acuerdo a las métricas de evaluación definidas en la subsección 3.2.1, y así se determinará el mejor modelo del *baseline* que se utilizará para compararlo con BERT.

En resumen se generaron 9 *tokenizaciones* distintas, y se entrenaron 9 modelos FastText, uno por cada *tokenización*, produciendo 9 sets de vectores de palabras. Los comentarios se modelaron como el promedio de los vectores de los *tokens* que contiene. Con cada uno de los 3 algoritmos de clasificación *multi-label* se entrenaron 18 modelos, dos por cada uno de los 9 sets de vectores de palabras, una vez sin usar el aumento artificial de datos y otra implementando EDA. Durante el entrenamiento, se ajustarán los hiperparámetros definidos por el modelador para elevar el rendimiento de los modelos de acuerdo a las métricas de evaluación que se describen en la siguiente subsección 3.2.1. Finalmente se analizarán los modelos para determinar cómo influyen los distintos factores de preprocesamiento en la clasificación y se elegirá el mejor modelo como *baseline* de clasificación.

3.2.1. Métricas de Evalución

Se tiene un set de ejemplos de *test* determinado por una de las particiones de la validación cruzada, que consta de n ejemplos (x_i, y_i), $x_i \in \mathbb{R}^D$, $y_i \in \{0,1\}^k$, con el set de etiquetas L y $|L| = k$. Una vez que se ha entrenado un clasificador *multi-label*

$h : \mathbb{R}^D \rightarrow \{0,1\}^k$, se desea evaluar el rendimiento del modelo. El término etiqueta relevante para un ejemplo se refiere a cuando $y_i = 1$ para algún comentario, es decir el ejemplo se le asignó una etiqueta manualmente. Se determinarán 7 métricas de evaluación que tienen como objetivo caracterizar la clasificación del modelo sobre los datos del set *test* no observados por este.

Se definen conceptos previos necesarios para entender lo que evalúa cada métrica. Primero, el modelo predice las etiquetas $\hat{y}$ para un determinado ejemplo x, cuyas etiquetas asignadas manualmente corresponden a y. Se definen como verdaderos positivos (vp) las etiquetas asignadas correctamente, es decir que si $y_i = 1$ para algún $i \in \{1, ..., k\}$, entonces $\hat{y}_i = 1$. Los falsos positivos (fp) son tal que el clasificador ha asignado una etiqueta que no corresponde al ejemplo, $y_i = 0$ e $\hat{y}_i = 1$. Verdaderos negativos (vn) se producen cuando $y_i = 0$ e $\hat{y}_i = 0$, y los falsos negativos (fn) se refieren a cuando el modelo no asigna una etiqueta que sí es relevante para el comentario, es decir $y_i = 1$ e $\hat{y}_i = 0$.

Las métricas más comunes utilizadas para modelos *multi-clase* son la precisión, la exhaustividad (*recall*) y el puntaje F1. Todas están definidas entre los valores entre 0 y 1, siendo 1 la mejor métrica que se puede obtener. La precisión tiene que ver con la fracción de etiquetas predichas correctamente por el modelo sobre el total de todos las etiquetas predichas, es decir la fracción $vp/(vp + fp)$. El *recall* indica la fracción etiquetas relevantes predichas correctamente sobre el total de etiquetas relevantes y se define como $vp/(vp + fn)$.

La precisión y *recall* por sí solos no son capaces de caracterizar que tan bueno es el modelo, por ejemplo, si el modelo ha predicho bien 1 etiqueta en el set de ejemplos pero no asignó 10 etiquetas que correspondía asignar, es decir tiene muchos falsos negativos, será un modelo con precisión de 100%, sin embargo el modelo no es eficaz. El puntaje F1 es equivalente a la media armónica entre la precisión y el *recall*, que se define como $(2 \cdot precision \cdot recall)/(precision + recall) = (2 \cdot vp)/(2 \cdot vp + fp + fn)$. La media armónica es menos sensible a los valores cercanos a 1 respecto a la

media aritmética, y es muy sensible a los valores próximos a 0, de tal manera que si la precisión o el *recall* de un modelo son pequeñas, el indicador F1 será menor.

En la clasificación *multi-label*, se pueden medir la precisión, el *recall* y el puntaje F1 de acuerdo a todos los comentarios del set de *test*, o agrupando las mediciones por categoría y luego promediando los resultados obtenidos en cada una. La métrica que agrupa las mediciones por categoría y luego las promedia le da más importancia a categorías con pocos ejemplos etiquetados, que en general tienen peor rendimiento de clasificación. A continuación se definen las métricas de evaluación que se utilizarán.

Macro Precisión: Para cada categoría $i \in 1, ..., k$ calcula todos los verdaderos positivos (vp_i) sobre la suma de verdaderos positivos y falsos positivos (fp_i), considerada como una clase binaria, y promedia los resultados obtenidos en cada una de las k categorías.

$$MacroPrecision = \frac{1}{k} \sum_{i=1}^{k} \frac{vp_i}{vp_i + fp_i} \tag{3.1}$$

Macro *Recall*: Para cada categoría $i \in 1, ..., k$ calcula todos los verdaderos positivos sobre la suma de verdaderos positivos y falsos negativos (fn_i), considerada como una clase binaria, y promedia los resultados obtenidos en cada una de las k categorías.

$$MacroRecall = \frac{1}{k} \sum_{i=1}^{k} \frac{vp_i}{vp_i + fn_i} \tag{3.2}$$

Macro F1: Para cada categoría $i \in 1, ..., k$, sea p_i y r_i la precisión y el *recall* de la categoría i respectivamente, se define

$$MacroF1 = \frac{1}{k} \sum_{i=1}^{k} \frac{2 \times p_i \times r_i}{p_i + r_i} \tag{3.3}$$

Micro Precisión: Es la precisión promediada entre todos los pares de ejemplos y etiquetas del set de *test*.

$$MicroPrecision = \frac{\sum_{i=1}^{k} vp_i}{\sum_{i=1}^{k} vp_i + \sum_{i=1}^{k} fp_i} \tag{3.4}$$

Micro *Recall*: Es el *recall* promediado entre todos los pares de ejemplos y etiquetas del set de *test*.

$$MicroRecall = \frac{\sum_{i=1}^{k} vp_i}{\sum_{i=1}^{k} vp_i + \sum_{i=1}^{k} fn_i} \tag{3.5}$$

Micro *F1*: Es el promedio armónico entre la micro precisión y el micro *recall*, es decir el puntaje F1 calculado con todos los pares de ejemplos y etiquetas del set de *test*. Sea mp_i y mr_i la micro precisión y la micro *recall* del set de *test*, se define

$$MicroF1 = \frac{2 \times mp_i \times mr_i}{mp_i + mr_i} \tag{3.6}$$

Exactitud del Subset (*Subset Accuracy*): Calcula la proporción de todos los comentarios en que todas sus etiquetas se predijeron correctamente sobre todos los comentarios del set de test. Esta métrica es bastante estricta, ya que dado un ejemplo que tiene 4 etiquetas relevantes, y se han clasificado 3 correctamente y ningún falso positivo, la métrica considerará que es un ejemplo mal clasificado. Se define

$$SubsetAccuracy = \frac{1}{n} \sum_{i=1}^{n} I(h(x_i) = y_i) \tag{3.7}$$

3.3. BERT para Clasificación de Multi-Label de Texto

El preprocesamiento que se realizó a los comentarios de los clientes fue transformar el texto a minúscula, aplicar la corrección de palabras y eliminar los acentos gráficos. A diferencia de los modelos presentados anteriormente, no es necesario remover las *stop words* ya que debido a la arquitectura de BERT, este le asigna menos atención a las palabras no informativas del texto y por lo tanto genera ruido en la caracterización de comentarios, que usualmente afecta negativamente el

resultado de clasificación de estos. Esto se comprobó empíricamente en experimentos preliminares.

Para *tokenizar* el texto se empleó el *tokenizador* integrado en la librería *transformers*, que emplea una segmentación de las palabras en subpalabras, utilizando el vocabulario del modelo original entrenado por el equipo de investigación de Google. Se implementó el método de validación cruzada para determinar las métricas de evaluación del modelo, igual que como se hizo en el *baseline* de clasificación.

Se utilizó el modelo Bert Multilingual Uncased disponible en la librería. Se experimentó utilizando 3 formas distintas de extraer el *output* de BERT. Es decir, se construyeron 3 modelos BERT que se diferencian en la forma de conectar el clasificador sobre la última capa. El clasificador utilizado es *Multi Layer Perceptron Classifier* (MLP) utilizado en el *baseline*, con la diferencia de que es implementado con 1 capa oculta en lugar de 2 ya que la velocidad de convergencia y puntaje F1 del modelo fue peor cuando se entrenaba el clasificador como fue definido originalmente. Ahora se detallan los 3 modelos propuestos

El primer modelo BERT propuesto "CLS pool" extrae solo el vector correspondiente al *token* `[CLS]` como lo sugiere el trabajo original, e implementa el clasificador MLP con *input* de dimensión 768 y *ouput* de dimensión 17. La función de pérdida es Focal Loss, que se ejecuta independientemente sobre cada *output* de los BS ejemplos de cada iteración de entrenamiento de la red. El segundo "Av. pool" y tercer "Av. pool excepto CLS" modelo se diferencian del primero en la forma de extraer el vector de *input* de MLP desde la capa de *output* de BERT. En el modelo "Av. pool" todos los vectores asociados a *tokens* se promedian y luego se ingresan al clasificador MLP. En el tercer modelo "Av. pool excepto CLS" se promedian todos los vectores asociados a *tokens* excepto el vector asociado al *token* `[CLS]`.

El algoritmo de optimización que se utilizó para minimizar el error durante el ajuste de parámetros de BERT es Adam, integrado en la librería *transformers*, presentado en *Decoupled Weight Decay Regularization* Loshchilov & Hutter (2017). Se probó

entrenar el modelo con tamaño de lote (*batch size*) igual a 16 y 24, tasa de aprendizaje igual a 6×10^{-6}, 1×10^{-5}, 4×10^{-5} y 1×10^{-4}. La función de pérdida es *Focal Loss*, con parámetros $\alpha = 0.5$ y $\gamma = 2$. El largo de la secuencia de *input* utilizado fue 200.

Se definen dos experimentos más, el primero consiste en entrenar el mejor modelo BERT, determinado por la métrica micro F1, usando una *tokenización* que no involucra corrección de palabras. El segundo corresponde a utilizar aumento artificial de datos utilizando EDA, similar a como se hizo en el *baseline*. Luego se comparan los resultados obtenidos.

Finalmente se comparan ls resultados de la clasificación *multi-label* de texto con BERT en este trabajo con los resultados que se presentan en "DocBERT: BERT for Document Classification" Adhikari et al. (2019), y en que los autores implementan BERT para clasificación *multi-label* de texto en sets de datos públicos que contienen textos asociados a un conjunto definido de etiquetas.

3.4. Resumen Extractivo de Texto

En este trabajo se experimentó realizando 4 experimentos para resumir los comentarios recibidos en un determinado periodo de tiempo y local, todos basados en la metodología implementada en los trabajos de Chou et al. (2019) y D. Miller (2019). Además se incorporó la información obtenida de la clasificación para generar los resúmenes. En adelante, a todo el texto que contestó el cliente al responder la encuesta de satisfacción será tratado como un "comentario", como se venía haciendo, y una oración correspondiente a la subdivisión un comentario determinada por la separación de este de acuerdo su puntuación (puntos y comas) será tratada como una "frase". Por ejemplo un comentario "Buena atención, tienda ordenada. Lento al pagar" se divide en las frases "Buena atención", "tienda ordenada" y "Lento al pagar".

Para realizar los experimentos se construyó un set de *test* que se utilizará para evaluar la calidad de los resúmenes generados. Se seleccionaron al azar 10 periodos

de tiempo y locales, y se crearon manualmente 2 resúmenes por cada uno, de manera que son utilizados como fuente de verdad. Luego los resúmenes generados automáticamente se comparan con los resúmenes hechos manualmente. En la subsección 3.4.1 se detalla como se realiza la comparación y como se determina la métrica de evaluación correspondiente.

El primer experimento se denomina"all-comments-k-means", en que primero se preprocesa el texto de los más de 160 mil comentarios de manera que puedan ser clasificados con BERT. Luego los comentarios son agrupados con repetición de acuerdo a las 17 combinaciones (<eje de negocio>, <polaridad>) que el clasificador les asignó. Los comentarios a los que no se les asignó ninguna etiqueta fueron descartados, ya que se se verificó que su contenido era mayormente no informativo.

Luego se obtienen los vectores que representan los comentarios, que corresponde al vector de *output* C de BERT, es decir, para realizar esto se elimina el clasificador añadido sobre BERT, luego se ingresa cada comentario como *input* y se extrae el vector C correspondiente a la primera posición del *output* generado. Sobre cada uno de los 17 conjuntos de vectores de comentarios $E_i, i \in \{1, ..., 17\}$ se ejecuta varias veces el algoritmo K-means, con el valor hiperparámetro k en el rango $[2, 30)$, y se determina el k final elegido de acuerdo al método de la silueta. Como resultado, se tienen $k_i, i \in \{1, ..., 17\}$ centroides por cada uno de los conjuntos formados. El objetivo detrás de esta *clusterización* es que el algoritmo logre identificar los subtópicos que puedan existir dentro de cada categoría (<eje de negocio>, <polaridad>).

Enseguida, los comentarios recibidos en el periodo de tiempo y locales del set de *test* se subdividen en frases y se procede a vectorizarlas utilizando BERT. Luego, a cada frase se le asigna un puntaje que corresponde al valor de la distancia euclidiana con el centroide más cercano y se ordenan de menor a mayor. Esta lista ordenada se filtra descartando las frases cuyo centroide más cercano es el mismo, y se deja solo la que tiene menor distancia a él. Finalmente se seleccionan las S frases con menor distancia componen el resumen generado. El número de frases seleccionadas S se

determina por una heurística definida como

$$S = \frac{l}{2}, \text{ si } l < 20$$

$$S = 10, \text{ si } 20 \leq l < 30$$

$$S = 12, \text{ si } 30 \leq l < 50$$

$$S = \frac{l}{4}, \text{ si } 50 \leq l$$

donde l corresponde al total de comentarios del periodo a resumir.

El experimento dos, que se referenciará como "all-comments-split-k-means", es similar al anterior, pero en lugar de clasificar los comentarios para determinar los conjuntos $E_i, i \in \{1, ..., 17\}$ se clasifican las frases obtenidas de la subdivisión de estos de acuerdo a su puntuación.

El tercer experimento, referenciado como "period-k-means", consiste en implementar el algoritmo K-means sobre los vectores que representan las frases de los periodos de tiempo y local correspondientes al set de *test*, eligiendo k como la cantidad de frases que se desea que contenga el resumen, y es determinado por la heurística definida anteriormente. Se selecciona una frase por cada *cluster* que corresponde a la más cercana al centroide de cada uno. Este experimento no incorpora la información del clasificador y es similar al trabajo realizado en D. Miller (2019).

El cuarto experimento, denominado "period-clf-k-means", es similar al anterior, pero primero se clasifican los vectores que representan las frases y posteriormente se les concatena la clasificación de cada uno. Luego se implementa K-means y se seleccionan la frases del resumen de igual manera que en el experimento anterior.

También se realizó un *baseline* de comparación mediante la generación del resumen seleccionando frases al azar, que será referenciado como "random". Las métricas de evaluación utilizadas para medir el desempeño de la calidad de los resúmenes generados será ROUGE-1, ROUGE-2, ROUGE-3 y ROUGE-4, explicados a continuación.

3.4.1. Métricas de Evalución

La métrica de evaluación utilizada para comparar la calidad de los resúmenes generados automáticamente será ROUGE (*Recall-Oriented Understudy for Gisting Evaluation*) que es una de las más comunes para medir el rendimiento de resúmenes automáticos. Dado que no hay un único resumen ideal para un cierto texto dado, la evaluación consiste en comparar el resumen generado automáticamente con una muestra de uno o más resúmenes hechos manualmente por personas, que se consideran como la fuente de verdad. Los resúmenes ideales que se utilizaron para calcular esta métrica fueron escritos por el tesista y una persona capacitada.

La forma en que se mide el rendimiento del indicador consiste en dividir el texto del resumen generado automáticamente en grupos de n palabras que aparecen consecutivamente, por ejemplo, si $n = 2$ el resumen generado automáticamente "en la caja había fila" se divide en los grupos ("en la", "la caja", "caja había", "había fila"). Luego se hace lo mismo con el resumen ideal, y el indicador consiste en la proporción de grupos que aparecen en el resumen generado automáticamente con respecto del resumen ideal. Por ejemplo, si el resumen ideal es "en la caja había una larga fila", se divide en los grupos de 2 palabras consecutivas ("en la", "la caja", "caja había", "había una", "una larga", "larga fila"), y la métrica ROUGE-2 tendrá el valor $0,5$, ya que 3 de los 6 grupos del resumen ideal aparecen en los grupos del resumen generado. Matemáticamente el indicador ROUGE-N se define

$$ROUGE - N = \frac{\sum_{S \in R_a} \sum_{n_{gram} \in S} Count_{match}(n_{gram})}{\sum_{S \in R} \sum_{n_{gram} \in S} Count(n_{gram})} \tag{3.8}$$

donde R es el conjunto de resúmenes como fuente de verdad, R_a es el conjunto de resúmenes generados automáticamente, n_{gram} representa los grupos de palabras consecutivas de largo N y S representa el texto de resumen. En este trabajo se utilizan las métricas de evaluación ROUGE-1, ROUGE-2, ROUGE-3 y ROUGE-4, que corresponden al indicador ROUGE utilizando secuencias de palabras de largo 1, 2, 3 y 4 respectivamente.

4. RESULTADOS Y ANÁLISIS

4.1. Baseline de Clasificación de Texto

Se realizaron 9 *tokenizaciones*, con diferentes combinaciones de los tratamientos de transformación de texto, que son transformación a minúscula (m), remoción de *stop words* (rsw), *regex* de caracteres especiales (s), *regex* de caracteres alfanuméricos (an), *regex* de letras del alfabeto (l), corrección de palabras (c) y remoción de acentos gráficos (rag). El resumen del tamaño de vocabulario como resultado de cada tokenización se muestra en la tabla 4.1.

La selección de *stop words* a remover se basó en el set de *stop words* de la librería Natural Language Tool Kit (NLTK) Loper & Bird (2002). No se incluyeron en el conjunto de *stop words* palabras con alta frecuencia en el set de datos que pueden cambian la polaridad del comentario, por ejemplo las palabras “no” o “pero”. El conjunto de *stop words* consta de 303 palabras en total.

Tabla 4.1. Tamaño vocabulario de ejemplos de entrenamiento y reducción relativa a *tokenización* de mayor tamaño.

Tokenización	**Tamaño vocabulario**	**Reducción relativa**
m-rsw-s	38438	
m-rsw-an	38131	- 0.80%
m-rsw-l	37382	- 2.75%
m-rsw-s-c	37457	- 2.55%
m-rsw-an-c	37150	- 3.35%
m-rsw-l-c	36412	- 5.27%
m-rsw-s-c-rag	34998	- 8.95%
m-rsw-an-c-rag	34691	- 9.75%
m-rsw-l-c-rag	33953	- 11.67%

Se entrenaron 9 modelos FastText, uno por cada *tokenización*. El *corpus* de entrenamiento consistió en el set de datos de más de 160 mil encuestas, con más de 1 millón de palabras en total. Los parámetros de entrenamiento de los modelos son los

predefinidos en la librería *fasttext* de Python Bojanowski et al. (2016), eligiendo la dimensión de los vectores de salida $D = 300$.

Todos los experimentos se realizaron utilizando validación cruzada, y las particiones del set de entrenamiento y *test* se realizó mediante el algoritmo *iterative stratification*. El set de *test* representa el 20% del total de datos etiquetados. En la práctica el set de entrenamiento consta de 4.320 ejemplos aproximadamente y el set de *test* 1.080 ejemplos aproximadamente, ya que depende de la asignación dada por estratificación iterativa, y la cantidad de datos por partición de la validación cruzada varía en ± 2 en cada *subset*. No se decidió subdividir el set de *test* en uno de validación debido a que existen pocos ejemplos de entrenamiento en algunas categorías, y determinar un set de validación podría haber generado una muestra sesgada y con alta variabilidad en la determinación de estimador de la métrica.

El aumento artificial de datos de entrenamiento se hizo con la implementación de *Easy Data Augmentation* (EDA) Wei & Zou (2019). El proceso consiste en que a cada comentario se le aplica uno de los siguientes cambios: se reemplazan n palabras por su sinónimo, se insertan n palabras que sean sinónimo de alguna palabra de la frase, se intercambian n palabras o se eliminan palabras con probabilidad p. Siguiendo la recomendación del trabajo de Wei & Zou (2019) y en base a las características del set de entrenamiento, se determinó que $n = 0.1l$, siendo l el largo del comentario procesado, la probabilidad de de eliminar una palabra al azar es $p = 0.1$ y la cantidad de nuevos comentarios generados por cada comentario procesado por el algoritmo es $n_{aug} = 4$.

A diferencia de la implemantación que se hace en EDA, en que se añaden n_{aug} comentarios por cada uno perteneciente al set de entrenamiento, se eligió aumentar los datos de los comentarios cuyas etiquetas estén presentes en menos del 10% de los comentarios. Estas son “Puntos Cencosud-Negativo”, “SAC-Positivo”, “Despacho-Negativo”, “Tarjeta Cencosud-Negativo”, “SAC-Negativo”, “Otros-Negativo”, “Facilidad-Positivo”, “Precio-Negativo”, “Tienda Física-Positivo”, “Tienda Física-Negativo” y “Producto-Negativo”. Antes de la implementación de

EDA la desviación estándar de la proporción de cantidad de comentarios por categoría, normalizada por el número de categorías de los comentarios es de aproximadamente 5,75%, después de la implementación es de 3,57%. EDA aumenta el tamaño del set de entrenamiento a aproximadamente 6.830 nuevos comentarios sintéticos cada vez, un incremento de 58%. En la tabla 4.2 se expone un ejemplo de comentarios artificiales originados a partir de uno original.

Para realizar el reemplazo palabras por sus sinónimos se utiliza WordNet G. A. Miller (1995), que es una base de datos léxica para el idioma inglés que relaciona palabras con sus sinónimos. En WordNet cada palabra puede estar asociada a diferentes dominios en que se utiliza, por ejemplo *"bank"* está asociada a los dominios *"banking"* y *"furniture"* y dependiendo del dominio y el contexto en que se utiliza la palabra se puede identificar el sinónimo correcto para un reemplazo. Se utilizó la interfaz NLTK Wordnet en español a través de la librería SpaCy Honnibal & Montani (2017), y se utilizaron los dominios correspondientes al contexto de la encuesta de satisfacción para el reemplazo e inserción de sinónimos.

Tabla 4.2. Ejemplos de aumento artificial de datos con algoritmo EDA.

Comentario original pre procesado	"buena disposicion de los vendedores pero lo que es el departamento servicio al cliente es pesimo"
Comentario artificial 1	"feliz disposicion diamante vendedor pero división excusado aluminio solicitante pesimo"
Comentario artificial 2	"buena disposicion pesimo pero departamento servicio cliente vendedores"
Comentario artificial 3	"buena disposicion vendedores pero departamento servicio cliente pesimo"
Comentario artificial 4	"buena disposicion pero vendedores departamento servicio cliente pesimo"

Los indicadores más relevantes en la determinación del mejor modelo es el micro F1 y el macro F1 ya que no hay distinción entre los costos de que el modelo prediga falsos negativos y falsos positivos. Micro F1 apunta a que la mayoría de los ejemplos se clasifiquen bien en cuanto a precisión y *recall*, y la métrica macro F1 le da mayor

importancia a que cada categoría se clasifique correctamente, independiente de la cantidad de ejemplos por categoría. Al ser un set de datos desbalanceado, las categorías con pocos ejemplos tienen peor rendimiento de clasificación que categorías con más datos y esto se ve reflejado en el indicador macro F1.

La representación vectorial de un comentario se hizo como el promedio de los *embeddings* de palabras que contiene cada uno de acuerdo a la *tokenización* correspondiente. Después de realizar el ajuste de hiperparámetros de cada modelo, los resultados de la implementación de Rakel y *Classifier Chains* se exponen las tablas 4.3 y 4.4. El mejor modelo de cada configuración de preprocesamiento propuesta se eligió de acuerdo a la métrica micro F1. Los indicadores micro y macro se presentan con una "m" y "M" respectivamente. Todos los resultados se muestran truncados en el 4 decimal.

Tabla 4.3. Resultados Clasificación Rakel

Tokenización	m F1	M F1	m Prec.	M Prec.	m Rec.	M Rec.	SS Acc
m-rsw-s	0.7190	0.6041	0.7777	0.6775	0.6687	0.5543	0.5038
m-rsw-s (EDA)	0.7053	0.6118	0.7334	0.6295	0.6795	0.6046	0.4706
m-rsw-an	0.7177	0.6053	0.7836	0.6882	0.6620	0.5517	0.5025
m-rsw-an (EDA)	0.7046	0.6109	0.7313	0.6307	0.6799	0.6014	0.4619
m-rsw-l	0.7190	0.6094	0.7784	0.6870	0.6683	0.5604	0.5026
m-rsw-l (EDA)	0.7049	0.6146	0.7351	0.6293	0.6772	0.6121	0.4665
m-rsw-s-c	0.7170	0.6076	0.7751	0.6713	0.6671	0.5649	0.4967
m-rsw-s-c (EDA)	0.7089	0.6193	0.7421	0.6404	0.6786	0.6101	0.4713
m-rsw-an-c	**0.7270**	0.6254	**0.7838**	**0.6991**	0.6781	0.5776	**0.5147**
m-rsw-an-c (EDA)	0.7108	**0.6318**	0.7593	0.6633	0.6682	0.6171	0.4825
m-rsw-l-c	0.7198	0.6134	0.7812	0.6943	0.6674	0.5608	0.5036
m-rsw-l-c (EDA)	0.7160	**0.6318**	0.7615	0.6625	0.6757	0.6178	0.4930
m-rsw-s-c-rag	0.7205	0.6133	0.7746	0.6793	0.6736	0.5707	0.4988
m-rsw-s-c-rag (EDA)	0.7135	0.6191	0.7370	0.6306	**0.6914**	0.6187	0.4802
m-rsw-an-c-rag	0.7254	0.6192	0.7766	0.6809	0.6808	0.5789	0.5039
m-rsw-an-c-rag (EDA)	0.7109	0.6306	0.7491	0.6526	0.6765	**0.6225**	0.4832
m-rsw-l-c-rag	0.7258	0.6141	0.7811	0.6794	0.6779	0.5688	0.5128
m-rsw-l-c-rag (EDA)	0.7131	0.6300	0.7556	0.6543	0.6754	0.6215	0.4849

Tabla 4.4. Resultados Clasificación Classifier Chains

Tokenización	m F1	M F1	m Prec.	M Prec.	m Rec.	M Rec.	SS Acc
m-rsw-s	0.7293	0.6242	0.7804	0.6867	0.6847	0.5826	0.5391
m-rsw-s (EDA)	0.7149	0.6309	0.7344	0.6349	0.6965	0.6392	0.5071
m-rsw-an	0.7320	0.6281	0.7860	0.6985	0.6850	0.5819	0.5449
m-rsw-an (EDA)	0.7162	0.6312	0.7384	0.6355	0.6953	0.6421	0.5104
m-rsw-l	0.7316	0.6243	0.7828	0.6899	0.6868	0.5813	0.5417
m-rsw-l (EDA)	0.7156	0.6253	0.7334	0.6248	0.6986	0.6397	0.5112
m-rsw-s-c	0.7320	0.6364	0.7815	0.6979	0.6886	0.5949	0.5406
m-rsw-s-c (EDA)	0.7187	0.6317	0.7385	0.6310	0.7001	0.6461	0.5076
m-rsw-an-c	0.7286	0.6295	0.7809	**0.7006**	0.6829	0.5807	0.5378
m-rsw-an-c (EDA)	0.7108	0.6218	0.7305	0.6196	0.6922	0.6398	0.5012
m-rsw-l-c	0.7381	0.6289	**0.7892**	0.6964	0.6933	0.5839	0.5495
m-rsw-l-c (EDA)	0.7189	0.6293	0.7392	0.6289	0.6997	0.6458	0.5147
m-rsw-s-c-rag	0.7345	0.6285	0.7787	0.6841	0.6952	0.5905	0.5460
m-rsw-s-c-rag (EDA)	0.7177	0.6280	0.7390	0.6324	0.6977	0.6392	0.5123
m-rsw-an-c-rag	0.7357	0.6286	0.7806	0.6865	0.6958	0.5901	0.5449
m-rsw-an-c-rag (EDA)	0.7181	**0.6390**	0.7493	0.6485	0.6896	**0.6467**	0.5008
m-rsw-l-c-rag	**0.7398**	0.6330	0.7876	0.6935	0.6976	0.5934	**0.5519**
m-rsw-l-c-rag (EDA)	0.7188	0.6252	0.7374	0.6241	**0.7011**	0.6421	0.5126

Con respecto al clasificador basado en redes neuronales, se definió que cada capa oculta de la red tiene 1024 unidades, la función de activación de las capas internas es ReLu y función de activación de la capa *output* sigmoide. La red se entrena en *batches* de 128 ejemplos y con un criterio de término de entrenamiento de 2 iteraciones de no mejora de la función de pérdida para evitar *overfitting*. En promedio la red se entrena en aproximadamente menos de 80 épocas sin EDA y en menos de 55 con EDA. Durante el ajuste de hiperparámetros se varió los parámetros de la función *Focal Loss*, donde $\alpha = 1, \gamma = 1$ corresponde a la función de pérdida de Entropía Cruzada Binaria. Se observó que la mejor combinación de estos parámetros es $\alpha = 0.5, \gamma = 2$. Después del ajuste de hiperparámetros los resultados son los siguientes

Tabla 4.5. Resultados Multi Layer Perceptron Classifier

Tokenización	m F1	M F1	m Prec.	M Prec.	m Rec.	M Rec.	SS Acc
m-rsw-s	0.7464	0.6513	0.7852	0.7366	0.7114	0.6014	0.5510
m-rsw-s (EDA)	0.7407	0.6562	0.7602	0.6884	0.7226	0.6388	0.5336
m-rsw-an	0.7447	0.6581	0.7787	0.7359	0.7135	0.6096	0.5484
m-rsw-an (EDA)	0.7371	0.6618	0.7222	0.6556	**0.7532**	**0.6783**	0.5149
m-rsw-l	0.7496	0.6563	0.7783	0.7306	0.7230	0.6142	0.5484
m-rsw-l (EDA)	0.7409	0.6624	0.7621	0.6909	0.7210	0.6443	0.5326
m-rsw-s-c	0.7428	0.6469	0.7827	0.7198	0.7069	0.6003	0.5395
m-rsw-s-c (EDA)	0.7359	0.6510	0.7601	0.6918	0.7133	0.6260	0.5260
m-rsw-an-c	0.7468	0.6568	0.7807	**0.7385**	0.7157	0.6128	0.5443
m-rsw-an-c (EDA)	0.7407	0.6586	0.7579	0.6904	0.7244	0.6410	0.5380
m-rsw-l-c	0.7470	0.6567	0.7771	0.7170	0.7193	0.6214	0.5486
m-rsw-l-c (EDA)	0.7420	0.6528	0.7637	0.6895	0.7217	0.6322	0.5337
m-rsw-s-c-rag	0.7520	0.6634	0.7842	0.7376	0.7230	0.6202	0.5552
m-rsw-s-c-rag (EDA)	0.7471	**0.6684**	0.7612	0.6958	0.7335	0.6544	0.5376
m-rsw-an-c-rag	0.7505	0.6605	0.7868	0.7347	0.7178	0.6134	0.5484
m-rsw-an-c-rag (EDA)	0.7435	0.6662	0.7616	0.7013	0.7264	0.6467	0.5367
m-rsw-l-c-rag	**0.7534**	0.6617	**0.7880**	0.7381	0.7217	0.6176	**0.5571**
m-rsw-l-c-rag (EDA)	0.7431	0.6552	0.7613	0.6850	0.7260	0.6351	0.5376

4.1.1. Ánalisis de Resultados de la Clasificación Baseline

En la tabla 4.6 se muestra un análisis de los factores de preprocesamiento que influyen en las métricas micro F1 y macro F1 de la clasificación final. Estos factores son la implementación de *Easy Data Augementation* (EDA), las expresiones regulares (*regex*)utilizadas que son 1) *regex* que incluye caracteres espaciales, 2) *regex* de caracteres alfanuméricos y 3) *regex* letras del alfabeto y la alteración de componentes gramaticales que son 1) no alterar la gramática, 2) la corrección de faltas de ortografías y 3) la corrección de faltas de ortografías más remoción de acentos gráficos (RAG). Los resultados son calculados como el promedio de las métricas micro F1 y macro F1 obtenidas en todos los modelos que incluyen el factor correspondiente.

Tabla 4.6. Análisis métrica F1 de acuerdo a factores de preprocesamiento.

Factor	Rakel		CC		MLP	
	m F1	M F1	m F1	M F1	m F1	M F1
con EDA	0.7087	**0.6215**	0.7165	**0.6301**	0.7412	**0.6591**
sin EDA	**0.7212**	0.6124	**0.7335**	0.6290	**0.7481**	0.6568
regex Especial	0.7140	0.6125	0.7245	0.6299	0.7441	0.6562
regex Alfanumérico	**0.7160**	**0.6205**	0.7234	**0.6312**	0.7438	**0.6603**
regex Letras	0.7148	0.6179	**0.7271**	0.6276	**0.7460**	0.6575
sin Corrección	0.7117	0.6093	0.7232	0.6273	0.7432	0.6576
Corrección	0.7149	0.6205	0.7243	**0.6311**	0.7425	0.6538
Corrección + RAG	**0.7182**	**0.6210**	**0.7274**	0.6303	**0.7482**	**0.6625**

Tabla 4.7. Análisis métrica Precisión de acuerdo a factores de preprocesamiento.

Factor	Rakel		CC		MLP	
	m Prec	M Prec	m Prec	M Prec	m Prec	M Prec
con EDA	0.7445	0.6435	0.7384	0.6329	0.7567	0.6876
sin EDA	**0.7791**	**0.6841**	**0.7830**	**0.6926**	**0.7824**	**0.7320**
regex Especial	0.7566	0.6547	0.7587	0.6611	**0.7722**	**0.7116**
regex Alfanumérico	0.7639	**0.6691**	**0.7619**	**0.6677**	0.7646	0.7094
regex Letras	**0.7649**	0.6676	0.7616	0.6596	0.7717	0.7085
sin Corrección	0.7565	0.6570	0.7592	0.6617	0.7644	0.7063
Corrección	**0.7666**	0.6716	0.7610	**0.6652**	0.7703	0.7078
Corrección + RAG	0.7623	**0.6628**	**0.7621**	0.6615	**0.7738**	**0.7154**

Tabla 4.8. Análisis métrica *Recall* de acuerdo a factores de preprocesamiento.

Factor	Rakel		CC		MLP	
	m Recall	M Recall	m Recall	M Recall	m Recall	M Recall
con EDA	**0.6764**	**0.6127**	**0.6959**	**0.6426**	**0.7269**	**0.6440**
sin EDA	0.6715	0.5653	0.6899	0.5865	0.7169	0.6123
regex Especial	**0.6764**	0.5872	**0.6938**	**0.6154**	0.7184	0.6235
regex Alfanumérico	0.6742	**0.5915**	0.6889	0.6140	**0.7251**	**0.6336**
regex Letras	0.6712	0.5884	0.6961	0.6143	0.7221	0.6274
sin Corrección	0.6726	0.5807	0.6911	0.6111	0.7241	0.6311
Corrección	0.6701	0.5895	0.6915	0.6156	0.7168	0.6222
Corrección + RAG	**0.6792**	**0.5968**	**0.6961**	**0.6170**	**0.7247**	**0.6312**

Los resultados indican que el mejor clasificador, determinado por la métrica micro F1, es el basado en redes neuronales, obteniendo mejores resultados en todas las métricas de evaluación.

En general la implementación de EDA disminuye el rendimiento de clasificación de acuerdo a la métrica micro F1 y aumenta el rendimiento en la métrica macro F1. EDA también disminuye consistentemente la precisión de los clasificadores y aumenta el *recall*. Esto confirma que el desbalance de comentarios por categoría en el set de datos afecta la representación y clasificación de estos y eventualmente tener más ejemplos etiquetados con las categorías con menor proporción en el set de datos podría mejorar el rendimiento de clasificación.

Con respecto a cómo afecta la *tokenización* del set de datos en la clasificación, el uso de expresiones regulares no tiene un efecto significativo. Por otro lado, el efecto de la corrección incrementa los resultados de las métricas en la clasificación final, a diferencia de no corregir, y eleva la precisión de los modelos.

En la tabla 4.9 se muestra un análisis de la métricas F1, Precisión y *Recall* por cada categoría ordenados por rendimiento en puntaje F1, obtenidos del mejor clasificador de acuerdo a la métrica micro F1, que es el basado en redes neuronales con prepocesamiento **m-rsw-l-c-rag**. Se observa que en general las categorías que tienen menos ejemplos etiquetados en el set de datos tienen peor clasificación. A pesar de que las categorías de eje de negocio “Facilidad-Positivo” y “Otros-Negativo” no son las que tienen menos ejemplos etiquetados, la ambigüedad en su definición definición podría explicar su mal resultado en la clasificación.

Tabla 4.9. Resultados Clasificación por Categoría

Categoría	F1	Precisión	Recall	Ejemplos Test
Precio-Positivo	0.845 ±(0.028)	0.864 ±(0.044)	0.828 ±(0.030)	110.2
Atención-Positivo	0.831 ±(0.017)	0.834 ±(0.024)	0.829 ±(0.027)	341.0
Atención-Negativo	0.773 ±(0.020)	0.790 ±(0.022)	0.758 ±(0.045)	258.6
Al Pagar-Negativo	0.771 ±(0.018)	0.793 ±(0.024)	0.752 ±(0.038)	189.6
Producto-Positivo	0.767 ±(0.029)	0.798 ±(0.042)	0.740±(0.039)	152.4
Tienda Física-Positivo	0.754 ±(0.036)	0.813 ±(0.082)	0.710 ±(0.046)	68.2
Tienda Física-Negativo	0.703 ±(0.051)	0.755 ±(0.064)	0.658 ±(0.045)	73.2
Al Pagar-Positivo	0.696 ±(0.057)	0.775 ±(0.045)	0.643 ±(0.095)	109.2
Puntos Cencosud-Negativo	0.687 ±(0.119)	0.739 ±(0.112)	0.655 ±(0.156)	11.0
Producto-Negativo	0.673 ±(0.045)	0.763 ±(0.085)	0.612 ±(0.072)	91.2
Tarjeta Cencosud-Negativo	0.639 ±(0.136)	0.670 ±(0.141)	0.612 ±(0.172)	17.0
Despacho-Negativo	0.547 ±(0.086)	0.681 ±(0.104)	0.458 ±(0.073)	12.2
Facilidad-Positivo	0.543 ±(0.075)	0.689 ±(0.088)	0.464 ±(0.112)	46.2
Otros-Negativo	0.534 ±(0.068)	0.696 ±(0.101)	0.441 ±(0.076)	32.2
Precio-Negativo	0.529 ±(0.079)	0.593 ±(0.048)	0.484 ±(0.106)	51.2
SAC-Negativo	0.477 ±(0.072)	0.592 ±(0.118)	0.412 ±(0.091)	17.0
SAC-Positivo	0.461 ±(0.082)	0.620 ±(0.177)	0.374 ±(0.051)	11.8

4.2. Clasificación de texto con BERT

En esta etapa se utilizan los modelos "BERT Av. pool", "BERT CLS pool" y "BERT Av. pool excepto CLS" propuestos en la sección 3.3. Los modelos se diferencian en la forma de extraer el *output* de BERT antes de ser ingresado al clasificador. Estas son 1) extraer solo el vector correspondiente al *token* `[CLS]` como lo sugiere el trabajo original, 2) promediar los vectores de la capa de *output* y 3) promediar todos los vectores de la capa de *output* excepto el correspondiente al *token* `[CLS]`, respectivamente. El clasificador *multi-label* consiste en una red neuronal de una capa oculta con *dropout* con probabilidad $p = 0,15$ y función de activación sigmoide. Se realizó un ajuste de los parámetros de BERT entrenando el modelo de manera supervisada utilizando los comentarios etiquetados que conforman del set de datos de entrenamiento.

Se probó entrenar el modelo con tamaño de lote (*batch size*) igual a 16 y 24, tasa de aprendizaje igual a 6×10^{-6}, 1×10^{-5}, 4×10^{-5} y 1×10^{-4}. La función de pérdida fue

Focal Loss, con parámetros $\alpha = 0.5$ y $\gamma = 2$, los mismos que utilizados en el *baseline*. A pesar de que el trabajo original los autores realizan la etapa de *fine-tuning* durante 2 a 4 épocas y con tasa de aprendizaje 2×10^{-5} a 5×10^{-5}, se observó que el modelo lograba obtener mejores resultados de clasificación con un entrenamiento de 40 épocas y con tasas de aprendizajes 6×10^{-6} y 1×10^{-5}. Es probable que los modelos entrenados en este trabajo convergieron más lento debido a que se eligió un clasificador basado en una red neuronal con una capa oculta, lo que implica entrenar más parámetros desde cero en comparación con un clasificador sin ninguna capa oculta.

Se definieron 5 particiones del set de datos usando el algoritmo de estratificación iterativa, y se realizaron 5 entrenamientos de cada modelo, cada vez con un set de datos de entrenamiento y *test* distintos. Se obtuvo mejores resultados utilizando tamaño de lote 24 y tasa de aprendizaje 6×10^{-6}. Los resultados se exponen en la tabla 4.10 ordenados de acuerdo a métrica micro F1, y se hace la comparación con el mejor modelo *Multi Layer Perceptron Classifier* de la sección anterior que es el de preprocesamiento "m-rsw-l-c-rag".

Tabla 4.10. Resultados BERT

Modelo BERT	m F1	M F1	m Prec.	M Prec.	m Rec.	M Rec.	SS Acc
BERT Av. pool	0.8204	0.7374	0.8400	0.7853	0.8018	0.7070	0.6413
BERT CLS pool	0.8195	0.7364	0.8425	0.7867	0.7980	0.7042	0.6451
BERT Av. pool excepto CLS	0.8167	0.7317	0.8373	0.7829	0.7973	0.6995	0.6374
FastText m-rsw-l-c-rag	0.7534	0.6617	0.7880	0.7381	0.7217	0.6176	0.5571

Los resultados dependiendo del tipo de extracción de vectores de la capa superior de BERT son similares, y se observa que no extraer el vector asociado al *token* `[CLS]` para realizar la clasificación *multi-label* es levemente peor. Esto concuerda con la recomendación de los autores de utilizar el vector `[CLS]` para clasificar el texto. Se puede observar que todas las métricas de rendimiento del modelo son considerablemente superiores a las obtenidas utilizando FastText presentados en la sección anterior.

En la figura 4.1 se muestra la evolución del valor de la métrica micro F1 determinada por la clasificación del set de *test* en las épocas de entrenamiento 8, 16, 24, 32 y 40 durante el entrenamiento del modelo "BERT Av. pool". En la figura 4.1 cada color representa una partición distinta del método de validación cruzada. Luego de realizar 40 épocas de entrenamiento, la desviación estándar para la métrica micro F1 es de 0.0103, donde el mejor modelo obtiene puntaje 0.8426 y el peor 0.8154.

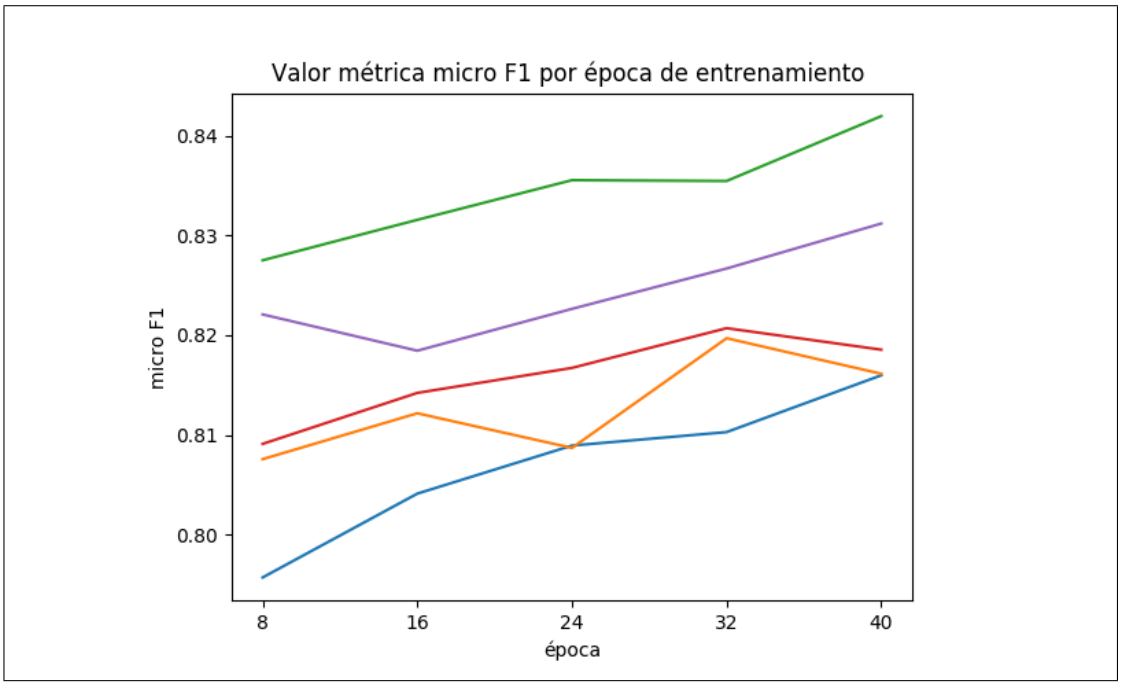


Figura 4.1. Valor métrica micro F1 por época de entrenamiento.

En cuanto al tiempo de entrenamiento, entrenar un modelo de clasificación con FastText demora menos de 2 minutos en un computador MacBook Pro 2017 con 2,9 GHz Intel Core i7. El proceso de *fine-tuning* del modelo BERT con 4.320 datos de entrenamiento se realizó en un dispositivo especializado en entrenamiento de redes neuronales, una GPU Nvidia Geforce GTX Titan X de 12GB GDDR5 y arquitectura 384 Bit, y su tiempo de entrenamiento demora aproximadamente 30 minutos. En comparación con FastText, el entrenamiento de BERT utilizando la CPU del computador MacBook Pro 2017 demora aproximadamente 22 horas, al menos 660 veces el tiempo de entrenamiento de FastText.

Se evaluó el efecto que tiene el no uso de la corrección de palabras sobre el set de datos, y el efecto que tiene usar la técnica de aumento de datos EDA descrita en la sección anterior utilizando modelo de acuerdo a la métrica F1. En este experimento se utilizó el modelo de clasificación BERT de acuerdo a la métrica F1. En la tabla 4.11 se muestran los resultados, que indican que la corrección manual de palabras que se realizó sobre el set de datos tiene un impacto positivo en la clasificación de texto, y que por otro lado la implementación de EDA disminuye el rendimiento del modelo de acuerdo a todas la métricas de evaluación. Esto se produce porque la implementación de EDA genera comentarios que a veces no tienen coherencia o sentido, y muestra que BERT es capaz de generar mejor comprensión del lenguaje que el esquema de representación que utiliza FastText, que es representar una frase como una "bolsa de palabras".

Tabla 4.11. Resultados BERT sin corrección y con EDA

Modelo BERT	m F1	M F1	m Prec.	M Prec.	m Rec.	M Rec.	SS Acc
Av. pool con corrección	0.8426	0.7653	0.8577	0.8023	0.8281	0.7374	0.6806
Av. pool sin corrección	0.8232	0.7371	0.8338	0.7696	0.8128	0.7131	0.6432
Av. pool EDA	0.8180	0.7455	0.8304	0.7789	0.8059	0.7206	0.6441

Se realizó un análisis del rendimiento de clasificación de acuerdo al largo del comentario. Se dividió el set de *test* en 3 particiones de acuerdo a la cantidad de *tokens* de *input* asociados a palabras. Los subsets correspondes a secuencias con menos de 20 *tokens*, de 20 a menos de 50 *tokens*, y de 50 a 200 *tokens*. Las características de cada subset en cuanto a cantidad de *tokens* promedio, palabras del comentario promedio, caracteres promedio y cantidad de etiquetas por comentario promedio se resumen en la tabla 4.12.

Tabla 4.12. Características subsets organizados de acuerdo al largo de la secuencia de tokens de input

Intervalo	comentarios	tokens	palabras	caracteres	etiquetas
$(0, 20)$	477	$11.00 \pm (3.46)$	$7.00 \pm (2.84)$	$40.66 \pm (15.66)$	$1.27 \pm (0.26)$
$[20, 50)$	493	$29.97 \pm (7.45)$	$22.05 \pm (5.94)$	$122.97 \pm (33.77)$	$1.58 \pm (0.28)$
$[50, 200]$	139	$86.55 \pm (37.84)$	$67.09 \pm (30.47)$	$365.24 \pm (158.29)$	$1.77 \pm (0.30)$

Tabla 4.13. Resultados de clasificación de comentarios de acuerdo al largo de la secuencia de tokens de input

Tokens por secuencia	m F1	M F1	m Prec.	M Prec.	m Rec.	M Rec.	SS Acc
$(0, 20)$	0.9002	0.7799	0.8970	0.8094	0.9033	0.7670	0.8031
$[20, 50)$	0.8189	0.7270	0.8231	0.7690	0.8147	0.7112	0.6187
$[50, 200]$	0.7379	0.6333	0.7619	0.6836	0.7154	0.6159	0.4245

Los resultados indican que el modelo tiene peor rendimiento de clasificación en textos largos. Esto ocurre debido a que los clientes incluyen más información sobre su experiencia de compra y por lo tanto los comentarios tienen más etiquetas asociadas y los hace más susceptibles a que se produzcan falsos negativos. En general, en los comentarios largos los clientes escriben más detalles de las situaciones determinantes en su experiencia de compra y con información no determinante para clasificarla. Es probable que por este motivo el rendimiento de clasificación sea peor.

En la selección de datos de entrenamiento se eligió una mayor proporción de comentarios largos respecto de la proporción que estos tienen en el total del set de datos entregados por la empresa, lo que supone que, si el modelo ha logrado una buena generalización sobre la clasificación de comentarios no vistos anteriormente, el rendimiento de clasificación todos los comentarios en el set de datos de 160 mil encuestas será mejor que los obtenidos en este análisis ya que los comentarios cortos se clasifican mejor.

Se exponen los resultados de clasificación por categoría de negocio, los que se muestran en la tabla 4.14. Se observa que clasificación de las categorías "Otros-Negativo" y "Facilidad-Positivo" tienen peor puntaje F1 que en los resultados del *baseline*, y refuerza la suposición de que se pueda deber a la ambigüedad de su definición.

Tabla 4.14. Resultados Clasificación por Categoría BERT

Categoría	F1	Precisión	Recall	Ejemplos Test
Precio-Positivo	$0.912 \pm (0.012)$	$0.910 \pm (0.010)$	$0.915 \pm (0.020)$	110.2
Atención-Positivo	$0.902 \pm (0.010)$	$0.909 \pm (0.017)$	$0.896 \pm (0.022)$	341.0
Producto-Positivo	$0.871 \pm (0.016)$	$0.883 \pm (0.017)$	$0.860 \pm (0.024)$	152.4
Tienda Física-Positivo	$0.857 \pm (0.040)$	$0.891 \pm (0.047)$	$0.827 \pm (0.066)$	68.2
Atención-Negativo	$0.852 \pm (0.013)$	$0.872 \pm (0.043)$	$0.836 \pm (0.038)$	258.6
Al Pagar-Negativo	$0.822 \pm (0.007)$	$0.804 \pm (0.012)$	$0.841 \pm (0.018)$	189.6
Al Pagar-Positivo	$0.807 \pm (0.037)$	$0.832 \pm (0.077)$	$0.790 \pm (0.049)$	109.2
Puntos Cencosud-Negativo	$0.805 \pm (0.039)$	$0.803 \pm (0.096)$	$0.818 \pm (0.064)$	11.0
Producto-Negativo	$0.785 \pm (0.027)$	$0.837 \pm (0.045)$	$0.741 \pm (0.043)$	91.2
Tienda Física-Negativo	$0.776 \pm (0.033)$	$0.830 \pm (0.062)$	$0.732 \pm (0.052)$	73.2
Tarjeta Cencosud-Negativo	$0.742 \pm (0.059)$	$0.735 \pm (0.092)$	$0.753 \pm (0.026)$	17.0
Despacho-Negativo	$0.700 \pm (0.071)$	$0.784 \pm (0.144)$	$0.640 \pm (0.039)$	12.2
SAC-Positivo	$0.700 \pm (0.121)$	$0.778 \pm (0.118)$	$0.645 \pm (0.146)$	11.8
Precio-Negativo	$0.625 \pm (0.029)$	$0.661 \pm (0.064)$	$0.597 \pm (0.048)$	51.2
Otros-Negativo	$0.567 \pm (0.058)$	$0.694 \pm (0.094)$	$0.484 \pm (0.062)$	32.2
Facilidad-Positivo	$0.544 \pm (0.043)$	$0.590 \pm (0.049)$	$0.507 \pm (0.055)$	46.2
SAC-Negativo	$0.448 \pm (0.168)$	$0.561 \pm (0.056)$	$0.412 \pm (0.224)$	17.0

Se comparan los resultados de clasificación obtenidos en este trabajo con los presentados en el trabajo DocBERT Adhikari et al. (2019), que implementa BERT para clasificación *multi-label* de texto. Los autores añaden una capa *fully-connected* sobre BERT, a diferencia de este trabajo en que se añaden 2, y adaptan la función objetivo de manera que se minimiza la función de pérdida *Binary Cross-Entropy Loss*. En este trabajo se utilizó Focal Loss, que corresponde a una extensión de la mencionada.

En DocBERT se implementa la clasificación sobre 4 sets de datos públicos, que corresponden a Reuters-21578, un set de datos de noticias, *arXiv Academic Paper*

dataset (AAPD), un set de datos de artículos académicos clasificados por tópicos, críticas de IMDB, un sitio de críticas a películas y críticas en Yelp 2014, un sitio de pedidos de comida. En la tabla 4.15 se caracterizan los sets de datos utilizados en DocBERT en cuanto a la cantidad de etiquetas definidas, ejemplos etiquetados (muestras) y el promedio de palabras por comentario. Además se añade la caracterización del set de datos utilizado en este trabajo (Encuestas).

Tabla 4.15. Caracterización sets de datos usados en DocBERT

Dataset	Etiquetas	Muestras	Palabras por comentario
Reuters	90	10.789	144,3
AAPD	54	55.840	167,3
IMBD	10	135.669	393,8
Yelp 2014	5	1.125.386	148,8
Encuestas	17	5.402	21,72

Se observa que el set de datos utilizado en este trabajo consta con menos muestras disponibles para el entrenamiento, y el promedio de palabras por comentario también es menor. A continuación, en la tabla 4.16 se muestran los resultados obtenidos utilizando $BERT_{BASE}$ para clasificación *multi-label* en DocBERT en cuanto a la métrica micro F1 en el set de validación, y se comparan con los resultados obtenidos en este trabajo.

Tabla 4.16. Comparación de resultados de DocBERT y este trabajo

Set de datos	micro F1
Reuters	0.923
AAPD	0.766
IMBD	0.560
Yelp 2014	0.726
Encuestas	0.820

Se observa que existe una gran variación del resultado de la clasificación, estos dependen fuertemente de las características del set de datos con el que se trabaja. Los

resultados obtenidos en este trabajo están dentro del rango de resultados obtenidos en DocBERT.

4.3. Resumen de Texto

A continuación se describe la implementación, resultados y análisis de los experimentos descritos en la sección 3.4. El clasificador utilizado para clasificar los textos es "BERT Av. pooling" descrito en la sección anterior, y la forma de preprocesar el texto también es la misma.

En el experimento "all-comments-k-means" se clasifican todos los comentarios del set de datos para agruparlos de acuerdo cada categoría (<eje de negocio>, <polaridad>). Antes se realizó un filtro para eliminar del set de datos todos los comentarios que contienen una sola palabra, debido a que se observó que estos son poco informativos para ser seleccionados en el resumen final. En total el set de datos redujo su tamaño a 159.518 observaciones, las que se clasificaron con el clasificador mencionado y se agruparon de acuerdo a las 17 etiquetas correspondientes. Las observaciones por agrupación varían en el rango desde 140 hasta 56.987. Luego de implementar K-means independientemente en cada uno de los 17 grupos formados y seleccionar k de acuerdo al método de la silueta se obtuvieron 89 centroides en total. En seguida se vectorizaron las "frases" correspondientes a los comentarios del set de *test* y a cada una se le asignó un puntaje correspondiente a la distancia con el centroide más cercano y se ordenaron ascendentemente en una lista. La lista se filtra dejando a lo más una frase asociada a cada centroide y por cada periodo de tiempo y local se seleccionó la cantidad de frases deseada para construir el resumen de acuerdo a la lista ordenada.

En el experimento dos, "all-comments-split-k-means", se realiza el mismo procedimiento, pero en lugar de clasificar los comentarios, se clasificaran las frases que conforman los comentarios. En total el set de datos contiene 220.631 frases. Las observaciones por agrupación de acuerdo a cada una de las 17 etiquetas varían en el

rango desde 113 hasta 61.792 y se conformaron 58 centroides después de la implementación de K-means en cada agrupación. La conformación del resumen se hizo de la misma manera que en el experimento anterior.

En el tercer y cuarto experimento, "period-k-means" y "period-clf-k-means", consiste en implementar el algoritmo K-means sobre los vectores que representan las frases de los periodos de tiempo y local correspondientes a un ejemplo del set de *test*, eligiendo k como la cantidad de frases que se desea que contenga el resumen. Se elige la frase más cercana a cada centroide de los *clusters* creados. La única diferencia entre los dos es que a los vectores que representan las frases en "period-clf-k-means" se les concatenó la clasificación de la frase.

La métrica de evaluación ROUGE-N se refiere a la exhaustividad (*recall*), sin embargo también se presenta la precisión y el puntaje F1 obtenidos. Los resultados de la métricas de evaluación de los experimentos y de los resúmenes generados al azar, denominados como "random", se exponen a continuación

Tabla 4.17. ROUGUE-1

	F1	Precisión	Recall
all-comments-k-means	0.4841	0.7176	**0.3870**
all-comments-split-k-means	**0.4876**	**0.7308**	0.3847
period-k-means	0.4616	0.7200	0.3544
period-clf-k-means	0.4657	0.7143	0.3599
random	0.4196	0.6287	0.3291

Tabla 4.18. ROUGUE-2

	F1	Precisión	Recall
all-comments-k-means	0.2603	0.3939	0.2043
all-comments-split-k-means	**0.2691**	**0.4150**	**0.2085**
period-k-means	0.2405	0.3808	0.1823
period-clf-k-means	0.2413	0.3766	0.1840
random	0.1936	0.2953	0.1496

Tabla 4.19. ROUGUE-3

	F1	Precisión	Recall
all-comments-k-means	0.1849	0.2810	0.1445
all-comments-split-k-means	**0.1917**	**0.2978**	**0.1476**
period-k-means	0.1680	0.2678	0.1270
period-clf-k-means	0.1724	0.2709	0.1309
random	0.1364	0.2099	0.1045

Tabla 4.20. ROUGUE-4

	F1	Precisión	Recall
all-comments-k-means	0.1447	0.2210	0.1128
all-comments-split-k-means	**0.1491**	**0.2330**	**0.1143**
period-k-means	0.1278	0.2039	0.0965
period-clf-k-means	0.1334	0.2102	0.1010
random	0.1066	0.1647	0.0814

Se observa que los experimentos en que se ejecuta K-means sobre todos los comentarios del set de datos tienen mejores resultados que los que se ejecuta K-means sobre los comentarios del periodo de tiempo y local. Se distingue que "all-comments-k-means" y "all-comments-split-k-means" tienen resultados similares, que son mejores que los otros dos experimentos propuestos y que todos los experimentos son mejores que el *baseline* "random".

La cantidad de *clusters* o centroides determinados en "all-comments-split-k-means" son menos que los determinados "all-comments-k-means", a pesar de que la cantidad de datos en el primero es mayor. Se presume que debido a que el clasificador en el experimento "all-comments-split-k-means" se ejecuta sobre las frases y no comentarios, estas son textos de menor longitud y tratan sobre menos temas, por lo tanto se le asignan menos etiquetas y luego K-means puede agruparlas en *clusters* de mejor manera.

Ahora se procede a comparar los resultados con los del trabajo relacionado con resumen de textos academicos D. Miller (2019) y los de auditorías Chou et al. (2019). En el experimento "period-k-means" se replicó el procedimiento de generación de resúmenes expuesto en D. Miller (2019). Los otros 3 experimentos definidos en este trabajo superaron el rendimiento de "period-k-means", probablemente porque incorporan información proveniente la clasificación que tiene cada comentario o frase, y esto mejora la calidad del resumen generado. En D. Miller (2019) no se exponen resultados en cuanto a métricas de evaluación por lo que no se pueden comparar los resultados numéricamente.

Por otro lado en Chou et al. (2019), los autores no exponen los resultados de las métricas de evaluación ROUGE, pero mencionan que la precisión y el *recall* son aproximadamente 50% cada uno. En este trabajo la precisión de ROUGE-1 supera el 70% y el *recall* tiene un valor de aproximadamente 38%. Es difícil comparar los resultados de ambos trabajos porque depende fuertemente de la forma en que se escribieron los resúmenes "ideales".

Los autores en Chou et al. (2019) se enfocan en una evaluación humana que indica, que en la mayoría de los casos los resúmenes automáticos eran funcionales e incluso algunos óptimos. Estos informan de múltiples hechos, que resultan en un texto poco coherente, en general presentan la información relevante pero esta difusa entre varias frases no deseadas y algunas veces la información está redactada de forma indirecta, por ejemplo, si se deseaba que explícitamente dijera "el beneficiario no cumplió porque la acción no se realizó", el resumen expresaba "es recomendable que el beneficiario haga la acción para que cumpla".

A continuación se analizará la calidad de un resumen generado por el modelo "all-comments-split-k-means" de los comentarios recibidos en el periodo de tiempo y local simbolizado por "P502-82". Se recibieron 33 comentarios en P502-82, por lo que, de acuerdo a la heurística presentada en 3.4, el resumen generado contiene 10 de las frases presentes en los comentarios, y es el siguiente.

"la tienda estaba limpia y ordenada. por la muy buena atencion y amabilidad con que fui atendido. muy buena poca variedad. nunca hay un vendedor disponible. fue muy amable y encontro lo que necesitaba. pero hay pocos vendedores y productos exibidos que a la hora de comprarlos se demoran mucho en buscarlos y finalmente no hay. porque no hice cola al pagar pero tampoco nadie atiende porque hay pocas vendedoras en general. la variedad de ropa y zapatos es muy limitada. amabilidad de las vendedoras y cajeras rapidez en la experiencia de compra. me probe y elegi con tranquilidad."

Los 33 comentarios preprocesados correspondientes al periodo "P502-82" se encuentran en el anexo B. Se observa que el resumen generado es poco coherente, y se conforma como una lista de hechos puntuales. Con respecto a la redacción se presentan los mismos problemas que en Chou et al. (2019), sin embargo se logra entender los hechos que determinaron las experiencias de compra de los clientes. Se aprecia que debido a la separación de los comentarios en frases, estas a veces pierden el contexto del que tratan, o el sujeto de la oración.

Se evalúa la cantidad de hechos y percepciones relevantes obtenidos en un resumen generado, sobre los hechos y percepciones relevantes que sucedieron en "P502-82". Aunque este proceso es subjetivo, se determinó contar como relevante cualquier percepción o hecho que trate sobre un eje de negocio definido. En la tabla 4.21 se expone la cantidad de hechos y percepciones relevantes detectadas en los comentarios recibidos en "P502-82", y se comparan con las cantidad de hechos y percepciones descritas en el resumen.

Tabla 4.21. Hechos y percepciones relevantes reales y detectadas por resumen automático

Eje-Polaridad	Percepciones reales	Percepciones resumen
Atención-Positivo	15	4
Atención-Negativo	12	3
Producto-Positivo	1	1
Producto-Negativo	4	3
Al Pagar-Positivo	4	2
Al Pagar-Negativo	3	0
Otros-Negativo	1	0
Facilidad-Positivo	2	1
Precio-Positivo	1	0
Tienda Física-Positivo	6	2

Se observa que la cantidad de hechos y percepciones expresadas en el resumen generado automáticamente es proporcional a las descritas en el set de comentarios correspondientes a “P502-82”, excepto respecto del eje de negocio y polaridad “Al Pagar-Negativo”. Se puede decir que este resumen cumple su función de resumir gran parte de la información recibida guardando proporciones respecto a los hechos recibidos en los comentarios del periodo. Esta evaluación se realizó sobre solo 1 periodo de tiempo y local seleccionado al azar, ya no es posible realizarla automáticamente, y habría que analizar más muestras para extender la robustez de este resultado.

En este trabajo, el enfoque del resumen es recopilar los hechos y percepciones más relevantes sobre los comentarios recibidos en un periodo de tiempo y local, sin embargo esta metodología podría ser extendida en un trabajo futuro para determinar un resumen orientado a recopilar información sobre un periodo de tiempo, local, eje de negocio, percepciones negativas u otra necesidad que pueda tener la empresa.

5. ANÁLISIS DEL NEGOCIO

En este capítulo se realizará un análisis cuantitativo y cualitativo de la clasificación y resumen de comentarios respecto a su utilidad para el negocio. Se hará una comparación de los índices de satisfacción del negocio con la polaridad de la clasificación de los comentarios para analizar la calidad de la clasificación y se mostrará la utilidad de la clasificación y resumen de comentarios para obtener inteligencia de negocio.

A continuación se exponen ejemplos seleccionados al azar de la clasificación de comentarios a los que se les asignó la categoría “Al Pagar-Negativo”, “Al Pagar-Positivo”, “Atención-Negativo” y “Atención-Positivo” obtenida a partir del uso del clasificador BERT sobre el set de datos entregados por la empresa. Se comparan las etiquetas asignadas con las etiquetas reales. Cuando la clasificación es correcta se marca como “Ok”.

Tabla 5.1. Ejemplos comentarios clasificados con categoría Al Pagar-Negativo

#	Comentario preprocesado	Etiquetas asignadas	Etiquetas reales
1	todos los vendedores o promotores viendo sus celulares constantemente luego en la caja un unico cajero que demoro demasiado	Al Pagar-Negativo; Atención-Negativo;	Ok
2	habia cajera en el piso aunq no encontre lo que buscaba compre algo que estaba en oferta	Al Pagar-Negativo; Producto-Negativo;	Al Pagar-Positivo; Producto-Negativo; Precio-Positivo;
3	la fila para pagar era eterna y no habia nadie que te ayudara con informacion de los productos	Al Pagar-Negativo; Al Pagar-Positivo; Atención-Negativo;	Al Pagar-Negativo; Atención-Negativo;
4	lento el tema de la caja imagino por la hora casi al cierre pocas funcionando	Al Pagar-Negativo;	Ok
5	fue facil encontrar lo que queria y aunque habia unas cajas muy llenas encontre una vacia y pude pagar rapidamente	Al Pagar-Negativo; Producto-Positivo; Tienda Física-Positivo;	Al Pagar-Negativo; Al Pagar-Positivo;
6	pocas y lentas las cajas	Al Pagar-Negativo;	Ok
7	me dejaron puesta la alarma en uno de los productos que compre me di cuenta en el primer piso en la seccion juvenil dama y me acerque al cajero que estaba ahi trabajando para que me sacara la alarma del producto con boleta en mano el me dijo que no podia sacarla que fuera al piso donde compre lo que me parecio una persona muy falta de respeto y con muy poca voluntad ademas no tuvo consideracion de que que anda con mi bebe de meses y un coche	Al Pagar-Negativo;	Ok
8	la cajera no parecia muy concentrada atendia a la gente sin respetar el orden de la fila	Al Pagar-Negativo;	Ok
9	tienen los productos pero lento el pago en caja	Al Pagar-Negativo; Producto-Positivo;	Ok
10	quise pagar y habia cajera y varias personas antes que yo esperando	Al Pagar-Negativo;	Ok

Tabla 5.2. Ejemplos comentarios clasificados con categoría Al Pagar-Positivo

#	Comentario preprocesado	Etiquetas asignadas	Etiquetas reales
1	la atencion excelente especificamente al momento de pagar la senorita que me atendio en la caja excelente los felicito por el personal igual el joven que me atendio en zapateria muy buena disposicion sobre todo porque fue en un horario que no habla casi vendedores al menos conmigo y otra clienta el nos atendio excelente	Al Pagar-Positivo; Atención-Positivo;	Ok
2	el pago fue muy rapido y los zapatos los agarre de la gondola	Al Pagar-Positivo;	OK
3	cajeras con buen trato amables buena disposicion	Al Pagar-Positivo; Atención-Positivo;	Al Pagar-Positivo;
4	estaba todo ordenado y al pasar por caja fue muy rapido	Al Pagar-Positivo; Tienda Física-Positivo;	Ok
5	excelente atencion del cajero el producto lo elegi sin ayuda	Al Pagar-Positivo;	Ok
6	la vendedora y cajera que me atendieron fueron muy atentas y el trato fue de lo mejor	Al Pagar-Positivo; Atención-Positivo;	Ok
7	mas que nada debido a la fecha navidena cajas con mucha gente	Al Pagar-Negativo; Al Pagar-Positivo;	Al Pagar-Negativo;
8	me atendio muy rapido el cajero	Al Pagar-Positivo;	Ok
9	expedito al llegar a cajas buena atencion de los vendedores	Al Pagar-Positivo; Facilidad-Positivo;	Al Pagar-Positivo; Atención-Positivo;
10	todo muy ordenado encontre lo que buscaba la cajera muy amable	Al Pagar-Positivo; Tienda Física-Positivo;	Al Pagar-Positivo; Tienda Física-Positivo; Producto-Positivo

Tabla 5.3. Ejemplos comentarios clasificados con categoría Atención-Negativo

#	Comentario preprocesado	Etiquetas asignadas	Etiquetas reales
1	nadie atendio pero igual encontre lo que buscaba	Atención-Negativo;	Atención-Negativo; Producto-Positivo
2	la variedad de productos es buena pero hay muy pocos venderdores cajas y la atencion en caja no es cordial	Al Pagar-Negativo; Atención-Negativo; Producto-Positivo;	Ok
3	al preguntar por el consulta precio a dos funcionarias no fueron capaz de darme una buena respuesta y de buena forma ambas tenian una actitud molesta	Atención-Negativo;	Ok
4	las vendedoras no atienden bien	Atención-Negativo;	Ok
5	poco personal tuve que esperar demasiado para que me atendieran	Atención-Negativo;	Ok
6	la persona que atendia la caja es una persona mal educada y nada de amable	Al Pagar-Negativo; Atención-Negativo;	Al Pagar-Negativo;
7	mala atencion muy lentas las cajas	Al Pagar-Negativo; Atención-Negativo;	Ok
8	los vendedores poco disponibilidad para buscar productos y mala atencion	Atención-Negativo;	Ok
9	porque no hay nadie que te atienda	Atención-Negativo;	Ok
10	porque no fui atendido tuve que perseguir al trabajador para que me atendiera	Atención-Negativo;	Ok

Tabla 5.4. Ejemplos comentarios clasificados con categoría Atención-Positivo

#	Comentario preprocesado	Etiquetas asignadas	Etiquetas reales
1	la atencion es muy buena la gran mayoria de mis cosas las he comprado en paris y nunca he temido ningun problema	Atención-Positivo;	Ok
2	por que la persona que me atendio me dio una sugerencia muy buena	Atención-Positivo;	Ok
3	por que no he tenido mayores dificultades en mis compras y la atencion es buena	Atención-Positivo;	Ok
4	buena atencion	Atención-Positivo;	Ok
5	porque son atentos al momento de efectuar una compra y siempre tengo una mejor opcion al momento que me atienden	Atención-Positivo;	Ok
6	por que nos ayudaron a buscar lo que necesitamos	Atención-Positivo;	Ok
7	rapida atencion cajera amable	Al Pagar-Positivo; Atención-Positivo;	Al Pagar-Positivo;
8	bien atendido	Atención-Positivo;	Ok
9	el lugar muy limpio y las vendedoras amables y con conocimiento con lo que uno le preguntaba muy bien atendida	Atención-Positivo; Tienda Física-Positivo;	Ok
10	excelente atencion y rapida	Atención-Positivo;	Ok

En los ejemplos mostrados se observa que el 75% de los ejemplos fueron clasificados con exactitud. La métrica *Subset Accuracy* de clasificación con BERT del capítulo anterior es 64% aproximadamente y el aumento de la exactitud de clasificación de esta muestra seleccionada al azar probablemente se debe a que el clasificador tiene mejor rendimiento empleado en textos cortos.

Se filtró el set de datos entregados por la empresa para obtener datos emitidos durante un periodo de un año, y luego se procedió a clasificarlos, obteniendo un total de 111.087 comentarios clasificados. De acuerdo a la metodología de cálculo del nivel de satisfacción CSAT, la evaluación de los clientes sobre un eje de negocio se realiza en una escala de 1 a 5. Los clientes que evaluaron con nota 1 o 2 se consideran detractores y los clientes que evalúan con nota 4 o 5 se consideran promotores. De acuerdo a esto se convirtió el puntaje CSAT en -1 si el cliente evaluó con nota 1 o 2 y a 1 si el cliente evaluó con nota 4 o 5. Análogamente se convirtió la polaridad de la clasificación de los comentarios en la misma escala de acuerdo a si es negativa o positiva.

Se agruparon los comentarios por cada eje de negocio, de tal manera que el cliente sea promotor o detractor de ese eje, es decir que haya puesto nota 1, 2, 4 o 5, y se graficó el índice CSAT transformado a -1 o 1 y la clasificación correspondiente de ese comentario respecto del eje de negocio transformado a -1 o 1 dependiendo de su polaridad. En el gráfico 5.1 se ilustra la comparación.

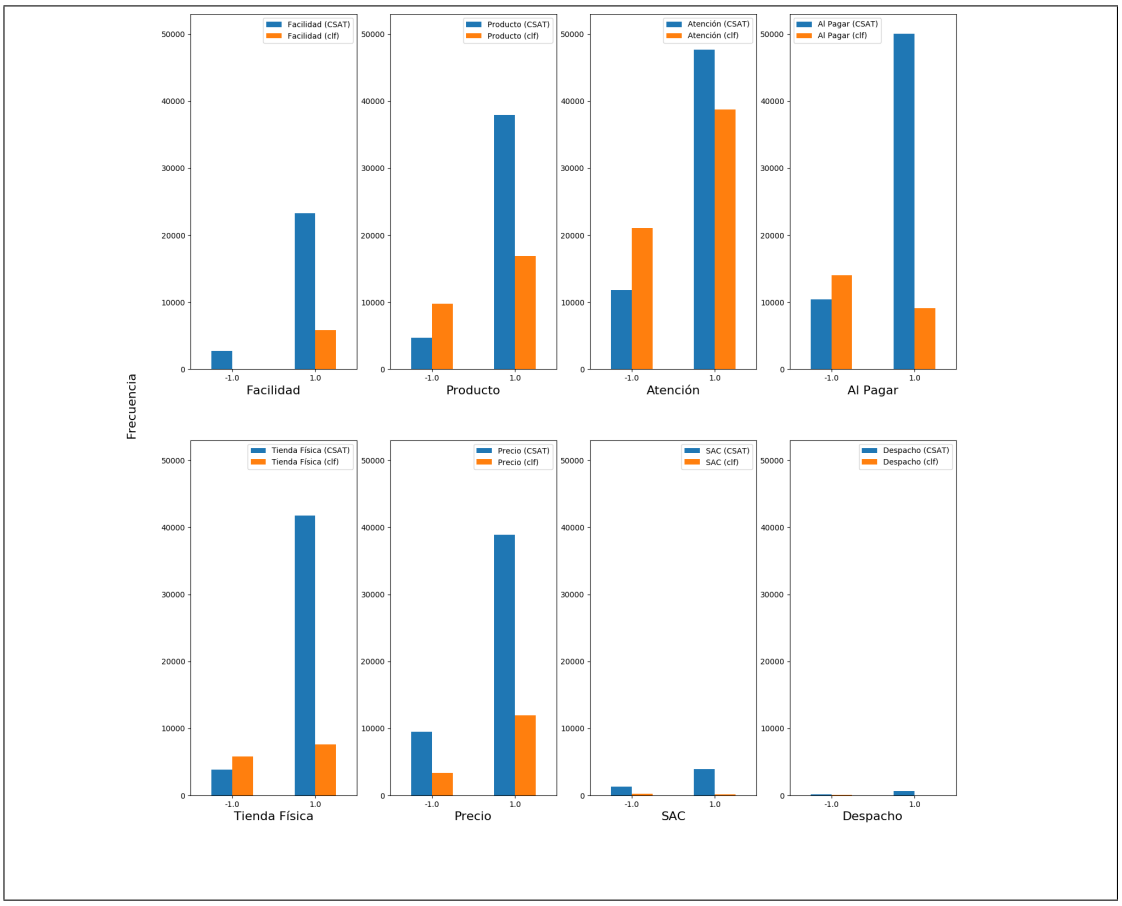


Figura 5.1. Comparación índice CSAT y clasificación de comentarios por eje de negocio

Se observa que los clientes promotores y detractores de las categorías “Servicio al cliente” y “Despacho” son pocos en comparación al resto debido a que en la encuesta de satisfacción se le pide evaluar la experiencia con respecto a esos ejes solo si corresponde, por lo que no se considerarán esas categorías en este análisis. En 4 de los 5 ejes en que se clasificaron comentarios con polaridad negativa se observa mayor

cantidad de comentarios negativos que la cantidad de detractores de acuerdo al indicador CSAT, y en contraste, en todos los ejes de negocio se observa que la cantidad de comentarios positivos son menos que la cantidad de evaluaciones que son promotoras. De acuerdo a esto, se puede observar que cuando los clientes comentan más sobre los aspectos negativos de su experiencia en comparación con los aspectos aspectos positivos.

Luego se filtró el set de datos de acuerdo a la clasificación por eje de negocio y polaridad de comentarios, y se comparó el índice CSAT de acuerdo a si son promotores o detractores. El gráfico se ve en la figura 5.2. Se observa nuevamente que dado que un cliente comentó negativamente sobre un eje de negocio, su nota puesta en el indicador CSAT no es necesariamente 1 o 2, y que dado que un cliente comentó positivamente sobre un eje de negocio, su nota propuesta sí es 4 o 5, siendo promotor del eje de negocio. La nota 3 se muestra como “nulo”.

Finalmente se realizó una comparación sobre la densidad de comentarios clasificados obtenidos por mes en el año, en que se observa que los clientes comentan más sobre la atención recibida por parte del personal.

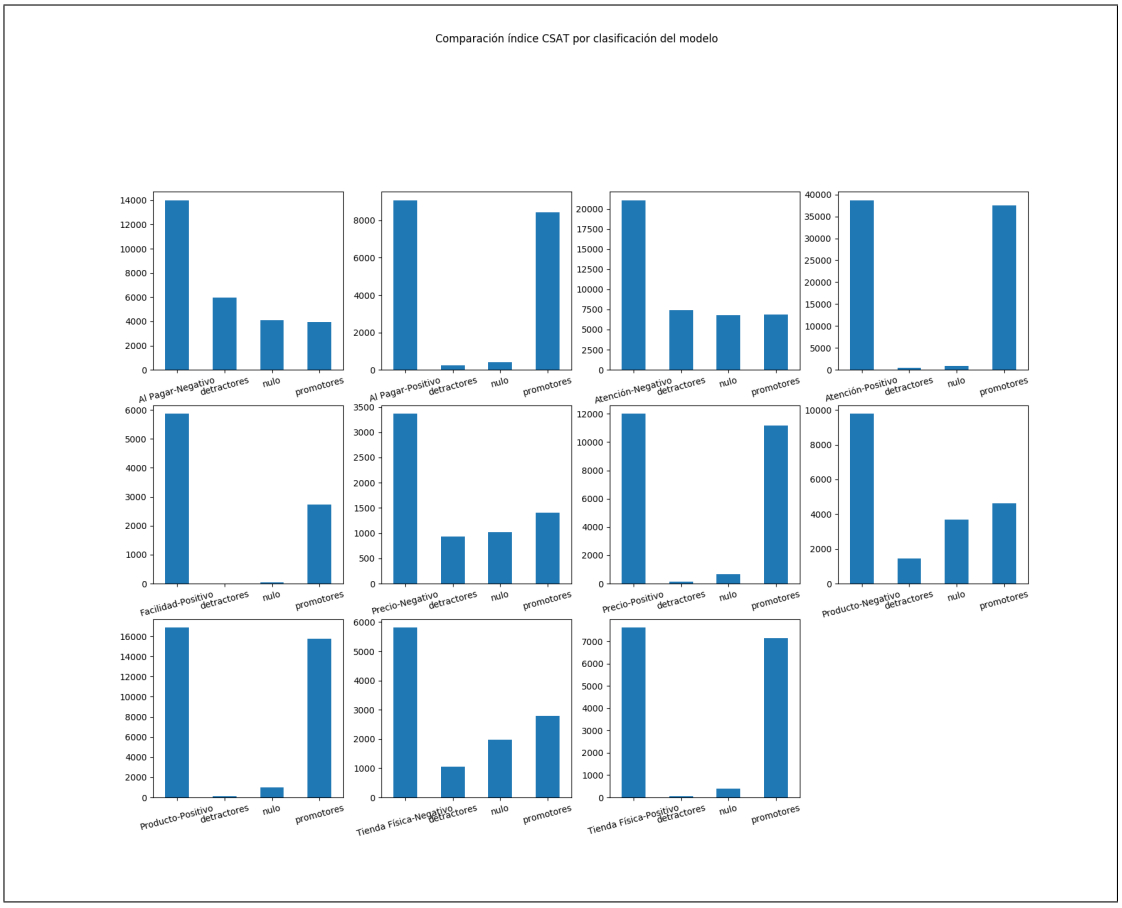


Figura 5.2. Índice CSAT vs clasificación de comentarios

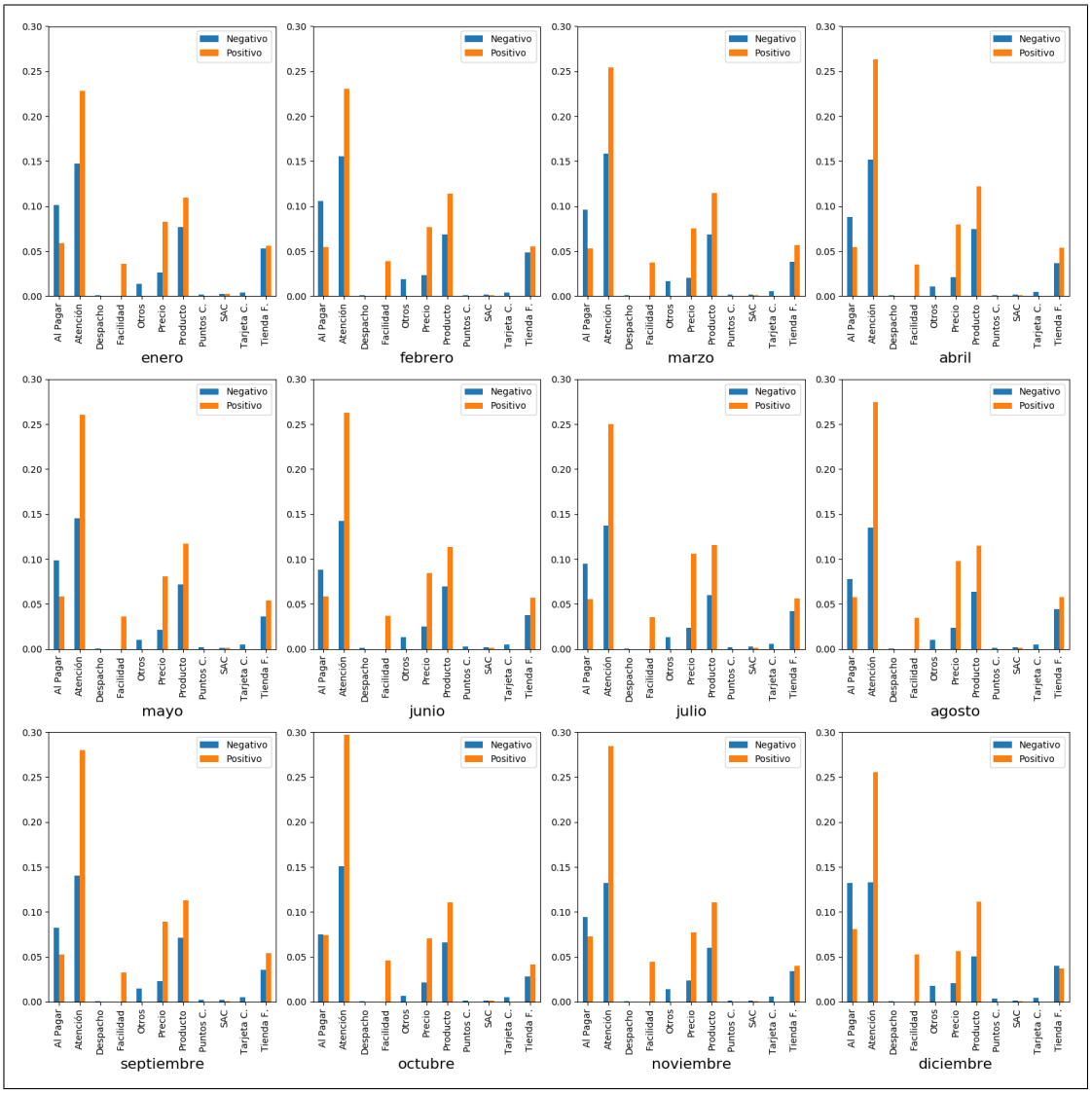


Figura 5.3. Polaridad de clasificación de comentarios por eje de negocio por mes

6. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se logró aplicar los recientes avances en campo de investigación de Procesamiento de Lenguaje Natural sobre una encuesta de satisfacción realizada por una empresa de la industria del retail para obtener nueva información de utilidad a partir del texto que escriben los clientes al contestar la encuesta.

Se experimentó utilizando el modelo lenguaje distribuido FastText cuya velocidad de entrenamiento facilita la realización de experimentos preliminares. Con respecto al preprocesamiento de texto se identificó que la corrección de ortografía de las palabras es relevante y afecta la modelación del lenguaje y rendimiento de clasificación de texto. También se encontró que el proceso de *tokenización* no afecta de mayor manera al rendimiento del modelo clasificador, sin embargo remover los acentos gráficos, e incluir solo caracteres alfabéticos es levemente mejor que los otros métodos.

Modelando el lenguaje con FastText, el aumento de datos artificiales de entrenamiento mejora el *recall* de la clasificación y disminuye la precisión del modelo, y en contraste al utilizarlo con BERT este disminuye su rendimiento de clasificación ya que genera frases poco coherentes.

Se analizaron diversos clasificadores *multi-label* en que se comprobó que el basado en redes neuronales obtuvo mejor rendimiento de clasificación que los otros métodos de aprendizaje de máquinas tradicionales. Además se comprobó la mejora sustancial en el Entendimiento del Lenguaje Natural que logra el modelo de propósito general BERT en comparación con el previo modelo de lenguaje FastText

Respecto de los resultados de la clasificación de texto, se encontró que los textos cortos se clasifican de mejor manera en comparación a los más largos, ya que en los últimos los clientes tienden a describir con mucho detalle su experiencia de compra o describir situaciones que no son relevantes para la empresa y por lo tanto se hace más difícil la clasificación.

Se analizaron variaciones de un método de resumen extractivo de texto realizando una *clusterización* de estos y se observó que un resumen generado automáticamente mantiene la proporción de hechos que tratan sobre diferentes ejes de negocio respecto del total de comentarios que fueron resumidos. Se comprobó que el método propuesto es mejor que seleccionar comentarios al azar para crear un resumen y que la incorporación de información obtenida a partir de la clasificación de comentarios mejora la cantidad de información relevante contenida en cada resumen. Los resúmenes de textos se realizaron en base a la agrupación de comentarios por un determinado periodo de tiempo y local, pero el método propuesto se puede extender para realizar resúmenes de acuerdo a otros tipos de agrupaciones.

De acuerdo a los resultados obtenidos por la clasificación de todo el set de comentarios entregados por la empresa se observa que los clientes en general escriben más sobre aspectos negativos de su experiencia de compra que sobre los positivos, que concuerda con la percepción de los encargados de revisar la encuesta de satisfacción de la empresa.

El clasificador es capaz de detectar percepciones negativas sobre el negocio que el indicador CSAT no detecta. En este trabajo se ha expuesto una base para construir una herramienta de análisis de texto que sirva para recopilar información relevante de inteligencia de negocio.

Con respecto al trabajo futuro se consideran aspectos que podrían mejorar el rendimiento de clasificación de comentarios. Primero se recomienda aumentar el set de ejemplos de entrenamiento ya que se observó que en categorías con menos comentarios etiquetados existe peor rendimiento de clasificación. Se propone crear una interfaz gráfica que facilite el proceso de etiquetamiento manual en que el usuario deba marcar como correctos ejemplos bien clasificados con el modelo actual y corregir los que no sean correctos. Esto alivianaría la tarea de leer y pensar qué etiquetas corresponden a un comentario y facilitaría tener un set de entrenamiento de mayor tamaño para reentrenar el modelo clasificador.

Segundo, se mostró que corregir faltas ortográficas y gramaticales aumenta el rendimiento de clasificación y se sugiere robustecer el método propuesto de corrección de palabras revisando las faltas ortográficas más comunes. Como trabajo futuro se propone corregir errores gramaticales a través de un modelo traductor de máquinas que sea capaz de traducir oraciones con faltas de ortografía y modismos a oraciones escritas en un lenguaje formal.

Tercero, se recomienda evaluar el impacto del lenguaje con modismos chilenos en el rendimiento de clasificación comparando el resultado obtenido de clasificar textos con lenguaje coloquial y textos escritos en español con estándar internacional. Si resulta que esto afecta el rendimiento de clasificación se debe evaluar la posibilidad de preentrenar BERT usando un *corpus* con español chileno obtenido a partir de la recopilación de textos de la encuesta de satisfacción, críticas a productos, comentarios en redes sociales, entre otros. El objetivo sería que el modelo BERT pueda aprender el uso y significado léxico y semántico de palabras coloquiales que probablemente no hayan estado en el set de entrenamiento del modelo original.

Finalmente, con la finalidad de crear una herramienta funcional para obtener inteligencia de negocio se sugiere tratar de diferente manera los comentarios dependiendo del largo de la respuesta. En textos cortos el clasificador es más exacto, sin embargo los textos largos en general incluyen información muy detallada que podría ser de valor para la empresa. De acuerdo a las necesidades, los métodos de clasificación y resumen de texto se pueden incorporar en un *software* que genere indicadores claves a partir del texto que complementen la información de los indicadores tradicionales NPS y CSAT.

BIBLIOGRAFÍA

Adhikari, A., Ram, A., Tang, R., & Lin, J. (2019). Docbert: BERT for document classification. *CoRR*, *abs/1904.08398*. Retrieved from http://arxiv.org/abs/1904.08398

Barrus, T. (2018, Jul). *pyspellchecker.* Retrieved from https://github.com/barrust/pyspellchecker

Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003, March). A neural probabilistic language model. *J. Mach. Learn. Res.*, *3*, 1137–1155. Retrieved from http://dl.acm.org/citation.cfm?id=944919.944966

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *J. Mach. Learn. Res.*, *3*, 993–1022. Retrieved from http://portal.acm.org/citation.cfm?id=944937 doi: http://dx.doi.org/10.1162/jmlr.2003.3.4-5.993

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Buscema, M. (1998, 02). Back propagation neural networks. , *33*, 233-70. doi: 10.3109/10826089809115863

Cachero-Martínez, S., & Vázquez, R. (2017). La experiencia de compra como creadora de lealtad actitudinal: ¿qué papel juega el compromiso con el detallista? *XXIX Congreso Internacional de Marketing AEMARK 2017*.

Chou, V., Kent, L., Góngora, J., Ballerini, S., & Hoover, C. (2019, 11). *Towards automatic extractive text summarization of a-133 single audit reports with machine learning.*

Dessel, G. V. (2014, November). *Measure customer satisfaction: Csat, ces and nps compared.* Retrieved from https://www.checkmarket.com/blog/csat-ces-nps-compared/

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Domínguez, G. (2019). *Diccionario de la rae en modo texto plano.* Retrieved from https://www.giusseppe.net/blog/archivo/2015/10/29/diccionario-de-la-rae-en-modo-texto-plano/

Evgeniou, T., & Pontil, M. (2001, 01). Support vector machines: Theory and applications. In (Vol. 2049, p. 249-257). doi: 10.1007/3-540-44673-7_12

fasttext.cc. (n.d.).

Gentile, C., Spiller, N., & Noci, G. (2007). How to sustain the customer experience:. *European Management Journal*, *25*(5), 395–410. doi: 10.1016/j.emj.2007.08.005

Glorot, X., Bordes, A., & Bengio, Y. (2011, 11–13 Apr). Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, & M. Dudík (Eds.), *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (Vol. 15, pp. 315–323). Fort Lauderdale, FL, USA: PMLR. Retrieved from http://proceedings.mlr.press/v15/glorot11a.html

Gutmann, M. U., & Hyvärinen, A. (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, *13*(Feb), 307–361.

Harris, Z. (1954). Distributional structure. *Word*, *10*(23), 146–162.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, *abs/1512.03385*. Retrieved from http://arxiv.org/abs/1512.03385

Honnibal, M., & Montani, I. (2017). *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.* (To appear)

Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. *ICLR 2015*. Retrieved from https://arxiv.org/abs/1412.6980

Lei Ba, J., Ryan Kiros, J., & E. Hinton, G. (2016, 07). Layer normalization.

Levenshtein, V. I. (1966, February). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, *10*, 707.

Lin, T., Goyal, P., Girshick, R. B., He, K., & Dollár, P. (2017). Focal loss for dense object detection. *CoRR*, *abs/1708.02002*. Retrieved from http://arxiv.org/abs/1708.02002

Loper, E., & Bird, S. (2002). Nltk: The natural language toolkit. In *In proceedings of the acl workshop on effective tools and methodologies for teaching natural language processing and computational linguistics. philadelphia: Association for computational linguistics.*

Loshchilov, I., & Hutter, F. (2017). Fixing weight decay regularization in adam. *CoRR*, *abs/1711.05101*. Retrieved from http://arxiv.org/abs/1711.05101

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *CoRR*, *abs/1310.4546*. Retrieved from http://arxiv.org/abs/1310.4546

Mikolov, T., Yih, W.-t., & Zweig, G. (2013, June). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north American chapter of the association for computational linguistics: Human language technologies* (pp. 746–751). Atlanta, Georgia: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/N13-1090

Miller, D. (2019). Leveraging bert for extractive text summarization on lectures. *CoRR*, *abs/1906.04165*. Retrieved from http://arxiv.org/abs/1906.04165

Miller, G. A. (1995, November). Wordnet: A lexical database for english. *Commun. ACM*, *38*(11), 39–41. Retrieved from http://doi.acm.org/10.1145/219717.219748 doi: 10.1145/219717.219748

Nam, J., Kim, J., Gurevych, I., & Fürnkranz, J. (2013). Large-scale multi-label text classification - revisiting neural networks. *CoRR*, *abs/1312.5419*. Retrieved from http://arxiv.org/abs/1312.5419

Nangia, N., & Bowman, S. R. (2019). Human vs. muppet: A conservative estimate of human performance on the GLUE benchmark. *CoRR*, *abs/1905.10425*. Retrieved from http://arxiv.org/abs/1905.10425

Olea, I. (2013, Dec). *Lemarios y listas de palabras del español.* Retrieved from https://github.com/olea/lemarios/

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dÁlché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Rajasekaran, A., & Varalakshmi, R. (2018, 06). Review on automatic text summarization. *International Journal of Engineering and Technology(UAE)*, *7*, 456-460. doi: 10.14419/ijet.v7i2.33.14210

Ramos, J. (1999). *Using tf-idf to determine word relevance in document queries.*

Read, J., Pfahringer, B., Holmes, G., & Frank, E. (2009, 08). Classifier chains for multi-label classification. In (Vol. 85, p. 254-269). doi: 10.1007/978-3-642-04174-7_17

Reddi, S. J., Kale, S., & Kumar, S. (2018). On the convergence of adam and beyond. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=ryQu7f-RZ

Rong, X. (2014). word2vec parameter learning explained. *CoRR*, *abs/1411.2738*. Retrieved from http://arxiv.org/abs/1411.2738

Sechidis, K., Tsoumakas, G., & Vlahavas, I. (2011). On the stratification of multi-label data. In *Machine learning and knowledge discovery in databases.*

Sosa, A. (2019, Aug). *Verbos en español y sus conjugaciones.* Retrieved from https://github.com/asosab/esp_verbos/

Tsoumakas, G., & Vlahavas, I. (2007). Random k-labelsets: An ensemble method for multilabel classification. *Machine Learning: ECML 2007 Lecture Notes in Computer Science*, 406–417. doi: 10.1007/978-3-540-74958-5_38

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. *CoRR*, *abs/1706.03762*. Retrieved from http://arxiv.org/abs/1706.03762

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, *abs/1804.07461*. Retrieved from http://arxiv.org/abs/1804.07461

Wei, J. W., & Zou, K. (2019). EDA: easy data augmentation techniques for boosting performance on text classification tasks. *CoRR*, *abs/1901.11196*. Retrieved from http://arxiv.org/abs/1901.11196

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... Rush, A. M. (2019, October). HuggingFace's Transformers: State-of-the-art Natural Language Processing. *arXiv e-prints*, arXiv:1910.03771.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., ... Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, *abs/1609.08144*. Retrieved from http://arxiv.org/abs/1609.08144

Zhang, M.-L., & Zhou, Z.-H. (2006). Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, *18*(10), 1338–1351. doi: 10.1109/tkde.2006.162

ANEXO

A. ALGORITMOS

Algoritmo 4: Iterative Stratification

Input : Un se de instancias D, asociadas a un set de labels $L = \{\lambda_1, ..., \lambda_q, \}$, el número de subsets deseados k, la proporción de ejemplos deseados en cada subset $r_1, ..., r_k$

Output: Subsets disjuntos $S_1, ..., S_k$ de D

para $j \leftarrow 1$ **a** k **hacer**
  $c_j \leftarrow |D| r_j$ // Calcular el número de ejemplos deseados en cada subset
// Calcular el número de ejemplos deseados de cada label en cada subset
para $i \leftarrow 1$ **a** $|L|$ **hacer**
  // Encuentra el número de ejemplos de cada label en el set inicial
  $D^i \leftarrow \{(\boldsymbol{x}, Y) \in D : \lambda_i \in Y\}$
  para $j \leftarrow 1$ **a** k **hacer**
    $c_j^i \leftarrow |D^i| r_j$
mientras $|D| > 0$ **hacer**
  // Encontrar el label con menor cantidad de ejemplos restantes (al menos 1)
  // Rompe empates al azar
  $D^i \leftarrow \{(\boldsymbol{x}, Y) \in D : \lambda_i \in Y\}$
  $l \leftarrow \arg\min_i (|D^i|) \cap \{i : D^i \neq \emptyset\}$
  para cada $(\boldsymbol{x}, Y) \in D^l$ **hacer**
    // Encuentra el (los) subset(s) con el mayor número de ejemplos deseados
    // para este label, rompiendo empates considerando el mayor número
    // de ejemplos deseados, rompiendo futuros empates al azar
    $M \leftarrow \arg\max_{j=1...k} (c_j^l)$
    si $|M| = 1$ **entonces**
      $m \in M$
    en otro caso
      $M' \leftarrow \arg\max_{j \in M} (c_j)$
      si $|M'| = 1$ **entonces**
        $m \in M'$
      en otro caso
        $m \leftarrow randomElementOf(M')$
    $S_m \leftarrow S_m \cup \{(\boldsymbol{x}, Y)\}$
    $D \leftarrow D \setminus \{(\boldsymbol{x}, Y)\} s$
    // Actualizar el número deseado de ejemplos **para cada** $\lambda_i \in Y$ **hacer**
      $c_m^i \leftarrow c_m^i - 1$
    $c_m \leftarrow c_m - 1$
devolver $S_1, ..., S_k$

B. COMENTARIOS PREPROCESADOS DEL PERIODO P502-82

“porque me atendieron me ayudaron a elegir y las vendedoras no andaban con el celular en la mano como en otras tiendas. buena atencion. la variedad de ropa y zapatos es muy limitada . harto stock pero todo de lo mismo. me probe y elegi con tranquilidad. me atendi sola y alpagar fui muy bien atendida. la vendedora fue muy amable y con muy buen trato. porque a la hora que fui a zapateria hay una persona que atiende y bien pero es poca. falta que los vendedores conversen menos entre si y esten mas atentos a los clientes. muy buena atencion . muy amable el personal . me ofrecieron ofertas buenas . muy satisfecha con mis compras. me atendi solo. muy gentil la atencion. rapido. muy buena poca variedad. falto un poco de ayuda de vendedora. si. la tienda estaba limpia y ordenada , pero hay pocos vendedores y productos exibidos que a la hora de comprarlos se demoran mucho en buscarlos y finalmente no hay. porque aunque estaban los productos me atendi sola. por la muy buena atencion y amabilidad con que fui atendido. no. resolvieron el tema que tenia , aunque en un principio la actitud no fue la mejor . pero todo bien al final. era un cambio de talla de un regalo para una familiar mayor . el tipo que me atendio ni siquiera me pregunto si era para regalo o no , no le saco los precios a las prendas , tampoco me informo que uno de los productos estaba en promocion 2 x1 , yo lo averigue , lo unico que le preocupo a el , durante todo el rato , era saber si yo andaba con bolsa reutilizable o no. amabilidad de las vendedoras y cajeras rapidez en la experiencia de compra. porque no hice cola al pagar pero tampoco nadie atiende porque hay pocas vendedoras en general. nunca hay un vendedor disponible . jamas . y las que estan argumentan no puedo atenderla porque estoy a cargo de los probadores yo no soy de esa marca habria que buscar a alguien yo no soy de esta seccion . es muy desagradable. buena porque al fin y al cabo uno se atiende sola ! solo necesita ir a la caja a cancelar . nunca estan los vendedores para hacerle alguna consulta. la vendedora de urban decay , fue muy amable y encontro lo que necesitaba. excelente atencion. solo compre . nada que decidir. porque esta ordenada . la verdad es que pague una compra para mi esposa . la atencion en caja y en probadores parece ser buena . que es la unica atencion que me ha tocado vivir tanto en paris como en otras tiendas retail . todo es self service. tienda ordenada y poca espera en caja. mucho personal viendo usando el celular. fui bien atendida y los probadores estaban disponibles pero al momento de pagar me tomo mas tiempo ya que solo habian dos cajas abiertas y

mucha gente que queria comprar. atencion expedita , probadores disponibles , productos ordenados”