Nota de Aceptación

Aprobado por el Comité de Trabajo de Grado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana para optar el título de Ingeniero de Sistemas y Computación.

Dr. Camilo Rocha
Decano de la Facultad de Ingeniería

ING. GERARDO MAURICIO SARRIA
Director Carrera Ingeniería Sistemas y Computación.

ING. JUAN PABLO GARCIA
Codirector Trabajo

ING. JULIAN GIL
Codirector Trabajo

ING. Luisa Fernanda Rincon
Jurado 1

ING. Mario Julian Mora Cardona
Jurado 2

Pontificia Universidad JAVERIANA Cali | Facultad de Ingeniería y Ciencias

Acta de Correcciones al Proyecto de Grado
Ingeniería de Sistemas

Fecha: 15 de Julio de 2024

Autor: Santiago Collantes Zuluaga

Nombre del Proyecto de Grado: Desarrollo de un Modelo de Aprendizaje Automático para la Asignación de Códigos de Producto por Sociedades Comisionistas de la Bolsa Mercantil de Colombia a partir de Descripciones de Productos en Supermercados.

Directores: Juan Pablo García Cifuentes, MBA y Dr. Julián Gil Gonzalez

Como indica el artículo 2.27 de las Directrices de Trabajo de Grado, he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Grado definieron que se efectuarán, como consta en el Acta de Calificación correspondiente.

Juan Pablo García Cifuentes

Julián Gil González

Desarrollo de un Modelo de Aprendizaje Automático para la Asignación de Códigos de Producto por Sociedades Comisionistas de la Bolsa Mercantil de Colombia a partir de Descripciones de Productos en Supermercados

Autor:
Santiago Collantes Zuluaga

Profesores:
Juan Pablo Garcia Cifuentes
Julian Gil Gonzalez

Pontificia Universidad Javeriana.

Cali, Valle del Cauca

15 de julio de 2024

Santiago de Cali, 24 de mayo del 2024.

Señores
Pontificia Universidad Javeriana Cali.
Dr. Gerardo Mauricio Sarria
Director Carrera de Ingeniería de Sistemas y Computación.
Cali.

Cordial Saludo.

Por medio de la presente nos permitimos informarle que el estudiante de Ingeniería de Sistemas y Computación Santiago Collantes Zuluaga (cod: 8944278) culminó satisfactoriamente bajo nuestra dirección el proyecto de grado titulado "Desarrollo de un Modelo de Aprendizaje Automático para la Asignación de Códigos de Producto por Sociedades Comisionistas de la Bolsa Mercantil de Colombia a partir de Descripciones de Productos en Supermercados".

Atentamente,

Juan Pablo García Cifuentes

Julian Gil Gonzalez

Santiago de Cali, 24 de Mayo del 2024.

Señores
Pontificia Universidad Javeriana Cali.
Dr. Gerardo Mauricio Sarria
Director Carrera de Ingeniería de Sistemas y Computación.
Cali.

Cordial Saludo.

Me permito presentar a su consideración el proyecto de grado titulado "Desarrollo de un Modelo de Aprendizaje Automático para la Asignación de Códigos de Producto por Sociedades Comisionistas de la Bolsa Mercantil de Colombia a partir de Descripciones de Productos en Supermercados" con el fin de cumplir con los requisitos exigidos por la Universidad para optar al título de Ingeniero de Sistemas y Computación.

Al firmar aquí, doy fe que entiendo y conozco las directrices para la presentación de trabajos de grado de la Facultad de Ingeniería aprobadas el 26 de Noviembre de 2009, donde se establecen los plazos y normas para el desarrollo del anteproyecto y del trabajo de grado.

Atentamente,

Santiago Collantes Zuluaga

Santiago Collantes Zuluaga
Código: 8944278

Abstract

This project presents the development of a machine learning model for the automatic assignment of product codes in the Colombian Mercantile Exchange (BMC) based on descriptions provided by its Brokerage Firms (SC). Using advanced natural language processing (NLP) and deep learning techniques, the aim is to improve the accuracy and efficiency of the current manual assignment process.

The implemented techniques include word embeddings with Word2Vec, pre-trained spaCy models, the combination of similarity metrics such as Jaccard and cosine, and Siamese networks (SBERT) for semantic sentence comparison. Throughout the project, several limitations were identified in the data corpus, such as the variability and quality of the descriptions, orthographic and lexical errors, and the lack of additional metadata.

The results showed that pre-trained models and Siamese networks provided significant improvements in assignment accuracy compared to models trained solely on our corpus. The combination of similarity metrics also proved effective in improving description alignment. The study concludes that the integration of additional data and continuous fine-tuning of the models can lead to further improvements in accuracy and efficiency.

Keywords: machine learning, natural language processing, word embeddings, Siamese networks, SBERT, Colombian Mercantile Exchange, product code assignment.

Resumen

Este proyecto presenta el desarrollo de un modelo de aprendizaje automático para la asignación automática de códigos de productos en la Bolsa Mercantil de Colombia (BMC) a partir de descripciones proporcionadas por sus Sociedades Comisionistas (SC). Utilizando técnicas avanzadas de procesamiento de lenguaje natural (NLP) y aprendizaje profundo, se busca mejorar la precisión y eficiencia del proceso actual de asignación manual.

Las técnicas implementadas incluyen word embeddings con Word2Vec, modelos preentrenados de spaCy, la combinación de métricas de similitud como Jaccard y coseno, y redes siamesas (SBERT) para la comparación semántica de oraciones. A lo largo del proyecto, se identificaron varias limitaciones en el corpus de datos, como la variabilidad y calidad de las descripciones, errores ortográficos y léxicos, y la falta de metadatos adicionales.

Los resultados mostraron que los modelos preentrenados y las redes siamesas proporcionaron mejoras significativas en la precisión de la asignación en comparación con los modelos entrenados únicamente con nuestro corpus. La combinación de métricas de similitud también demostró ser efectiva para mejorar el alineamiento de descripciones. El estudio concluye que la integración de datos adicionales y el ajuste fino continuo de los modelos pueden llevar a mejoras adicionales en la precisión y eficiencia de este proyecto.

Palabras clave: aprendizaje automático, procesamiento de lenguaje natural, word embeddings, redes siamesas, SBERT, Bolsa Mercantil de Colombia, asignación de códigos de productos.

Índice general

Capítulo 1

Análisis

1.1. Introducción

En la era actual de la digitalización y el avance tecnológico, la capacidad de procesar grandes volúmenes de datos y extraer información relevante se ha convertido en una ventaja competitiva crucial para diversas industrias. Particularmente en el sector financiero, donde las transacciones y operaciones se cuentan por millones y se generan a diario, la eficiencia y precisión en la gestión de la información es esencial. La Bolsa Mercantil de Colombia (BMC), una entidad vital en el sistema financiero colombiano, no es ajena a estos desafíos. Las Sociedades Comisionistas (SC) que operan bajo la BMC enfrentan una tarea laboriosa y crítica: la asignación manual de códigos de productos estandarizados basados en las descripciones proporcionadas por supermercados. Este proceso, además de ser propenso a errores, conlleva un elevado costo en términos de tiempo y recursos humanos.

Dentro de este contexto, surge la necesidad de innovar y buscar soluciones tecnológicas que permitan automatizar y optimizar este proceso. La inteligencia artificial, y en particular el aprendizaje automático, ha demostrado ser una herramienta valiosa en el procesamiento y clasificación de datos en múltiples campos. Así, este proyecto de grado propone el desarrollo de modelos basados en aprendizaje automático para la asignación automática de códigos de productos, apoyándose en técnicas de procesamiento de lenguaje natural (NLP) como lo son los *word embeddings* y *transformers* para interpretar y mapear las descripciones proporcionadas por los supermercados.

Esta investigación no solo busca mejorar la eficiencia operativa y reducir los costos para las Sociedades Comisionistas, sino que también aspira a sentar un precedente sobre cómo las tecnologías emergentes pueden ser aplicadas en el sector financiero colombiano, potenciando su competitividad y adaptabilidad en un mundo cada vez más digitalizado. Con este propósito, el presente anteproyecto delineará los objetivos, la metodología, y las expectativas de este proyecto.

1.2. Definición del problema

1.2.1. Antecedentes

El análisis de similitud de oraciones es una aplicación crítica de NLP, utilizada en una variedad de contextos, desde la comparación de productos hasta la clasificación basada en similitudes. Un caso de estudio [1] utilizó NLP para identificar similitudes entre dos reportajes de noticias, empleando técnicas de vectorización y cálculo de distancia de coseno para medir la similitud. La vectorización convierte los documentos en vectores

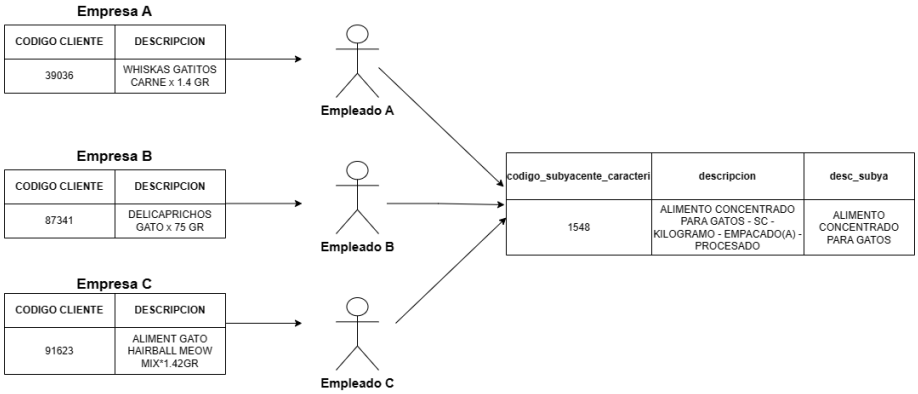

Figura 1.1: Escenario actual.

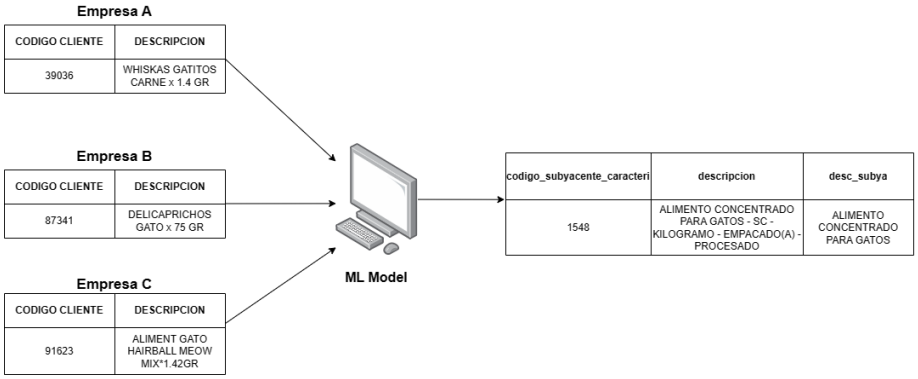

Figura 1.2: Escenario deseado.

numéricos utilizando medidas como la Frecuencia de Término (TF) y la Frecuencia de Documento Inversa (IDF), seguido por el cálculo de similitud de coseno. Herramientas como Word2Vec y Doc2Vec de la biblioteca Gensim de Python se usaron para vectorizar los textos y calcular la similitud. Estos métodos proporcionaron puntuaciones de similitud de alrededor del 72.62 % y 79.06 %, respectivamente.

Sin embargo, los avances recientes en NLP, especialmente en modelos preentrenados como BERT y GPT-3, han ampliado significativamente las capacidades en este campo. Estos modelos han logrado un rendimiento notable en tareas de NLP, impulsando una tendencia hacia el preentrenamiento a gran escala, aunque su implementación en escenarios industriales plantea desafíos específicos, como la necesidad de conocimientos mejorados y la destilación de conocimientos para su eficiente despliegue[2].

Además, la investigación reciente ha demostrado que el uso de embeddings de oraciones basados en BERT y la similitud de texto basada en coseno puede ahorrar mucho tiempo y esfuerzo en la tarea de etiquetado de datos para clasificación de textos. Este enfoque ha logrado una precisión y puntuaciones F1 de 90 % en la clasificación de textos, lo que destaca su eficacia para aplicaciones prácticas[3].

Estudios específicos sobre el etiquetado de textos han demostrado que la combinación de técnicas tradicionales y avanzadas de NLP puede mejorar significativamente los resultados. Por ejemplo, el trabajo de "Ticket-BERT"ha desarrollado un modelo de BERT afinado específicamente para etiquetar incidentes en sistemas de gestión de tickets, logrando un rendimiento superior al de los clasificadores tradicionales al combinar TF-IDF

y BERT con estrategias de aprendizaje activo[4]. Otro estudio exploró el uso de modelos BERT en la detección y clasificación de ciberacoso en redes sociales, combinando BERT con BiLSTM para mejorar la precisión del etiquetado en un entorno desafiante[5].

Por lo tanto, en el contexto de este proyecto, la integración de estas técnicas y herramientas avanzadas en modelos de NLP, como el uso de embeddings de oraciones basados en BERT, permite abordar de manera más efectiva y eficiente la tarea de asignación automatizada de códigos de productos a partir de descripciones. Estas innovaciones representan un paso adelante significativo en la precisión, la eficiencia y la practicidad de los sistemas de NLP aplicados a entornos comerciales y operativos.

1.3. Justificación

El sector financiero, en particular las Sociedades Comisionistas de la Bolsa Mercantil de Colombia, enfrenta constantes desafíos en su operatividad diaria, buscando siempre la eficiencia y precisión en cada uno de sus procesos. Uno de estos desafíos radica específicamente en la asignación manual de códigos de productos estandarizados, un procedimiento que, aunque pueda parecer simple a primera vista, conlleva una carga operativa considerable.

Anecdóticamente, diversos individuos responsables de llevar a cabo esta tarea han manifestado que la conversión de un producto del código interno al código del BMC requiere, en promedio, de 2 a 3 minutos. Esta cifra, que podría parecer insignificante en solitario, adquiere una magnitud preocupante cuando se considera que una sola empresa puede registrar hasta 9000 productos.

Cantidad de productos	**Mejor tiempo (min)**	**Peor tiempo (min)**
1	2	3
2	4	6
3	6	9
...	...	...
8999	17998	26997
9000	18000	27000

Cuadro 1.1: Avance progresivo del tiempo de conversión de productos

Realizando un cálculo sencillo, esto se traduce en aproximadamente 450 horas de trabajo dedicadas exclusivamente a esta actividad. Eso equivale a más de 18 días completos, considerando un esfuerzo continuo las 24 horas, o bien, si tomamos en cuenta una jornada laboral estándar de 8 horas, a cerca de 56 días laborales.

Concepto	**Descripción**
Tiempo promedio de conversión por producto	2 a 3 minutos
Número máximo de productos por empresa	9000 productos
Tiempo total de conversión (mínimo)	18000 minutos (300 horas)
Tiempo total de conversión (máximo)	27000 minutos (450 horas)
Días completos (24 horas continuas)	Más de 18 días
Días laborales (jornada de 8 horas)	Cerca de 56 días

Cuadro 1.2: Resumen del tiempo de conversión de productos

La automatización de este proceso no solo permitiría un significativo ahorro en tiempo, sino también en recursos humanos, reduciendo los posibles errores humanos y garantizando una mayor consistencia y precisión en la asignación de códigos. Además, el

tiempo ahorrado podría ser redirigido hacia actividades de mayor valor agregado para las Sociedades Comisionistas, potenciando su competitividad y eficiencia en el mercado.

Asimismo, en un contexto global donde la digitalización y la innovación tecnológica son imperativos para mantener la competitividad, la incorporación de herramientas basadas en inteligencia artificial y aprendizaje automático se presenta no solo como una oportunidad, sino como una necesidad inminente. Por lo tanto, este anteproyecto no solo busca ofrecer una solución práctica a un problema operativo concreto, sino también sentar las bases para una transformación tecnológica más amplia en el sector financiero colombiano.

1.4. Objetivos de la Tesis

1.4.1. Objetivo general

Desarrollar un modelo de aprendizaje automático avanzado que utilice técnicas de procesamiento de lenguaje natural para la asignación eficiente y precisa de códigos de productos en la Bolsa Mercantil de Colombia, mejorando la precisión, reduciendo los errores humanos y los costos operativos asociados al proceso actual.

1.4.2. Objetivos específicos

1. Preparar un conjunto de datos que contenga descripciones de productos de supermercados y sus correspondientes códigos de la Bolsa Mercantil de Colombia (BMC).

2. Implementar dos modelos de procesamiento de lenguage natural, basados en técnicas de aprendizaje de maquina, para mapear automáticamente las descripciones de productos a los códigos de la BMC y comparar sus resultados.

3. Evaluar los modelos desarrollados utilizando métricas de rendimiento apropiadas, como top-k accuracy, para garantizar su eficacia y precisión en la asignación de códigos.

1.5. Marco Teórico

1.5.1. Bolsa Mercantil de Colombia (BMC)

A palabras oficiales de la BMC [6]La Bolsa Nacional Agropecuaria (BNA) fue establecida en 1979, funcionando como una sociedad de economía mixta. Esta entidad contó con el apoyo de diversas instituciones gubernamentales y privadas, incluyendo el Ministerio de Agricultura y el Idema. En 2010, la BNA cambió su razón social y se convirtió en la Bolsa Mercantil de Colombia (BMC), ofreciendo diversas opciones financieras a través de sus sociedades comisionistas.

Actualmente, la BMC es la Bolsa de Productos y Servicios de Colombia, que busca generar valor al país promoviendo mercados eficientes y opciones de financiamiento no bancario. Forma parte del Grupo Bolsa Mercantil de Colombia.

1.5.2. Sociedades Comisionistas (SC)

Las SC son sociedades anónimas que actúan como intermediarias entre compradores y vendedores en la plataforma de la BMC. Están integradas por profesionales especializados y certificados por el Autorregulador del Mercado de Valores en Colombia (AMV) y están supervisadas por la Superintendencia Financiera de Colombia.

1.5.3. Clasificación de productos y su importancia

La correcta clasificación de productos es esencial para la operación eficiente de servicios como el Mercado de Comercialización entre Privados (MERCOP). Una correcta clasificación garantiza que los productos y servicios se ajusten adecuadamente a las necesidades de los clientes. El cuadro 1.3 nos da una idea a de la cantidad de capital que estos servicios proveen en un periodo de 6 meses, con lo cual nos podemos dar una idea de la importancia de su correcto funcionamiento.

Producto	**Valor de Venta (COP)**
ARROZ CASCARA NACIONAL HUMEDO	38.728.900.000
ARROZ CASCARA NACIONAL SECO	29.697.685.320
FIBRA DE ALGODON	19.443.648.693
CAFE EXCELSO	9.916.615.740
MIEL	7.650.000.000
MAIZ AMARILLO NACIONAL SECO	3.072.000.000
CARNE DE CERDO x Lote	2.921.601.800
ARROZ CRISTAL	2.250.000.000
MAIZ BLANCO NACIONAL SECO	1.942.500.000
FRIJOL SOYA NACIONAL	976.830.279
YUCA	877.500.000
SORGO NACIONAL	308.902.230
GULUPA	62.720.000

Cuadro 1.3: Valor de ventas de diferentes productos en el MERCOP (2022-2023)

1.5.4. Natural Lenguage Processing(NLP)

Definición y evolución histórica del NLP

El NLP es una disciplina que se centra en la interacción entre las máquinas y el lenguaje humano, permitiendo a los sistemas inteligentes entender, interpretar y generar lenguaje de manera efectiva [7].

La historia del NLP generalmente comienza en la década de 1950. Desde entonces, ha experimentado una evolución significativa, impulsada por avances en técnicas y tecnologías. La verdadera evolución en el campo del NLP se observó en años posteriores, donde ha mejorado considerablemente en términos de eficacia y aplicabilidad. Esta evolución ha sido posible gracias a la interacción entre diversas disciplinas, como la informática, la ingeniería de la información, la inteligencia artificial y la lingüística [7].

Aplicaciones y relevancia del NLP

El NLP se utiliza en una amplia gama de aplicaciones en tiempo real. Con la incorporación del aprendizaje profundo, se han abierto puertas a diferentes aplicaciones[7], como por ejemplo:

- **Clasificación de Textos:** Esta técnica involucra categorizar y asignar etiquetas predefinidas a documentos, oraciones o frases según su contenido. Su objetivo es determinar automáticamente la clase o categoría a la que pertenece un texto. Es fundamental en NLP, con aplicaciones prácticas como análisis de sentimientos, detección de spam, etiquetado de temas, identificación de idiomas, entre otros. Los algoritmos de clasificación de textos analizan características y patrones en el texto para hacer predicciones precisas sobre su categoría.

- **Similitud de Oraciones:** Esta medida indica cuán similares son dos fragmentos de texto o en qué grado expresan el mismo significado. Las tareas relacionadas incluyen identificación de paráfrasis o duplicados, búsqueda y aplicaciones de coincidencia. Los métodos comunes para la similitud de texto van desde productos punto simples de vectores de palabras hasta clasificación por pares y, más recientemente, redes neuronales profundas.

- **Análisis de Sentimientos** Es una técnica de NLP utilizada para determinar si los datos son positivos, negativos o neutrales. Se realiza a menudo en datos textuales para ayudar a las empresas a monitorear la opinión sobre marcas y productos en comentarios de clientes y entender las necesidades de estos.

Para este proyecto, la similitud de oraciones es particularmente relevante. Es necesario entrenar un modelo para identificar si dos descripciones son similares según el conjunto de datos. Si el modelo determina una alta similitud entre descripciones, se asignará el código de producto correspondiente. Esta tarea es crucial para garantizar la precisión en la asignación de códigos, ya que permite que el modelo reconozca y compare las sutilezas y matices en las descripciones de productos, facilitando una asignación más precisa y eficiente de códigos.

Técnicas y Herramientas en NLP

El campo del Procesamiento de Lenguaje Natural (NLP) se ha beneficiado enormemente de una serie de técnicas avanzadas que han surgido en los últimos años, impulsadas en gran medida por el aprendizaje profundo. A continuación, se describen algunas de las técnicas más prominentes y que son especialmente relevantes para nuestro proyecto:

- **Word Embeddings**: Estos embeddings convierten palabras en vectores numéricos, capturando relaciones semánticas entre palabras. Word2Vec, GloVe y FastText son ejemplos populares que transforman el texto en un espacio vectorial, facilitando diversas tareas de NLP[8].

- **Transformers**: Introducidos en 2017 [11], los transformers han revolucionado el NLP. Modelos como BERT, GPT y T5, que se basan en la atención auto-regresiva, han establecido nuevos estándares en múltiples tareas de NLP, desde traducción hasta generación de texto.

- **Redes Siamesas:**: Una técnica emergente que está ganando relevancia en el campo del NLP son las redes siamesas. Estas redes son especialmente eficaces para comparar la similitud entre dos entradas de texto. Al actualizar una sub-red, se reflejan cambios en la sub-red espejo, permitiendo comparaciones basadas en vectores de características. Esta característica las hace útiles en una variedad de aplicaciones del mundo real, incluyendo la similitud de oraciones, una tarea crucial para nuestro proyecto de asignar códigos a productos basados en descripciones de texto. [13]

Después de las técnicas, es esencial mencionar las herramientas que han facilitado la implementación y aplicación de estas técnicas:

- **Herramientas de Python**: NLTK es una biblioteca fundamental para el procesamiento del lenguaje. TextBlob simplifica tareas comunes de NLP, mientras que PyTorch-NLP y TensorFlow ofrecen capacidades para modelos de aprendizaje profundo.

- **Herramientas de Java**: OpenNLP y Stanford NLP son bibliotecas robustas que ofrecen una amplia gama de funciones de NLP.

- **Herramientas Node**: Retext y NLP.js son ejemplos de bibliotecas de NLP para JavaScript.

El NLP ha pasado de depender en gran medida de reglas escritas a mano a beneficiarse del aprendizaje automático, que permite a los algoritmos aprender patrones y características del lenguaje a partir de grandes conjuntos de datos. Esta evolución ha llevado a aplicaciones de NLP más eficientes y precisas, capaces de abordar la complejidad del lenguaje humano.

Métricas de evaluación

Para evaluar el rendimiento de los modelos desarrollados en este proyecto, se utilizaron dos métricas principales: la similitud coseno y la precisión top-k (top-k accuracy).

- **Similitud Coseno:** La similitud coseno es una métrica ampliamente utilizada para medir la similitud entre dos vectores. En el contexto del procesamiento de lenguaje natural, se emplea para cuantificar la similitud entre descripciones de productos transformadas en vectores. La similitud coseno entre dos vectores $\mathbf{A}$ y $\mathbf{B}$ se define como:

$$\text{sim}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \tag{1.1}$$

  donde $\mathbf{A} \cdot \mathbf{B}$ es el producto punto de los vectores y $\|\mathbf{A}\|$ y $\|\mathbf{B}\|$ son las normas de los vectores $\mathbf{A}$ y $\mathbf{B}$ respectivamente. Esta métrica toma valores entre -1 y 1, donde 1 indica que los vectores son idénticos, 0 indica que son ortogonales, y -1 indica que son diametralmente opuestos[9].

- **Precisión Top-k (Top-k Accuracy):** La precisión top-k es una métrica que evalúa la capacidad de un modelo para generar las k predicciones más probables y que estas incluyan la respuesta correcta. En este proyecto, la precisión top-k se utiliza para medir cuán a menudo la descripción correcta del producto se encuentra entre las k descripciones más similares propuestas por el modelo. La precisión top-k se calcula como:

$$\text{Precisión top-k} = \frac{\text{Número de descripciones correctas en el top-k}}{\text{Número total de descripciones evaluadas}} \tag{1.2}$$

  Esta métrica es especialmente útil cuando se desea evaluar el rendimiento del modelo en un escenario donde no se espera una coincidencia exacta, pero se desea que la respuesta correcta esté dentro de un rango de las mejores predicciones del modelo [10].

1.5.5. Word embeddings y la similitud de oraciones

En el contexto de NLP, los word embeddings desempeñan un papel crucial en la interpretación del significado y la estructura de las palabras. Estas representaciones vectoriales no solo capturan información semántica y sintáctica, sino que también permiten comparar la similitud semántica entre oraciones de manera más refinada y matizada. Esta capacidad es especialmente valiosa en proyectos como el nuestro, donde la comparación precisa de descripciones textuales es fundamental.

Las investigaciones avanzadas en este campo, como el estudio de V. Novotný et al.[14], resaltan el uso de técnicas de regularización y medidas de similitud suave para mejorar la capacidad de los word embeddings en tareas de similitud de oraciones. Estas

técnicas ayudan a afinar los modelos para que capturen mejor las relaciones complejas entre palabras y frases. En nuestro proyecto, esto es esencial para el análisis comparativo de descripciones de productos, donde cada palabra puede tener un impacto significativo en la interpretación del texto.

Para este proyecto, se ha decidido desarrollar y utilizar embeddings pre-entrenados basados en Word2Vec. Esta decisión se basa en la eficiencia y calidad de estos modelos, demostrada en estudios comparativos [15]. Word2Vec permite capturar las relaciones semánticas y sintácticas entre palabras, lo cual es esencial para comparar con precisión las descripciones de productos. Utilizando estas técnicas avanzadas, se esperaba lograr un modelo que sea rápido, eficiente y sensible a las variaciones sutiles en el lenguaje, conduciendo a resultados más precisos y una mejor comprensión del contenido textual.

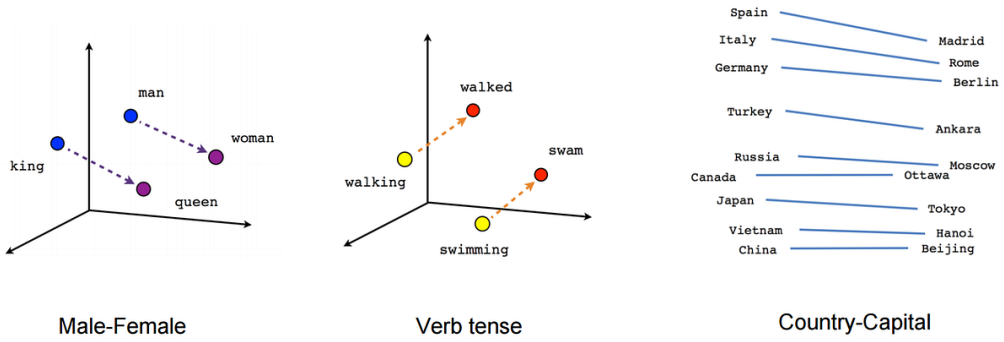

Figura 1.3: Representación visual de word embeddings.[12]

La implementación de word embeddings, enriquecida con estas técnicas avanzadas y haciendo uso de embeddings pre entrenados, no solo mejoró la precisión en la asignación de códigos de productos, sino también optimizó la eficiencia del proceso de clasificación. Al aplicar medidas de similitud suave y regularización, se esperaba lograr un modelo que no solo fuera rápido y eficiente, sino también altamente adaptable y sensible a las variaciones sutiles en el lenguaje. Esto, en última instancia, conduciría a resultados más precisos y una mejor comprensión del contenido textual.

1.5.6. Transformers y Redes Siamesas en similitud de oraciones

Los transformers, introducidos por [11], han revolucionado el campo del Procesamiento de Lenguaje Natural (NLP) y han establecido nuevos estándares en múltiples tareas de NLP, desde la traducción hasta la generación de texto. A diferencia de las arquitecturas basadas en redes neuronales recurrentes (RNN) o convolucionales (CNN), los transformers se basan completamente en mecanismos de atención para capturar relaciones contextuales en el texto.

La arquitectura del transformer consta de un encoder y un decoder, cada uno de los cuales está compuesto por una serie de capas idénticas. La pieza central del transformer es el mecanismo de .ªtención auto-regresiva ", que permite que cada palabra en una secuencia preste atención a todas las demás palabras, ponderando su importancia relativa. Esta capacidad de prestar atención a diferentes partes de una secuencia, independientemente de su posición, le da al transformer una ventaja significativa al manejar dependencias a larga distancia en el texto.

El modelo transformer también introduce el concepto de .ªtención multi-cabeza", donde diferentes versiones de la atención auto-regresiva se calculan y se combinan en

paralelo, permitiendo al modelo capturar diferentes tipos de relaciones en los datos.

Una de las ventajas más significativas de los transformers es su capacidad para ser entrenados en paralelo, a diferencia de las RNNs que requieren cálculos secuenciales. Esto ha permitido el entrenamiento de modelos mucho más grandes y complejos, lo que ha llevado a mejoras significativas en las tareas de NLP.

Los transformers, como se mencionó anteriormente, han revolucionado el campo del Procesamiento de Lenguaje Natural (NLP). BERT, desarrollado por Google, es uno de los modelos transformer más populares y efectivos. Sin embargo, uno de los desafíos con BERT es la sobrecarga computacional al comparar similitud entre oraciones.

En el artículo "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks" por Nils Reimers e Iryna Gurevych[16], se presenta una solución a este desafío. Los autores introducen Sentence-BERT (SBERT), una modificación de BERT que utiliza estructuras de redes siamesas y tripletas para generar embeddings de oraciones que capturan su significado semántico. Estos embeddings pueden ser comparados eficientemente usando similitud coseno, reduciendo significativamente el tiempo y la sobrecarga computacional. Esta capacidad de SBERT para generar representaciones semánticamente significativas

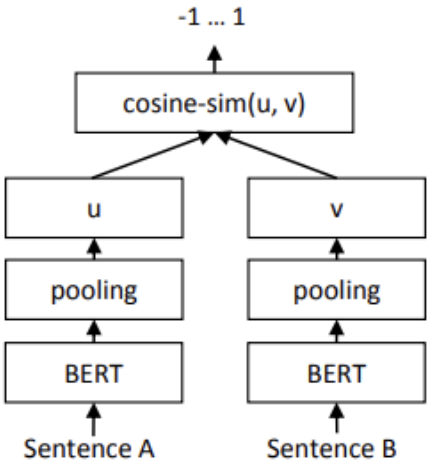

Figura 1.4: Arquitectura de SBERT para inferir scores de similaridad.[16]

de oraciones y compararlas de manera eficiente lo hace especialmente relevante para tareas como la asignación automatizada de códigos de producto basada en descripciones. Al poder determinar con precisión cuán similares son dos descripciones en términos de contenido semántico, se puede mejorar la precisión y eficiencia de la clasificación.

1.6. Metodología

El proyecto se enfocó en utilizar técnicas de NLP de manera iterativa para desarrollar modelos de machine learning que lograran la correcta asignación de codigos basados en descripciones, de manera que cada iteración experimental permitió observar lo distintos resultados de cada técnica aplicada y como afecta esto de manera positiva o negativa a los modelos entrenados. Se exploraron métodos como word embeddings y transformers modificados con redes siamesas (SBERT) para este fin. Algunos de los pasos a seguir para lograr este cometidos fueron:

- **Recolección y Preparación de Datos**: Se realizó una cuidadosa selección de datasets que contenían descripciones detalladas de productos. Los datos fueron

limpiados, normalizados y preprocesados para garantizar su calidad. Este proceso incluyó la eliminación de datos irrelevantes o duplicados, y la transformación de textos en un formato adecuado para el modelado.

- **Desarrollo de Modelos**: Se implementaron dos modelos principales: uno basado en word embeddings y otro en transformers modificados con redes siamesas(SBERT). Se experimentó con diferentes configuraciones y técnicas para encontrar la combinación óptima que ofrezca la mejor precisión y eficiencia.
- **Evaluación y Optimización de Modelos**: Los modelos fueron evaluados utilizando métricas estándar en NLP para medir rendimiento de modelos multiclase, como lo es *top-k accuracy*. Se realizó ajustes iterativos basados en los resultados para optimizar el rendimiento, incluyendo la modificación de parámetros y la reestructuración del modelo si es necesario.
- **Análisis de Resultados y Ajustes**: Se analizaron los resultados de las pruebas para identificar áreas de mejora. Se realizaron ajustes y refinamientos en los modelos para mejorar la precisión y eficiencia en la asignación de códigos a productos. Se documentaron los hallazgos y se utilizarán para informar las decisiones futuras del proyecto.

1.7. Delimitaciones y Alcances

Se poseen siguientes delimitaciones y alcances para este proyecto:

- Todo el desarrollo se llevará a cabo utilizando el lenguaje de programación Python.
- El proyecto se centrará únicamente en la asignación automática de códigos de productos en la Bolsa Mercantil de Colombia (BMC) basándose en las descripciones proporcionadas por sus Sociedades Comisionistas (SC).
- Los modelos implementados utilizarán técnicas de procesamiento de lenguaje natural (NLP) y aprendizaje profundo, específicamente word embeddings con Word2Vec, modelos preentrenados de spaCy, métricas de similitud como Jaccard y coseno, y redes siamesas (SBERT).
- El alcance del proyecto incluye la recolección, preprocesamiento y análisis de datos, el desarrollo y evaluación de modelos de aprendizaje automático, y la comparación de sus resultados.
- No se considerarán otras fuentes de datos para entrenamiento ni se integrarán metadatos adicionales más allá de las descripciones textuales proporcionadas por las SC.

1.7.1. Entregables

Como resultado de este proyecto se entregará lo siguiente:

- Documento de tesis.
- Un script en Python que contendrá el código del preprocesamiento de los datos, el entrenamiento de los modelos y su respectiva evaluación.
- Informes detallados sobre la precisión y el rendimiento de cada modelo implementado.
- Recomendaciones y sugerencias para futuras investigaciones y mejoras basadas en los hallazgos del proyecto.

1.8. Relevancia

Este proyecto es relevante en el contexto empresarial colombiano, ya que puede generar ahorros sustanciales de tiempo y recursos para las Sociedades Comisionistas de la BMC. Además, contribuirá al avance de la aplicación de la inteligencia artificial en el sector financiero de Colombia y podría tener un impacto positivo en la competitividad de la industria.

Capítulo 2

Conjunto de datos

2.1. Introducción al conjunto de datos

El conjunto de datos que fundamenta este proyecto se origina de la colaboración con la Bolsa Mercantil de Colombia (BMC) y las Sociedades Comisionistas afiliadas. Este conjunto de datos es esencial para la realización de nuestro estudio, dado que su núcleo está compuesto por una lista detallada de códigos BMC, los cuales se acompañan de descripciones precisas para cada código. La relevancia de estos códigos y sus descripciones asociadas radica en que constituyen el objetivo principal de nuestro proceso de clasificación de productos, permitiendo una identificación efectiva y precisa en el marco de la automatización propuesta. Para ofrecer una visión concreta del formato y la estructura de estos datos, el cuadro 2.1 presenta un extracto representativo de la lista de códigos BMC.

codigo_subyacente_caracteristica	**descripcion**
1	ARROZ CASCARA IMPORTADO - AMERICANO - KILOGRAMO...
2	ARROZ CASCARA IMPORTADO - AMERICANO - KILOGRAMO ...
3	ARROZ CASCARA IMPORTADO - ARGENTINO - KILOGRAMO ...
4	ARROZ CASCARA IMPORTADO - ARGENTINO - KILOGRAMO ...
5	ARROZ CASCARA IMPORTADO - ECUATORIANO - KILOGRAMO ...
...	

Cuadro 2.1: Descripción de características (Parte de una tabla más grande)

Podemos observar un resumen del total de codigos y los tipos de datos del conjunto provisto por la BMC en el cuadro 2.2, ademas de atributos que no fueron inlcuidos por cuestiones de espacio en el cuadro 2.1. Debemos tener en cuenta que aunque los codigos son tomados como tipo de dato int64, su uso dentro de nuestro ejercicio bien podría ser el mismo que el de un tipo object o string, ya que estos son usados para clasificar y no tiene ningún valor o uso numérico real.

Adicionalmente, hemos recibido contribuciones en forma de varios conjuntos de datos proporcionados por 28 empresas que hacen parte de las Sociedades Comisionistas de la BMC. Estos conjuntos han sido etiquetados a mano, asociando las diversas descripciones de productos con sus correspondientes códigos BMC. La importancia de estos datos donados reside en su papel fundamental para el entrenamiento de los modelos de aprendizaje automático que estamos desarrollando.

El cuadro 2.3 ofrece un ejemplo del formato de estos conjuntos de datos donados, proporcionando un vistazo a la variedad de las descripciones de productos que se han etiquetado. Este insight inicial es indispensable para comprender la complejidad del desafío que enfrentamos y la meticulosidad requerida en el procesamiento y análisis de

Característica	**Detalle**
Número de registros	7958
Tipo de atributos	
codigo_subyacente_caracteristica	int64
descripcion	object
estado	object
permitido_orf	object
desc_subya	object
desc_caracteristica	object
desc_unidad	object
desc_empaque	object
desc_naturaleza	object

Cuadro 2.2: Resumen del conjunto de datos

los datos para lograr una clasificación exitosa.

ID	CODIGO BMC	CODIGO CLIENTE	DESCRIPCION	FACTOR CONVERSION
811045607	8198	62870	AZAFRAN ESPAÑOL BADIA x 0.4GR	0,0004
811045607	1548	39036	WHISKAS GATITOS CARNE x 1.4 GR	0,0014
811045607	1548	91623	ALIMENT GATO HAIRBALL MEOW MIX*1.42GR	0,00142
811045607	1548	91622	ALIMENT GATO ORIGINAL MEOW MIX*1.42GR	0,00142
...				

Cuadro 2.3: Descripción de productos y factores de conversión.

En el cuadro 2.4 podemos observar un resumen de los conjuntos de datos provistos por 28 empresas que donaron esta información para nuestro proyecto, además del tipo de algunas columnas que, por espacio, no fueron incluidas en el cuadro 2.3.

Detalle	**Valor**
Número de registros	34136
Tipo de atributos:	
FECHA HORA	float64
TIPO ID	object
ID	float64
CODIGO BMC	float64
CODIGO CLIENTE	object
FACTOR CONVERSION	float64
Nit. Cliente	float64
CODIGO PROD. CLIENTE	float64
DESCRIPCION PRODUCTOS	object
DESCRIPCION_CLIENTE	object

Cuadro 2.4: Resumen de los datos

Para finalidad de este proyecto podemos observar de primera mano como algunos de los atributos más relevantes para la asignación de códigos basandonos en la similitud de descripciones de los distintos productos aqui listados son la descripcion del producto dada por la BMC, la descripcion del producto por parte de la SC y la correcta asignacion manual del codigo BMC para los distintos productos aqui listados.

2.2. Preprocesamiento y Limpieza de Datos

El preprocesamiento y la limpieza de datos constituyen etapas cruciales en cualquier proyecto de aprendizaje automático, incluido este. Estos procesos tienen como objetivo maximizar la utilidad de los datos disponibles y minimizar el impacto negativo que pueden tener los datos irrelevantes o ruidosos en el análisis posterior. Para nuestro proyecto, este proceso comienza con la unificación de los distintos conjuntos de datos donados por las Sociedades Comisionistas (SC), lo cual nos permite tener una visión comprensiva y global del corpus con el que trabajaremos, además de facilitar un acceso más práctico a los datos.

2.2.1. Unificación de Conjuntos de Datos de las SC

El primer paso en nuestro proceso de preprocesamiento consiste en combinar todos los conjuntos de datos proporcionados por las SC. Esta unificación es esencial para establecer una base de datos homogénea y amplia, permitiéndonos realizar análisis más robustos y representativos. Al consolidar estos conjuntos, nos aseguramos de tener una muestra lo suficientemente grande y variada, crucial para la eficacia de los modelos de aprendizaje automático que desarrollaremos. El cuadro 2.4 provee una vista general del total de registros obtenidos, los atributos resultantes de unir todas estos conjuntos de datos y sus respectivos tipos.

2.2.2. Limpieza de Datos

Posteriormente, nos enfocamos en la limpieza de los datos, lo cual implica varias subetapas detalladas a continuación:

Identificación y Manejo de Valores Nulos:

La presencia de valores nulos o de tipo NaN puede distorsionar el análisis y afectar el rendimiento de los modelos, por este motivos determinamos que aquellas columnas donde la mayoria de valores son de tipo NaN deberian de ser descartadas automáticamente. Podemos observar los resultados de este análisis después de identificar las columnas con una alta proporción de valores nulos.

Columna	**% NaN**
FECHA HORA	100.000000
TIPO ID	0.000000
ID	0.799742
CODIGO BMC	0.764589
CODIGO CLIENTE	0.799742
FACTOR CONVERSION	0.000000
Nit. Cliente	99.964846
CODIGO PROD. CLIENTE	99.964846
DESCRIPCION PRODUCTOS	99.964846
DESCRIPCION_CLIENTE	0.799742

Cuadro 2.5: Porcentaje de valores NaN por columna

Con esta información vemos que atributos como 'FECHA HORA', 'Nit. Cliente', 'CODIGO PROD. CLIENTE', 'DESCRIPCION PRODUCTOS', pueden ser descartados de nuestro corpus al ser en su inmensa mayoría datos con valores nulos.

Eliminación de Atributos Irrelevantes:

Determinamos la relevancia de cada columna evaluando su aporte potencial al modelo. Aquellos atributos que no añaden valor, ya sea por ser información duplicada o por no tener relación con la asignación de códigos BMC, son eliminados. Este enfoque asegura que solo mantenemos datos que directamente influencian la precisión y eficacia de nuestras predicciones.

Al revisar nuestro conjunto de datos, identificamos que los atributos 'TIPO ID', 'ID' y 'FACTOR CONVERSION' tienen la menor relevancia para nuestro análisis. Por ejemplo, 'TIPO ID' muestra un valor constante ('A') en todas sus entradas, lo cual no aporta información útil para nuestros propósitos. Por otro lado, 'ID' se utiliza para señalar alguna característica específica del producto; sin embargo, en la comparación de similitudes entre descripciones, un valor numérico tiene poco que ofrecer, especialmente cuando la identificación precisa del producto ya es facilitada por el atributo 'CODIGO CLIENTE'. Este hecho hace que 'ID' sea de valor limitado para nuestro análisis. Finalmente, 'FACTOR CONVERSION', que indica el valor numérico asociado a la conversión del peso del producto a una medida más normalizada y estándar, tampoco es relevante para este proyecto, ya que los valores numéricos incluidos en las descripciones no son de nuestro interés y probablemente serán omitidos.

El cuadro 2.6 nos permite visualizar los atributos resultantes en nuestro corpus después de este ejercicio de descarte.

Detalle	**Valor**
Número de registros	34136
Tipo de atributos:	
CODIGO BMC	float64
CODIGO CLIENTE	object
DESCRIPCION_CLIENTE	object

Cuadro 2.6: Resumen de los datos posterior a eliminacion de atributos

Detección y Eliminación de Duplicados:

Los registros duplicados pueden introducir un sesgo en el análisis, por lo que es crucial identificar y eliminar estos duplicados. Implementamos procedimientos automatizados para detectar tales instancias, como lo es el método `duplicated()` y usamos el método `drop_duplicates()` (ambos de la librería Pandas de Python) para deshacernos de estos datos, garantizando que cada entrada en nuestro conjunto de datos sea única y contribuya de manera significativa al entrenamiento del modelo. En el cuadro 2.7 podemos observar como el numero de registros cambia después de hacer esta revisión.

Detalle	**Valor**
Número de registros	17611
Tipo de atributos:	
CODIGO BMC	float64
CODIGO CLIENTE	object
DESCRIPCION_CLIENTE	object

Cuadro 2.7: Resumen de los datos posterior a eliminación de duplicados

Normalización y Estandarización:

Nuestro conjunto de datos incluye códigos BMC como únicos elementos numéricos, que sirven más como etiquetas categóricas que como valores numéricos reales. Por lo tanto, la normalización y estandarización tradicionales no son aplicables en este contexto. La acción que hemos decidido tomar es convertir los códigos BMC de tipo float a int, simplificando así el manejo de los datos y alineándolos más estrechamente con su uso práctico como identificadores discretos. Esta conversión mejora la eficiencia computacional y facilita el procesamiento y análisis posterior sin alterar la integridad o el significado de los datos.

Detalle	**Valor**
Número de registros	17611
Tipo de atributos:	
CODIGO BMC	int64
CODIGO CLIENTE	object
DESCRIPCION_CLIENTE	object

Cuadro 2.8: Resumen de los datos posterior a cambio de tipos

2.2.3. Integración de Descripciones BMC

Para mejorar nuestro análisis, hemos vinculado cada producto en nuestro conjunto de datos con la descripción estándar del BMC, utilizando el código BMC como referencia. Esto asegura una comparación precisa entre las descripciones de productos de las SC y los estándares del BMC, mejorando la coherencia y precisión del análisis. Tras la integración

CODIGO BMC	CODIGO CLIENTE	DESCRIPCION_CLIENTE	DESCRIPCION_BMC
6361	340	ABLANDA CARNES UNIDOS X 20GR PX12	MEZCLA DE CONDIMENTOS - ...
139	759	ACACIAS DE LA INIDIA UNIDOS X15 P12	AROMATICAS ELABORADAS - SC -...
1319	973	ACEITE DE OLIVA 100% PURO X1000 C15	ACEITE REFINADO DE OLIVA KG -...
1319	894	ACEITE DE OLIVA 100% PURO X500 CX24	ACEITE REFINADO DE OLIVA KG -...
8347	1041	ACEITE PAHORRAR VEGETAL X2700ML CX6	MEZCLA DE ACEITES VEGETALES - ...
...			

Cuadro 2.9: Descripción de productos y su correspondencia en BMC

de las descripciones del BMC a nuestro conjunto de datos, realizamos una revisión para identificar filas con valores faltantes (NaN). De los 17,611 registros analizados, identificamos 323 filas con valores NaN. Dada la importancia de contar con un conjunto de datos completo para nuestro análisis y modelado, decidimos eliminar estas filas.

Detalle	**Valor**
Número de registros	17288
Tipo de atributos:	
CODIGO BMC	int64
CODIGO CLIENTE	object
DESCRIPCION_CLIENTE	object
DESCRIPCION_BMC	object

Cuadro 2.10: Resumen de los datos posterior a la integracion de de descripciones BMC y limpieza de datos faltantes

2.2.4. Estandarizacion de textos

La estandarización de texto es un paso crucial para preparar nuestros datos para análisis consistentes y comparaciones precisas entre las descripciones de productos de las SC y las descripciones del BMC. A continuación, describimos los pasos implementados para estandarizar las descripciones en nuestro corpus.

Conversión a Minúsculas

En la mayoría de las herramientas y bibliotecas de NLP, la práctica estándar es convertir el texto a minúsculas. Esto ayuda a mantener la consistencia cuando se utilizan diferentes recursos y herramientas que ya esperan texto en minúsculas.

Convertimos todas las descripciones a minúsculas para uniformar el caso de texto y eliminar variaciones debidas a diferencias de mayúsculas o minúsculas.

```
1
2 resultado_df_limpio['DESCRIPCION_CLIENTE'] = resultado_df_limpio['
      DESCRIPCION_CLIENTE'].str.lower()
3
4 resultado_df_limpio['DESCRIPCION_BMC'] = resultado_df_limpio['
      DESCRIPCION_BMC'].str.lower()
```

Listing 2.1: Codigo en Python para la conversion de minusculas

Separación de Números y Texto

Al separar números de texto, facilitamos el procesamiento analítico y lingüístico del contenido, permitiendo que el texto y los números se manejen de acuerdo a sus necesidades específicas. Esto mejora la claridad del análisis y la precisión de los modelos de datos. Aplicamos una función para separar palabras de números y unidades comunes.

```
1
2 def separate_numbers(text):
3     text = re.sub(r'(\D)(\d)', r'\1 \2', text)
4     text = re.sub(r'(\d)(\D)', r'\1 \2', text)
5     return text
6
7 resultado_df_limpio['DESCRIPCION_CLIENTE'] = resultado_df_limpio['
      DESCRIPCION_CLIENTE'].apply(separate_numbers)
```

Listing 2.2: Código en Python para la separación de números y texto

Limpieza y Formateo de Texto

Eliminar caracteres especiales y espacios extra asegura que el texto se mantenga limpio y sin ruido innecesario. Esto previene errores en la interpretación de los datos por parte de algoritmos de NLP y facilita un análisis más eficiente.

```
1
2 resultado_df_limpio['DESCRIPCION_CLIENTE'] = resultado_df_limpio['
      DESCRIPCION_CLIENTE'].str.strip().str.replace(r'\s+', ' ', regex=True)
3
4 resultado_df_limpio['DESCRIPCION_BMC'] = resultado_df_limpio['
      DESCRIPCION_BMC'].str.strip().str.replace(r'\s+', ' ', regex=True)
5
6 resultado_df_limpio['DESCRIPCION_CLIENTE'] = resultado_df_limpio['
      DESCRIPCION_CLIENTE'].str.replace(r'[^a-zA-Z0-9\s]', '', regex=True)
7
8 resultado_df_limpio['DESCRIPCION_BMC'] = resultado_df_limpio['
      DESCRIPCION_BMC'].str.replace(r'[^a-zA-Z0-9\s]', '', regex=True)
```

Listing 2.3: Código en Python para limpieza y formateo de datos

Estandarización de Unidades y Términos Comunes

Normalizar términos y unidades mediante su conversión a formas estándares reduce la variabilidad en el conjunto de datos. Esto facilita la comparación directa y la agregación de datos, y mejora la interpretación automática del texto.

```
1
2  # Crear un diccionario con las abreviaturas y sus reemplazos
3  replacements = {
4      r'\bg\b': 'gramos',
5      r'\bgms\b': 'gramos',
6      r'\bgrs\b': 'gramos',
7      r'\bgr\b': 'gramos',
8      r'\bkg\b': 'kilogramo',
9      r'\bml\b': 'mililitros',
10     r'\blt\b': 'litro'
11 }
12
13 # Aplicar los reemplazos a las columnas deseadas
14 for key, value in replacements.items():
15     resultado_df_limpio['DESCRIPCION_CLIENTE'] = resultado_df_limpio['
           DESCRIPCION_CLIENTE'].str.replace(key, value, regex=True)
16     resultado_df_limpio['DESCRIPCION_BMC'] = resultado_df_limpio['
           DESCRIPCION_BMC'].str.replace(key, value, regex=True)
```

Listing 2.4: Código en Python para la estandarización de unidades

Eliminación de Stopwords

Remover palabras comunes que no contribuyen significativamente al significado del texto (stopwords) enfoca el análisis en el contenido más relevante. Esto reduce la carga de procesamiento y mejora la eficacia de las técnicas de análisis de texto.

```
1  nltk.download('stopwords')
2
3  stop_words = set(stopwords.words('spanish'))
4
5  def remove_stopwords(text):
6      words = text.split()
7      return ' '.join(word for word in words if word.lower() not in
           stop_words)
8
9  resultado_df_limpio['DESCRIPCION_CLIENTE'] = resultado_df_limpio['
       DESCRIPCION_CLIENTE'].apply(remove_stopwords)
10
11 resultado_df_limpio['DESCRIPCION_BMC'] = resultado_df_limpio['
       DESCRIPCION_BMC'].apply(remove_stopwords)
```

Listing 2.5: Código en Python para la eliminación de Stopwords

Estos pasos garantizan que las descripciones de productos estén preparadas de manera óptima para el análisis, mejorando la coherencia, legibilidad y la precisión en la comparación de textos. El cuadro 2.11 nos da un ejemplo de como cambiaron los datos después de este proceso en los atributos principales

DESCRIPCION_CLIENTE	DESCRIPCION_BMC
ablanda carnes unidos x 20 gramo px 12	mezcla condimentos ablanda carnes kilogramo empacadoa procesado
acacias inidia unidos x 15 p 12	aromaticas elaboradas sc kilogramo empacadoa procesado
aceite oliva 100 puro x 1000 c 15	aceite refinado oliva kilogramo 661 sc kilogramo empacadoa procesado
aceite oliva 100 puro x 500 cx 24	aceite refinado oliva kilogramo 661 sc kilogramo empacadoa procesado
aceite pahorrar vegetal x 2700 mililitros cx 6	mezcla aceites vegetales mezcla aceites vegetales procesado
...	

Cuadro 2.11: Descripción de Cliente y Descripción BMC posterior a estandarización de textos

2.2.5. Preparación de los Datos para el Modelado

Para preparar nuestras descripciones de productos para análisis de texto más avanzados y precisos, hemos implementado procesos de tokenización y lematización. Estos pasos son cruciales para descomponer el texto en unidades más manejables y para reducir las palabras a su forma base o lema.

Tokenización

La tokenización es el proceso de dividir cadenas de texto en piezas más pequeñas, o tokens, que son generalmente palabras. Esto facilita el manejo individual de cada palabra dentro del procesamiento de lenguaje natural.

TOKENS_DESCRIPCION_CLI...	TOKENS_DESCRIPCION_BMC
[ablanda, carnes, unidos, x, 20, gramo, px, 12]	[mezcla, condimentos, ablanda, carnes, kilogramo, ...]
[acacias, inidia, unidos, x, 15, p, 12]	[aromaticas, elaboradas, sc, kilogramo, empacado, ...]
[aceite, oliva, 100, puro, x, 1000, c, 15]	[aceite, refinado, oliva, kilogramo, 661, sc, ...]
[aceite, oliva, 100, puro, x, 500, cx, 24]	[aceite, refinado, oliva, kilogramo, 661, sc, ...]
[aceite, pahorrar, vegetal, x, 2700, mililitros, cx, 6]	[mezcla, aceites, vegetales, mezcla, aceites, ...]
...	

Cuadro 2.12: Comparación de Tokens de Descripción de Cliente y BMC

Lematización

La lematización es un proceso más sofisticado que la tokenización. Se trata de reducir las palabras a su lema o forma base, considerando su rol gramatical. Este paso es crucial para agrupar variantes de una palabra para que puedan ser analizadas como un único elemento, mejorando la precisión de los análisis semánticos y estadísticos.

LEMMA_DESCRIPCION_CLIENT...	LEMMA_DESCRIPCION_BMC
[ablandar, carne, unido, x, 20, gramo, px, 12]	[mezclar, condimento, ablandar, carne, kilogramo, ...]
[acacia, inidiar, unido, x, 15, p, 12]	[aromatica, elaborado, sc, kilogramo, empacado, ...]
[aceite, olivo, 100, puro, x, 1000, c, 15]	[aceite, refinado, olivo, kilogramo, 661, sc, ...]
[aceite, olivo, 100, puro, x, 500, cx, 24]	[aceite, refinado, olivo, kilogramo, 661, sc, ...]
[aceite, pahorrar, vegetal, x, 2700, mililitros, cx, 6]	[mezclar, aceite, vegetal, mezclar, aceit, vegetal, ...]
...	

Cuadro 2.13: Comparación de Lemma Tokens de Descripción de Cliente y BMC

2.3. Análisis Exploratorio de Datos

Antes de proceder con el desarrollo y entrenamiento de los modelos, es crucial realizar un análisis exploratorio de los datos para comprender mejor sus características, identificar patrones y detectar posibles anomalías o problemas que deban abordarse. Este proceso proporciona información valiosa que puede guiar las decisiones sobre el preprocesamiento de datos y la selección de técnicas de modelado apropiadas.

2.3.1. Estadísticas descriptivas básicas:

Se calcularon las estadísticas descriptivas básicas. En el caso de las columna de 'CODIGO BMC', esta fue tomada como una columna no numérica al ser cada código un valor que tiene una función de etiqueta, más que de valor numérico como tal, con lo cual se reportaron el recuento de valores únicos y la frecuencia de cada valor.

Medida	COD. BMC	COD. CLIENTE	DESC._CLIENTE	DESC._BMC
Count	17288	17288	17288	17288
Unique	2319	16508	17014	2315
Top	7949	2	azucar	yogurt yogurt ...
Freq	419	6	6	419

Cuadro 2.14: Resumen estadístico de las descripciones de las SC y BMC

Los datos comprenden 17,288 observaciones, destacándose la variabilidad en la nomenclatura utilizada por las SC comparada con las descripciones estandarizadas de la BMC. La diversidad de descripciones se refleja en la aparición de 16,508 códigos únicos de las SC y 2315 descripciones únicas proporcionadas por BMC, señalando una amplia gama de productos y una potencial complejidad en su categorización.

La moda para las descripciones de las SC apunta a 'azúcar' como el término más frecuente, sugiriendo que ciertos productos tienen una presencia más destacada en las transacciones. Por otro lado, la descripción más común de BMC es 'yogurt saborizado unidad empacadoa proc...', que aparece 419 veces, indicando posiblemente un estándar en la clasificación de productos lácteos.

2.3.2. Visualizaciones clave

Uno de los enfoques visuales utilizados fue el histograma de frecuencias para la columna 'CODIGO BMC'. Este gráfico proporciona una representación clara de la distribución de los códigos de producto utilizados por las SC, resaltando los códigos más comunes y cualquier patrón subyacente en su ocurrencia. A través de histogramas de frecuencia

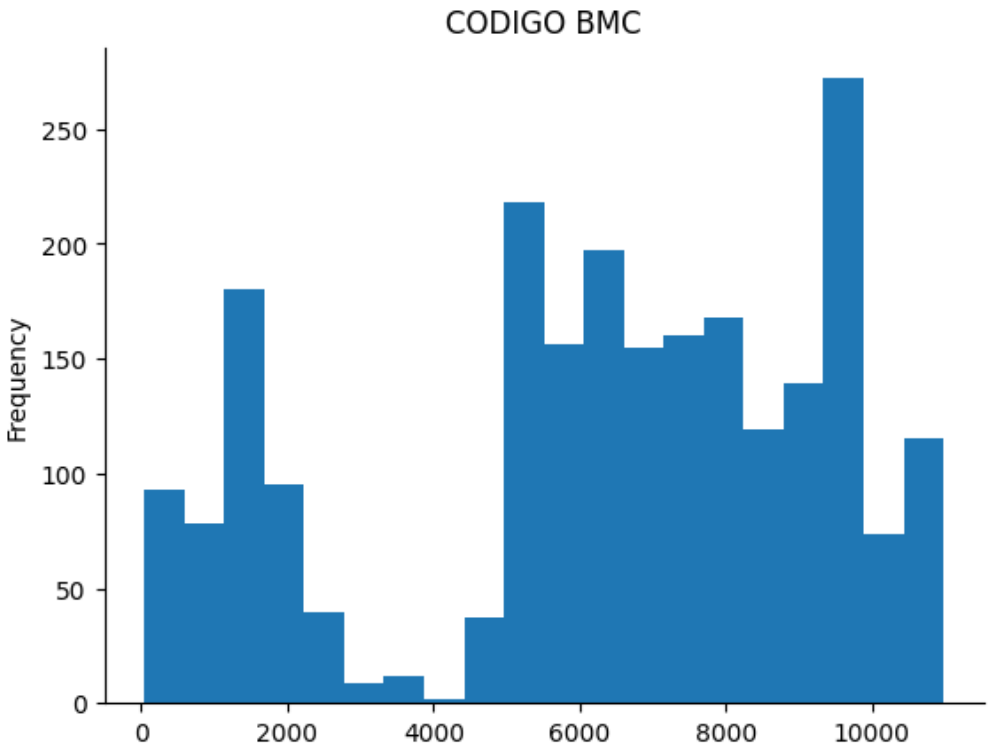

Figura 2.1: Histograma de frecuencias para la columna 'CODIGO BMC'

de códigos BMC, observamos una distribución no uniforme, con picos significativos que podrían representar categorías de productos de alta rotación o de mayor importancia comercial. Estos gráficos resaltan la presencia de códigos BMC específicos que dominan el conjunto de datos, lo que podría reflejar tanto tendencias del mercado como prácticas estandarizadas de etiquetado por parte de la BMC.

Códigos de producto faltantes:

Se descubrió que un porcentaje significativo (70.85 %) de los códigos subyacentes listados en la base de datos de la Bolsa Mercantil de Colombia (BMC) no se utilizaban en las descripciones proporcionadas por las SC. Este hallazgo sugiere la necesidad de revisar la relevancia de los códigos subyacentes, analizar las necesidades de los clientes y considerar ajustes en la estrategia de inventario y oferta de productos.

Productos más relevantes:

A través del análisis de frecuencias, se identificaron los códigos de producto más comúnmente utilizados por las SC. Esta información puede ser valiosa para priorizar los

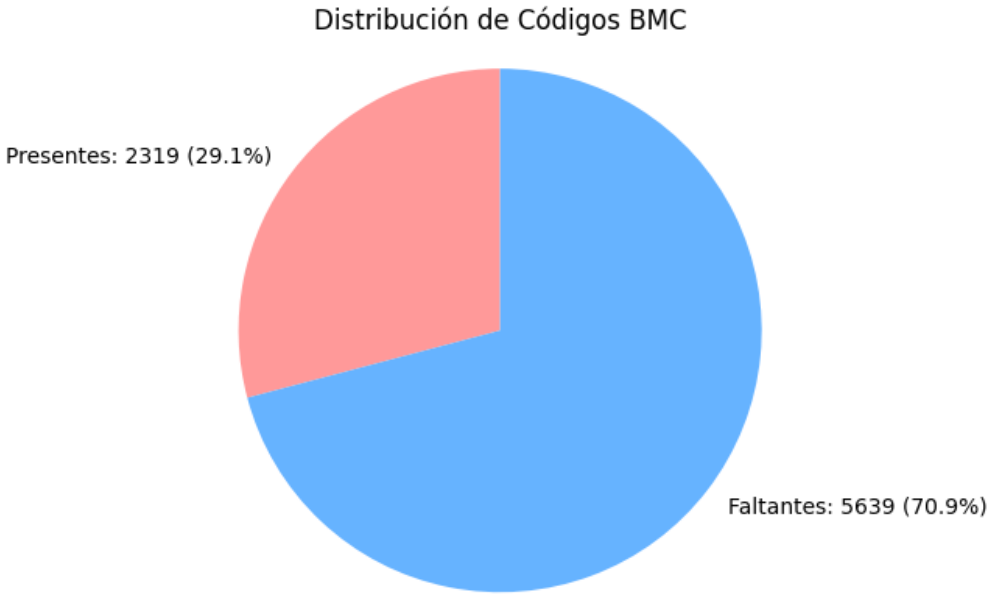

Figura 2.2: Proporción de códigos BMC usados por las SC

esfuerzos de asignación de códigos y garantizar una alta precisión en los productos más demandados.

CODIGO BMC	DESCRIPCION_BMC	Frecuencia
7949	yogurt yogurt saborizado unidad empacadoa proc...	419
1550	alimento concentrado perros sc kilogramo empac...	279
6074	galleta dulce galleta dulce relleno unidad emp...	262
5840	galleta dulce relleno unidad empacadoa procesado	227
5078	vino vino tinto unidad empacadoa procesado	204
...	...	...
7903	mezcla condimentos salsa base camarones fermen...	1
7916	hongo shitake kilogramo empacadoa natural	1
4720	proteina soya sc kilogramo granel procesado	1
4685	leche polvo entera nacional kilogramo sc kilog...	1
43	maiz amarillo importado argentino kilogramo sa...	1

Cuadro 2.15: Descripciones más frecuentes en la categoría BMC

El código 7949 ('yogurt yogurt saborizado...') es el más común, con 419 ocurrencias. Los códigos 1550 ('alimento concentrado perros sc kilogramo empac...') y 6074 ('galleta dulce galleta dulce relleno unidad emp...') también son muy frecuentes, con 279 y 262 ocurrencias, respectivamente. Varios códigos tienen una sola ocurrencia, lo que sugiere productos menos comunes o posibles anomalías.

Análisis de Frecuencia de Términos

El análisis de la frecuencia de términos en las descripciones de las SC revela una preponderancia de términos relacionados con unidades de medida y categorías de pro-

ductos. El término más frecuente es "gramo", seguido de cerca por "x", lo que podría estar indicando el uso de "xçomo marcador de cantidad o tamaño en la lista de productos. Este hallazgo sugiere que las SC tienden a especificar productos en términos de su peso, lo que es crucial para la clasificación y el etiquetado en el contexto de la BMC.

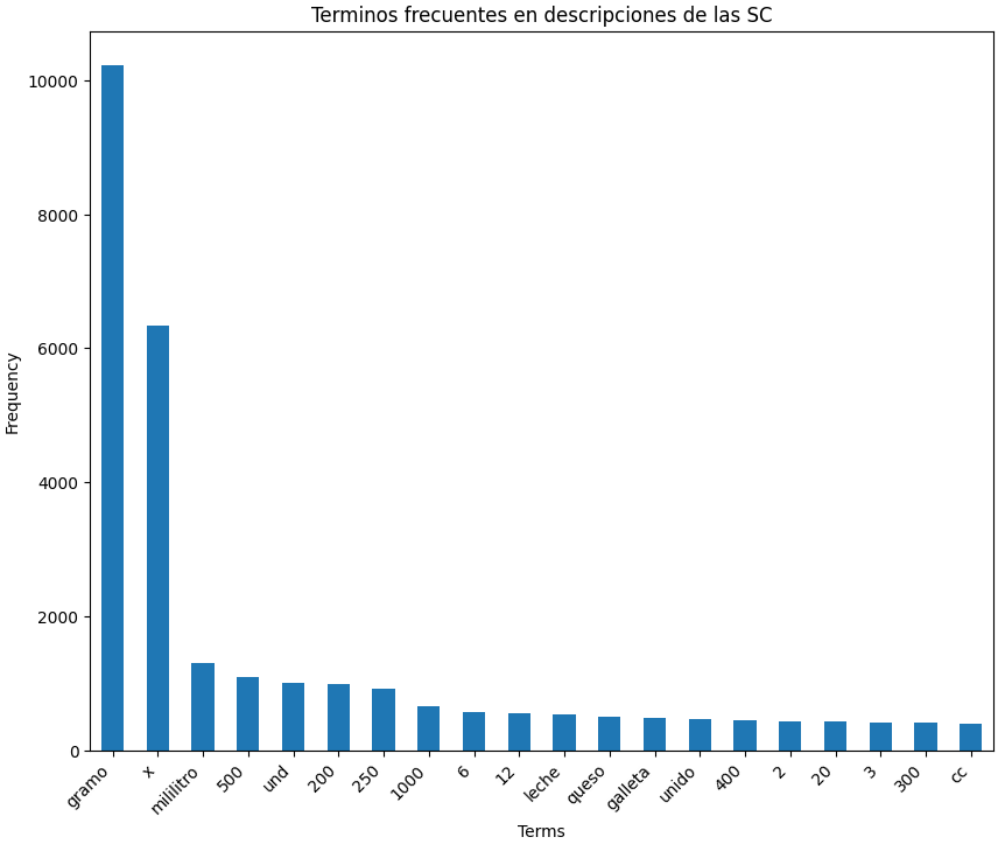

Figura 2.3: Frecuencia de términos usada por las SC

En cuanto a las descripciones del BMC, los términos con mayor frecuencia como "procesado", "kilogramoz ünidadïndican un lenguaje estandarizado utilizado en las especificaciones de productos. Estos términos reflejan la naturaleza y la presentación de los productos, mostrando que el BMC prioriza la forma en que los productos son procesados y su cantidad al detallar descripciones.

El segundo gráfico de frecuencia de términos muestra que términos como .^{em}pacadoz ünidad"son altamente prevalentes, lo que sugiere que el empaque y la presentación de los productos son consideraciones importantes en el BMC.

Errores ortográficos:

Se detectaron palabras mal escritas en las descripciones de productos proporcionadas por las SC y la BMC. Aunque algunas de estas palabras pueden ser nombres de marcas o abreviaturas, es importante abordar los errores ortográficos reales, ya que pueden afectar el rendimiento de las técnicas de procesamiento de lenguaje natural, como la lematización.

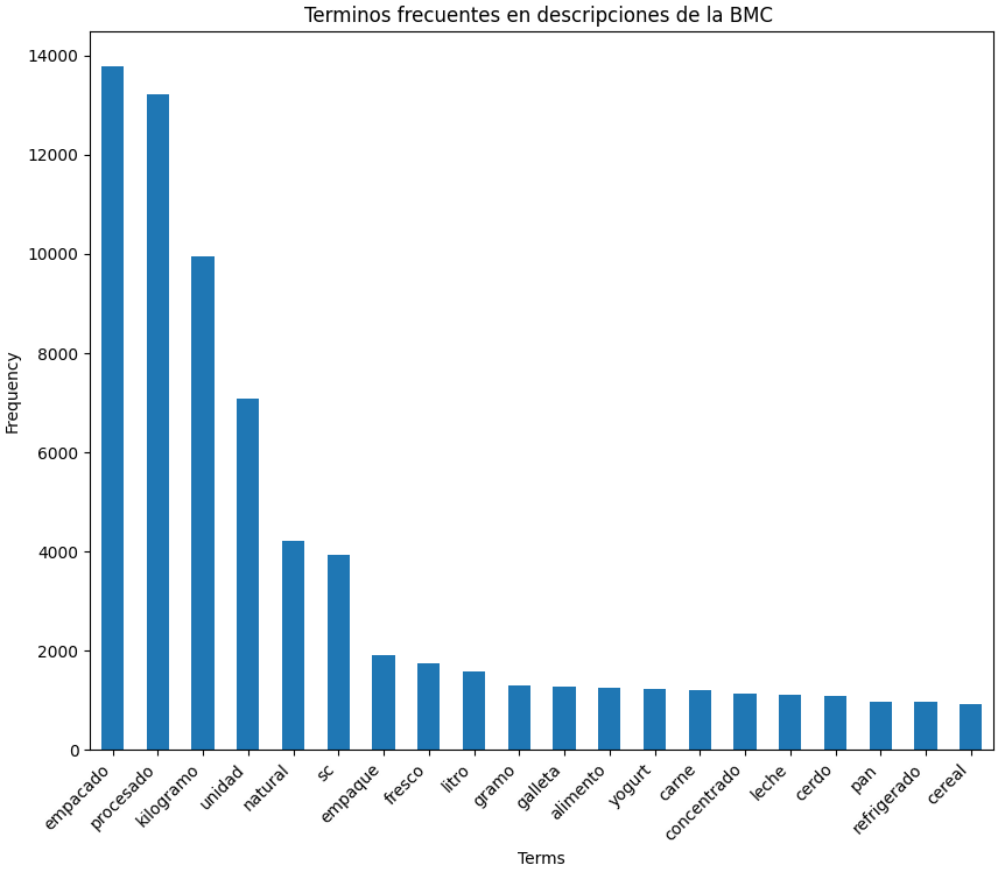

Figura 2.4: Frecuencia de términos usada por la BMC

Nuestra revisión de las descripciones de productos proporcionadas por las SC y aquellas de la BMC ha identificado un número significativo de errores ortográficos, sumando un total de 5,567 y 699 instancias, respectivamente.

2.3.3. Observaciones preliminares

El análisis de frecuencia de términos y las descripciones de productos revela discrepancias sustanciales entre el lenguaje utilizado por las SC y las descripciones estandarizadas por BMC. Este desajuste presenta un desafío significativo para la alineación automática de los códigos de productos, una tarea esencial para la clasificación precisa y eficiente en la Bolsa Mercantil de Colombia.

Los histogramas de frecuencia de términos resaltan la presencia de ciertos códigos BMC con una utilización mucho más alta que otros, lo que indica una oportunidad para enfocar la formación de modelos de clasificación supervisada en torno a estos códigos preponderantes. Este enfoque podría facilitar la identificación de patrones de productos y la predicción de categorías de productos para nuevos registros de descripciones.

La normalización de las descripciones de productos surge como un elemento crucial en la preparación de los datos. El objetivo es mejorar la calidad de la lematización y, en consecuencia, la eficacia de los algoritmos de aprendizaje automático utilizados. Ajustar las descripciones para que reflejen un estándar común permitirá un procesamiento

Descripción	**Instancias de Errores Ortográficos**
Descripciones de Productos de las SC	5567
Descripciones de Productos de BMC	699

Cuadro 2.16: Comparación de Errores Ortográficos en Descripciones de Productos

lingüístico más preciso y consistente, llevando a resultados más fiables.

Además, hemos identificado que los errores ortográficos pueden tener un impacto significativo en la efectividad de las tecnologías de procesamiento de lenguaje natural (NLP), particularmente en tareas de lematización y análisis semántico. Los errores en la escritura pueden resultar en interpretaciones erróneas del contenido textual, afectando así la confiabilidad de cualquier conocimiento generado a partir de estas descripciones. Por lo tanto, es esencial implementar estrategias robustas de corrección ortográfica para mejorar la exactitud de los datos textuales, lo que se traduce en un mejor rendimiento de los sistemas de NLP y en la toma de decisiones empresariales informadas.

Capítulo 3

Desarrollo e Implementación

3.1. Metodologia de prueba e implementacion

En este capítulo, se detalla el proceso de desarrollo e implementación de los modelos de aprendizaje automático diseñados para la asignación automatizada de códigos de productos a partir de la similitud de descripciones textuales. Se emplearon técnicas de Procesamiento de Lenguaje Natural (NLP) y modelos de aprendizaje automático, específicamente Word Embeddings y Sentence Transformers (SBERT), para lograr este objetivo. A continuación, se describen los pasos seguidos para la implementación y desarrollo de los modelo.

1. **Carga de Modelos:** Se seleccionaron y cargaron los modelos pertinentes de Word Embeddings y Sentence Transformers. Estos modelos fueron elegidos por su capacidad para capturar las relaciones semánticas entre palabras y frases, lo cual es esencial para medir la similitud entre descripciones textuales.

2. **Generación de Embeddings:** Utilizando los modelos cargados, se generaron los embeddings de palabras y frases. Estas representaciones vectoriales en un espacio de alta dimensión permiten comparar descripciones textuales de manera eficaz.

3. **Cálculo de Similitud del Coseno:** Una vez generados los embeddings, se calculó la similitud del coseno entre los vectores correspondientes a las descripciones de los productos. La similitud del coseno es una métrica que mide el ángulo entre dos vectores en un espacio vectorial, proporcionando un valor que indica cuán similares son dos descripciones.

4. **Evaluación de la precisión:** Para evaluar el rendimiento de los modelos, se comparó la similitud del coseno más alta con la descripción correspondiente de la Bolsa Mercantil de Colombia (BMC). La precisión del modelo se determinó calculando el porcentaje de coincidencias correctas entre las descripciones de los productos y los códigos asignados. el detalle de este proceso puede verse en la subseccion 3.1.1

3.1.1. Evaluación de Correspondencia

Para evaluar el rendimiento de nuestros modelos, no se utilizaron las métricas más convencionales, sino una versión modificada de la métrica **top-k accuracy**.

Modificación de la Top-k Accuracy

Debido a la aproximación poco convencional de nuestra tarea, modificamos la top-k accuracy para que se ajuste mejor a nuestras necesidades específicas. En lugar de simplemente verificar si la descripción correcta del producto está entre las k principales predicciones, adaptamos la métrica para evaluar la correspondencia de descripciones de productos y códigos asignados.

Esta métrica se basa en la proporción de coincidencias correctas entre las descripciones de los productos y los códigos asignados. Específicamente, calculamos la precisión evaluando si la descripción del producto con la mayor similitud del coseno coincide con la descripción correspondiente de la Bolsa Mercantil de Colombia (BMC). Si la descripción más similar es correcta, se considera una coincidencia exitosa. La precisión total del modelo se obtiene dividiendo el número de coincidencias correctas por el número total de descripciones en el corpus.

Definiciones

- D_C: Lista de descripciones del cliente.
- D_{BMC}: Lista de descripciones del BMC.
- $S(d_c, d_{bmc})$: Índice de similitud entre la descripción del cliente d_c y la descripción del BMC d_{bmc}.

Proceso de Evaluación

Para cada $d_c \in D_C$, buscamos $d_{bmc} \in D_{BMC}$ que maximice $S(d_c, d_{bmc})$:

$$\text{match}(d_c) = \arg \max_{d_{bmc} \in D_{BMC}} S(d_c, d_{bmc})$$

Conteo de Coincidencias Correctas

Contamos las coincidencias correctas:

$$N_{\text{correct}} = \sum_{d_c \in D_C} (\text{match}(d_c) \text{ coincide correctamente con } d_c)$$

Cálculo del Porcentaje de Alineación Correcta

$$P_{\text{correct}} = \frac{N_{\text{correct}}}{|D_C|} \times 100\,\%$$

Este método personalizado de evaluación es adecuado para nuestra tarea, ya que nos permite medir directamente la capacidad del modelo para asignar correctamente los códigos de productos basándose en la similitud textual, lo cual es el objetivo principal de nuestro proyecto.

3.1.2. Evaluación de la Correspondencia Top 3

Además de la evaluación de la correspondencia directa, evaluamos también con k=3, es decir, verificamos si la descripción correcta del BMC se encontraba dentro de las tres descripciones más similares. Esto incluyó los siguientes pasos:

1. Obtener los índices de las similitudes más altas para cada descripción del cliente, ordenadas en orden descendente.

2. Identificar las tres descripciones únicas del BMC más similares para cada cliente.
3. Verificar si la descripción correcta del BMC se encontraba dentro de estas tres descripciones.

Este enfoque permitió evaluar la precisión del modelo en un contexto más flexible, aumentando las posibilidades de una correspondencia correcta y de utilidad en un contexto de producción.

3.1.3. Similitud Jaccard

Para explorar alternativas en calculo de similitud que pudiesen mejorar los resultados que obtuvimos, quisimos explorar nuevas formas de medir la similitud basándonos en la cantidad de palabras compartidas entre descripciones.La *Similitud Jaccard* es una métrica utilizada para medir la similitud entre dos conjuntos. Se define como el tamaño de la intersección dividido por el tamaño de la unión de los conjuntos. En el contexto de NLP, se usa para comparar descripciones de texto convirtiéndolas en conjuntos de palabras y calculando cuántas palabras comparten:

$$\text{Similitud Jaccard} = \frac{|A \cap B|}{|A \cup B|}$$

Donde A y B son conjuntos de palabras de dos descripciones. Esta métrica es útil porque es intuitiva y fácil de calcular, y proporciona una medida clara de la similitud basada en el contenido compartido [21].

Implementación de la Similitud Jaccard

Para implementar la similitud Jaccard, seguimos estos pasos:

1. **Conversión de Texto a Conjuntos**: Convertimos cada descripción de producto en un conjunto de palabras clave (tokens). Esto implica lematizar las palabras, eliminando palabras vacías, signos de puntuación y palabras que no sean alfabéticas.
2. **Precomputación de Conjuntos**: Precomputamos los conjuntos de palabras para cada descripción tanto de las SC como del BMC. Esto facilita el cálculo posterior de la similitud Jaccard. Para este fin utilizamos la función `text_to_set` para convertir cada descripción a un conjunto de palabras clave.
3. **Cálculo de la Similitud de Jaccard**: Calculamos la similitud de Jaccard entre cada descripción del cliente y todas las descripciones del BMC. Esto se hace calculando la intersección y la unión de los conjuntos y dividiendo el tamaño de la intersección por el tamaño de la unión.
4. **Evaluación de Precisión**: Para cada descripción del cliente, identificamos las descripciones del BMC con la mayor similitud de Jaccard. Evaluamos cuántas veces la descripción correcta del BMC se encuentra dentro de las más similares.

3.2. Implementación del Modelo de Word Embeddings: Word2Vec

3.2.1. Proceso de Entrenamiento

Se usó como primera aproximación el modelo Word2Vec para generar word embeddings con las las descripciones tokenizadas de los productos, con el objetivo de capturar

las relaciones semánticas entre las palabras que en ellas se encontraban. Para esto se procedió de la siguiente manera:

1. Se combinaron y procesaron todas las descripciones de las SC y del BMC.

2. Se entrenó el modelo Word2Vec con estas descripciones, configurando parámetros como el tamaño del vector, la ventana de contexto y el conteo mínimo de palabras.

Para entender mejor como estos parámetros pueden afectar al entrenamiento tenemos como aclaración en [17] que:

- **Vector size:** El parámetro *vector_size* en Word2Vec define el número de dimensiones del espacio vectorial en el cual se mapearán las palabras. Un mayor tamaño de vector puede llevar a modelos más precisos y detallados, ya que cada palabra se representa con más características numéricas. Sin embargo, valores más grandes requieren más datos de entrenamiento para ser efectivos,el cual no es nuestro caso.

- **Window size:** El parámetro *window* especifica el tamaño de la ventana de contexto, es decir, el número de palabras antes y después de la palabra objetivo que se considerarán durante el entrenamiento. Un valor más grande permite capturar más contexto, pero también puede incluir palabras menos relevantes.

- **Min Count:** El parámetro *min_count* establece el umbral mínimo de frecuencia para que una palabra sea incluida en el vocabulario del modelo. Las palabras que aparecen menos veces que este umbral serán ignoradas. Este parámetro ayuda a filtrar palabras raras y errores tipográficos que pueden no ser útiles para el entrenamiento, Al ser el corpus tan variado y al eliminar palabras conocidas como *stopwords* se debería de mantener este valor bajo.

Parámetros de entrenamiento primera iteración

Parámetro	**Valor**
Vector Size	150
Window Size	7
Min Count	3
Workers (hilos de CPU)	4

Cuadro 3.1: Parámetros de entrenamiento del Modelo Word2Vec 1

Parámetros de entrenamiento segunda iteración

Parámetro	**Valor**
Vector Size	100
Window Size	5
Min Count	1
Workers (hilos de CPU)	4

Cuadro 3.2: Parámetros de entrenamiento del Modelo Word2Vec 2

Parámetros de entrenamiento tercera iteración

Parámetro	Valor
Vector Size	300
Window Size	10
Min Count	5
Workers (hilos de CPU)	4

Cuadro 3.3: Parámetros de entrenamiento del Modelo Word2Vec 3

Después de observar resultados, para las pruebas subsiguientes se utilizaron los parámetros de entrenamiento de la iteración 1.

3.2.2. Generación de Vectores de Documento

Una vez entrenado el modelo Word2Vec, se generaron vectores para cada descripción de producto. Estos vectores representan las descripciones en un espacio vectorial, lo que facilita la comparación y el cálculo de similitudes.

1. Se creó una función para obtener vectores de documento promediando los vectores de las palabras en la descripción.
2. Se aplicó esta función para generar vectores tanto para las descripciones de las SC como para las descripciones del BMC.

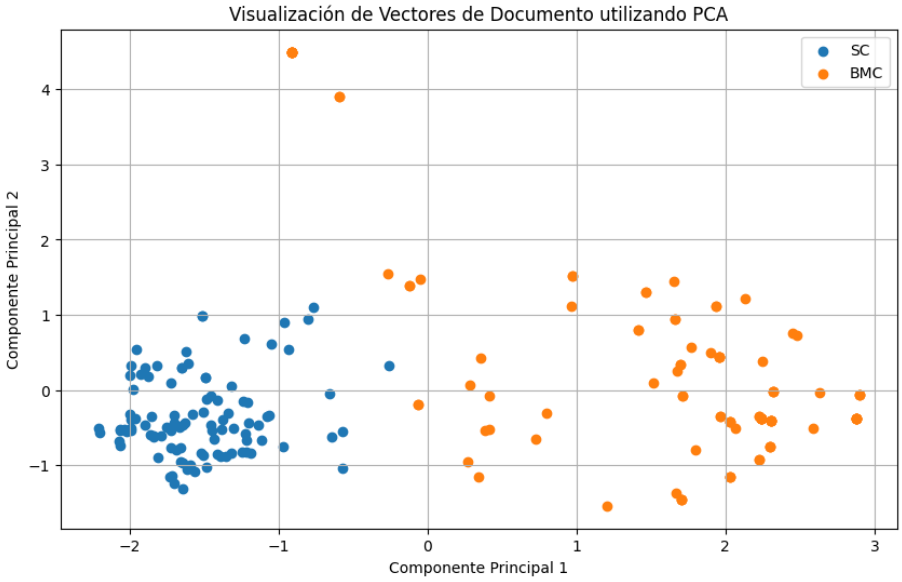

Figura 3.1: Visualización de Vectores de Documento utilizando PCA.

La figura 3.1 permite visualizar cómo los vectores de documento, reducidos a dos dimensiones, se distribuyen en el espacio y cómo se diferencian las descripciones de los productos de BMC y las SC. Los puntos azules y naranjas tienden a agruparse en diferentes áreas del gráfico, indicando que las descripciones de BMC y las SC tienen características distintivas que PCA puede separar en el espacio reducido tambien se observa que en algunas áreas muestran superposición entre los dos grupos, lo que puede indicar similitudes entre ciertas descripciones de BMC y las SC.

3.2.3. Cálculo de Similitudes

Con los vectores generados, se calculó la similitud de coseno entre las descripciones de las SC y las descripciones del BMC. Esta métrica mide la similitud entre dos vectores, proporcionando un valor entre -1 y 1. Para esto:

1. Se alinearon correctamente los vectores generados para cada par de descripciones.
2. Se calculó la similitud de coseno para cada par de vectores alineados.
3. Se almacenaron los valores de similitud para análisis posteriores.

3.2.4. Análisis de Similitud Promedio

Para visualizar la distribución de las similitudes calculadas, se creó un histograma. Este gráfico permitió observar la frecuencia de diferentes valores de similitud y analizar el desempeño del modelo.

1. Cálculo de la similitud promedio entre todas las parejas de descripciones.
2. Creación de un histograma para visualizar la distribución de las similitudes.
3. Identificación de la media de similitudes para evaluar el rendimiento general del modelo.

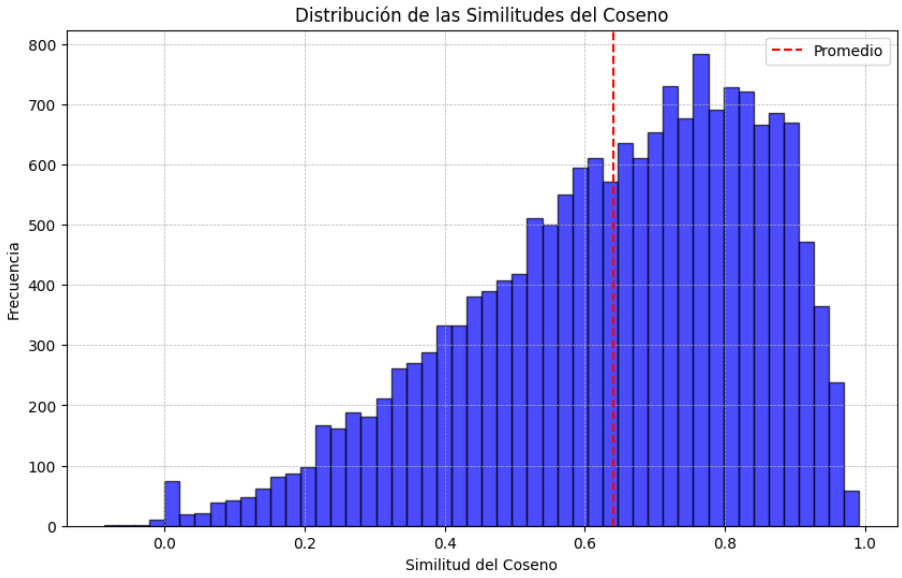

Figura 3.2: Similitud promedio entre todas las parejas de descripciones.

3.2.5. Optimización y Reducción de Redundancias

Para optimizar el proceso de cálculo de similitudes y reducir redundancias, se implementó una técnica para identificar y eliminar descripciones duplicadas en las descripciones del BMC, ya que muchas se repetían. Esto permitió disminuir el número de cálculos innecesarios y mejorar la eficiencia del modelo.

Para este fin se usó *OrderedDict* la cual es una clase en la biblioteca de colecciones de Python que mantiene el orden en el que se añaden los elementos. A diferencia de los diccionarios estándar en versiones anteriores de Python, que no garantizaban ningún orden, OrderedDict recuerda el orden de inserción de los elementos.[20]. De esta manera se siguieron los siguientes pasos:

1. Se creó una función para convertir listas de tokens en tuplas, permitiendo su uso en estructuras que eliminan duplicados.
2. Se utilizaron `OrderedDict` para mantener el orden de las descripciones mientras se eliminaban duplicados.
3. Se generaron vectores únicos para las descripciones del BMC utilizando el modelo Word2Vec previamente entrenado.

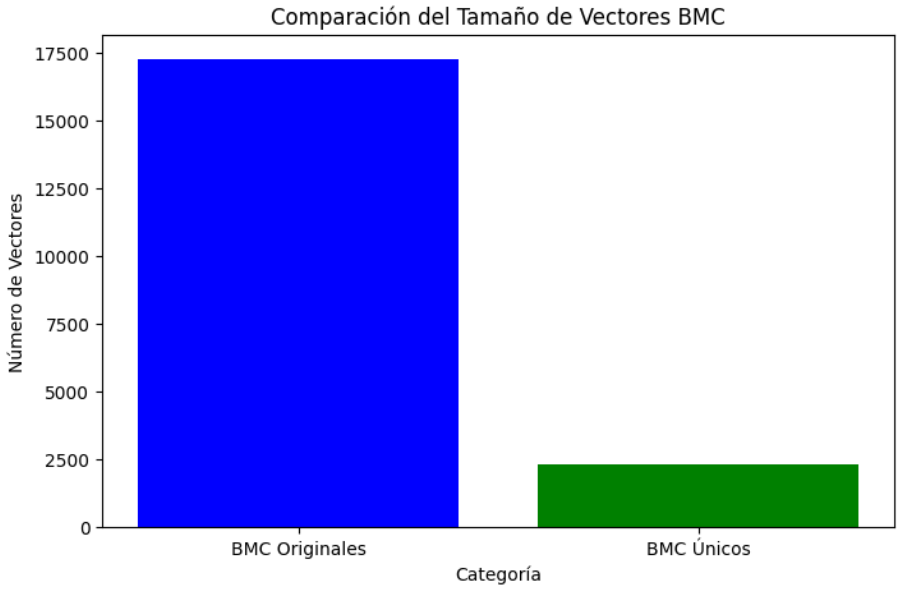

Figura 3.3: Comparación de cantidad de vectores después de reducción de redundancias.

El resultado de esta optimización mostró una reducción significativa en el número de vectores de descripciones del BMC, lo que se visualiza en la figura 3.3.

3.2.6. Cálculo de Similitudes Mejorado

Una vez generados los vectores únicos, se procedió a calcular las similitudes de coseno entre las descripciones de las SC y las descripciones del BMC. Este proceso permitió identificar la descripción del BMC más similar para cada descripción del cliente en un tiempo menor al original.

1. Se utilizaron los vectores únicos generados previamente para las descripciones del BMC.
2. Se calcularon las similitudes de coseno entre los vectores de las descripciones de las SC y los vectores únicos del BMC.
3. Se identificaron los índices de las descripciones del BMC con mayor similitud para cada cliente.

3.2.7. Evaluación de la Correspondencia

1. **Evaluación directa:** Se evaluó la correspondencia de las descripciones utilizando los índices de similitud máxima obtenidos. Se verificó cuántas veces la descripción del BMC más similar correspondía correctamente a la descripción del cliente.

2. **Evaluación Top 3:** Se realizó una evaluación adicional para verificar si la descripción correcta del BMC se encontraba dentro de las tres descripciones más similares.

3.2.8. Utilización de N-gramas

Para mejorar la agrupación de términos que frecuentemente aparecen juntos y reducir la diversidad del corpus, se implementaron N-gramas. Los *N-gramas* son secuencias contiguas de n elementos de un texto, donde n puede ser cualquier número entero positivo. En el contexto del procesamiento de lenguaje natural (NLP), estos elementos suelen ser palabras o caracteres. Los N-gramas se utilizan para modelar el contexto local de un texto y capturar patrones comunes.

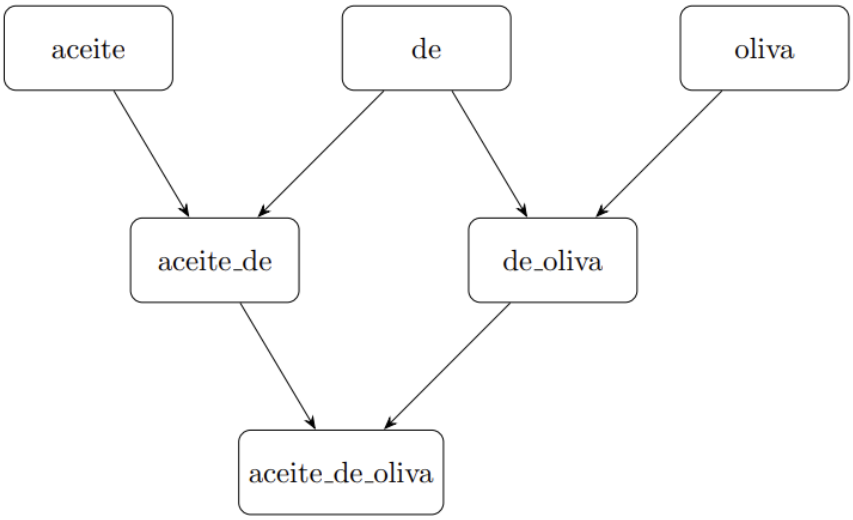

Figura 3.4: Ejemplo de N-gramas.

Ejemplo:

- Unigrama: {aceite, de, oliva}
- Bigramas: {aceite_de, de_oliva}
- Trigrama: {aceite_de_oliva}

Aunque son conceptualmente simples, como se denota en el ejemplo y en la figura 3.4, los N-gramas pueden capturar dependencias locales en el texto, lo que los hace valiosos para muchas aplicaciones [18].

1. Se utilizaron las bibliotecas `Phrases` y `Phraser` de Gensim para detectar y fusionar frases comunes en las descripciones.

2. Ya que en las descripciones procesadas se eliminaron las *stop words*, se aplicó el modelo de bigramas a las descripciones para identificar y agrupar términos frecuentes.

3. Se entrenó un nuevo modelo Word2Vec utilizando las descripciones con los N-gramas aplicados.

3.2.9. Generación y Evaluación de Vectores con N-gramas

Tras la creación del modelo Word2Vec con N-gramas, se generaron vectores para las descripciones de las SC y del BMC. Posteriormente, se calcularon las similitudes de coseno para evaluar la correspondencia entre las descripciones.

1. Generar vectores para las descripciones de las SC y del BMC utilizando el nuevo modelo Word2Vec con N-gramas.

2. Calcular las similitudes de coseno entre los vectores de las SC y los vectores del BMC.

3. Evaluar la correspondencia y calcular la similitud promedio directa.

Los resultados mostraron que la utilización de N-gramas mejoró la agrupación de términos, aunque la similitud promedio directa no fue tan alta como se esperaba.

3.2.10. Prueba de Procesamiento de Descripciones Unidas

Se llevó a cabo un experimento para procesar las descripciones de las SC y del BMC juntas, es decir, tratar cada tupla de descripciones como una sola cadena de texto. La hipótesis era que, combinando las descripciones, el modelo podría aprender mejor las relaciones entre términos.

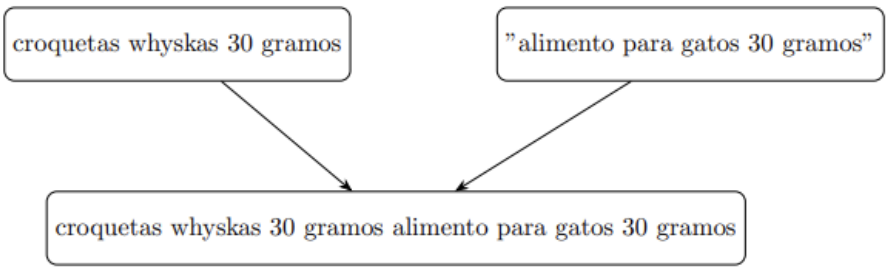

Figura 3.5: Ejemplo de union de descripciones.

Este enfoque no resultó efectivo, ya que la similitud promedio directa fue la más baja hasta el momento, indicando que la combinación de descripciones no ayudó al modelo a entender mejor las relaciones entre términos.

3.2.11. Entrenamiento con Lemmatización

Como ultima prueba con Word2Vec de las descripciones estandar, se entrenó un modelo utilizando lematización, una técnica que reduce las palabras a su forma base o lema. Con esto se buscó permite agrupar variantes de una palabra, para mejorar la precisión de los análisis. los pasos que utilizamos para lograr esto fueron:

1. Preprocesar las descripciones utilizando lemas. Estos ya las teníamos almacenadas gracias a nuestro preprocesamiento.

2. Entrenar un modelo Word2Vec con las descripciones lematizadas.

3. Generar vectores para las descripciones de las SC y de la BMC.

4. Evaluar la correspondencia y calcular la similitud promedio directa.

5. Calcular las similitudes de coseno entre los vectores y evaluar la correspondencia.

Los resultados mostraron una mejora en la similitud promedio directa; sin embargo, esta mejora no se ve reflejada en la correspondencia correcta dentro de los tres primeros resultados más similares.

3.2.12. Aproximación con Descripciones Subyacentes

Para mejorar la precisión en la asignación de códigos de productos, se intentó una aproximación más simple utilizando descripciones subyacentes proporcionadas por el BMC. Estas descripciones subyacentes son versiones simplificadas de las descripciones completas (ejemplo en la figura 3.6) lo que reduce la diversidad del corpus y potencialmente mejora la precisión de las asignaciones.

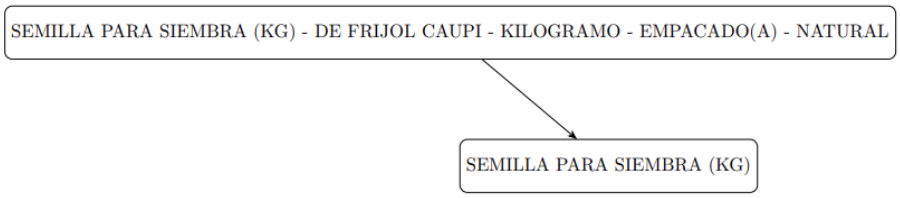

Figura 3.6: Ejemplo de descripciones subyacentes.

Para este fin se cargaron y prepararon los datos, incluyendo una nueva columna para las descripciones subyacentes. Las descripciones subyacentes fueron tokenizadas y lematizadas para su procesamiento.

3.2.13. Entrenamiento con Tokens de las descripciones subyacentes

El modelo Word2Vec se entrenó utilizando las descripciones tokenizadas de las SC y del BMC, incluyendo las descripciones subyacentes.

1. Se combinaron y procesaron todas las descripciones de las SC y las descripciones subyacentes del BMC.

2. Se entrenó el modelo Word2Vec con estas descripciones, configurando parámetros como el tamaño del vector, la ventana de contexto y el conteo mínimo de palabras.

3. Se generaron vectores para las descripciones de las SC y las descripciones subyacentes del BMC.

3.2.14. Optimización y Reducción de Redundancias de descripciones subyacentes

Para optimizar el proceso, se identificaron y eliminaron descripciones duplicadas en las descripciones subyacentes del BMC, al igual que se hizo con las descripciones originales con el fin de evitar trabajo de computo innecesario.

1. Se creó una función para convertir listas de tokens en tuplas, permitiendo su uso en estructuras que eliminan duplicados.
2. Se utilizaron `OrderedDict` para mantener el orden de las descripciones mientras se eliminaban duplicados.
3. Se generaron vectores únicos para las descripciones subyacentes del BMC.

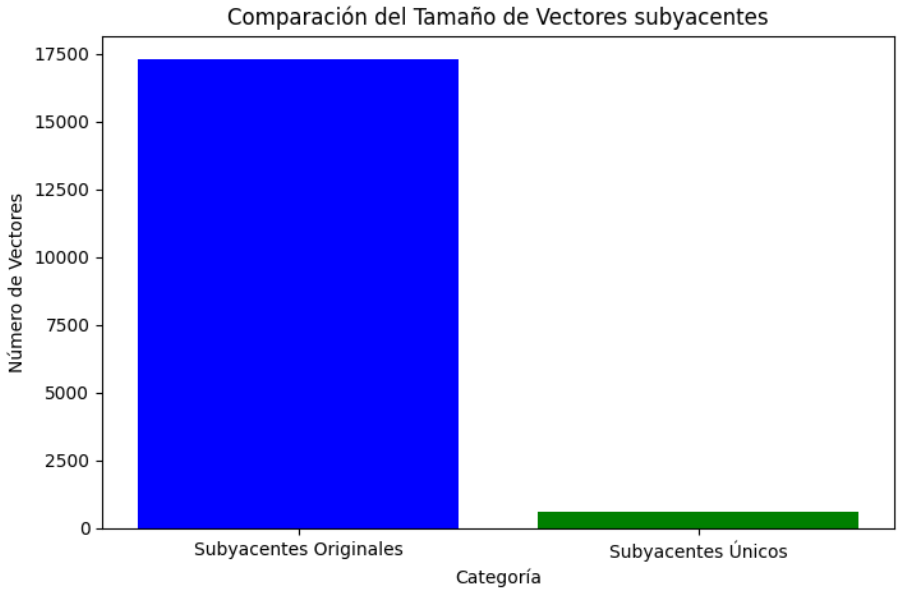

Figura 3.7: Comparación de cantidad de vectores subyacentes.

Después de realizar este ejercicio se logro pasar de 17288 a 612 vectores únicos, como se observa en la figura 3.7. Esto representa una mejora substancial en el tiempo de computo de las similitudes, así mismo como una disminución en la complejidad de la tarea en sí.

3.2.15. Cálculo de Similitudes

Con los vectores generados, se calculó la similitud de coseno entre cada descripción del cliente y las descripciones subyacentes del BMC. Para esto se realizaron los siguientes pason:

1. Calcular las similitudes de coseno directamente para parejas alineadas de descripciones.
2. Evaluar la correspondencia y calcular la similitud promedio directa.
3. Crear un histograma para visualizar la distribución de las similitudes.

En la figura 3.8 podemos observar como la similitud promedio aumenta usando descripciones subyacentes.

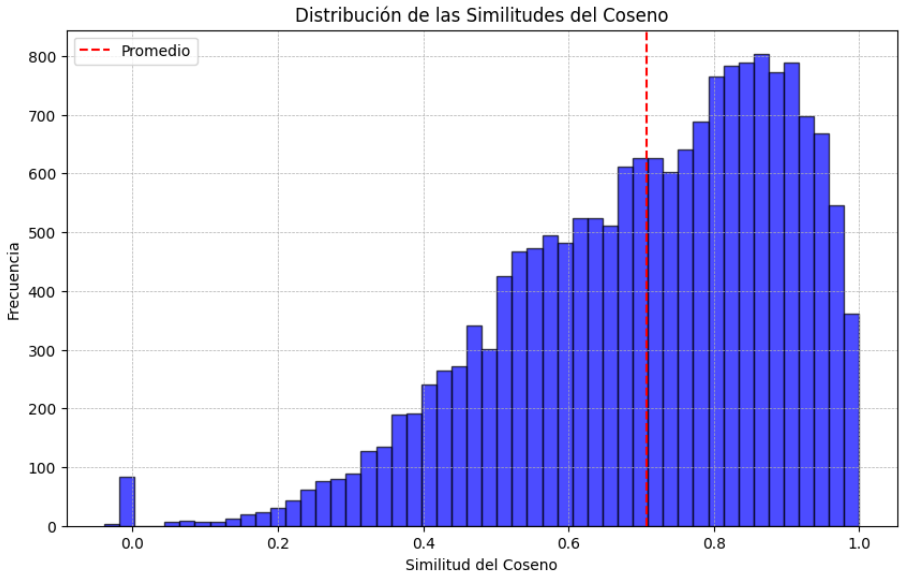

Figura 3.8: Similitud promedio con descripciones subyacentes.

3.2.16. Evaluación de la Correspondencia

Se evaluó la precisión del modelo verificando cuántas descripciones del cliente coincidían correctamente con la descripción subyacente del BMC más similar.

1. Calcular las similitudes de coseno para cada vector de cliente contra todos los vectores únicos de descripciones subyacentes del BMC.

2. Identificar el índice de la mayor similitud y almacenar los resultados.

3. Contar el número de coincidencias correctas y calcular el porcentaje de alineación correcta.

3.3. Evaluación con Modelo Preentrenado: spaCy

Ante el rendimiento de los modelos Word2Vec entrenados directamente con nuestro corpus se contempló el uso de bibliotecas adicionales y modelos preentrenados con vectores de palabras basados en *Word2Vec* o *GloVe*. *spaCy* es una biblioteca de procesamiento de lenguaje natural de código abierto en Python, diseñada para ser rápida y eficiente. Incluye modelos preentrenados para varios idiomas que pueden realizar tareas como tokenización, lematización, etiquetado gramatical, análisis sintáctico y reconocimiento de entidades nombradas[19].

Para evaluar el rendimiento del modelo preentrenado de spaCy en el objetivo de nuestro proyecto, se siguieron varios pasos para calcular las similitudes entre las descripciones de las SC y las descripciones del BMC. El objetivo fue comparar el desempeño de este modelo con los modelos entrenados anteriormente.

3.3.1. Carga del Modelo Preentrenado

El modelo `es_core_news_md` de *spaCy* es específico para el español y está optimizado para manejar texto en este idioma. Al igual que *spaCy*, es conocido por su facilidad de uso y su capacidad para integrarse con otras bibliotecas y herramientas de NLP. [19].

1. Se descargó e instaló el modelo el modelo preentrenado `es_core_news_md` de spaCy.
2. Se cargó el modelo preentrenado en el entorno de trabajo.

3.3.2. Generación de Vectores

Se generaron vectores para las descripciones de las SC y del BMC utilizando el modelo de spaCy. Esta etapa involucró:

1. Precalcular los vectores para las descripciones de las SC.
2. Identificar y generar vectores para las descripciones únicas del BMC.

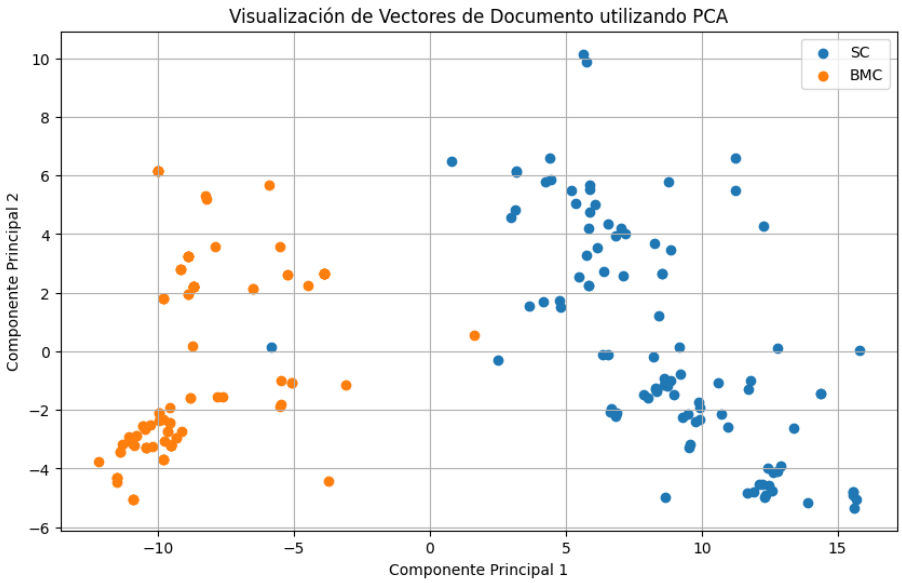

Figura 3.9: Visualización de Vectores de Documento utilizando PCA.

La figura 3.9 permite visualizar cómo los vectores de documento generados usando *spaCy*, reducidos a dos dimensiones, se distribuyen en el espacio y cómo se diferencian las descripciones de los productos de BMC y las SC. A diferencia de los vectores generado por *Word2Vec* se nota una mayor distancia entre los puntos anaranjados y azules que representan los distintos tipos de descripciones del corpus, con unas cuantas superposiciones minimas que nos permiten entrever una menor similitud promedio entre descripciones de BMC y las SC.

3.3.3. Cálculo de Similitudes

Con los vectores generados, se calculó la similitud de coseno entre cada descripción del cliente y todas las descripciones únicas del BMC.

1. Calcular las similitudes de coseno para cada vector de cliente contra todos los vectores únicos del BMC.
2. Identificar el índice de la mayor similitud y almacenar los resultados.

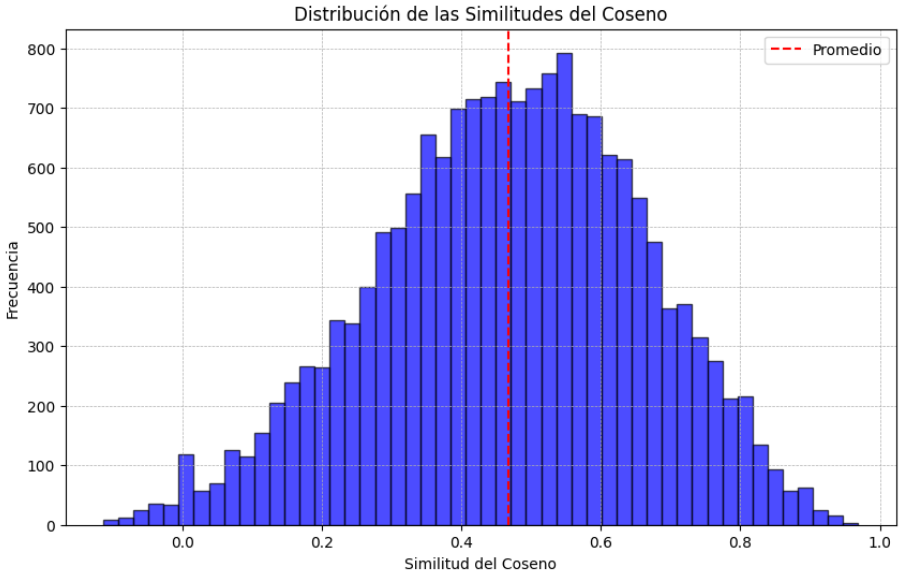

Figura 3.10: Similitud promedio directa con spaCy.

Para visualizar la distribución de las similitudes, se creó un histograma (figura 3.10) que muestra la frecuencia de diferentes valores de similitud de coseno. Este gráfico ayudó a entender mejor la distribución de los resultados y la efectividad del modelo spaCy.

3.3.4. Evaluación con Top k=1

Se evaluó la precisión del modelo de spaCy verificando cuántas descripciones del cliente coincidían correctamente con la descripción del BMC más similar.

1. Contar el número de coincidencias correctas y calcular el porcentaje de alineación correcta.
2. Imprimir detalles individuales de las coincidencias para verificar los resultados.

3.3.5. Evaluación Top k=3

Para una evaluación más detallada, se realizó una prueba adicional para verificar si la descripción correcta del BMC se encontraba dentro de las tres descripciones más similares.

1. Calcular las similitudes de coseno para cada vector de cliente contra todos los vectores únicos del BMC.
2. Identificar los índices de las tres mayores similitudes y verificar si la descripción correcta del BMC estaba entre ellas.

Podemos observar que usando `spaCy` y el modelo `es_core_news_md` se obtiene una similitud promedio ligeramente inferior a la obtenida con `Word2Vec`

3.3.6. Evaluación con Descripciones Subyacentes y spaCy

Después de intentar una aproximación con descripciones subyacentes y un modelo Word2Vec, se realizó una evaluación adicional utilizando el modelo preentrenado de spaCy. El objetivo fue comparar los resultados obtenidos con las descripciones subyacentes más simples con los embeddings generados por spaCy.

Evaluación Top k =1 : Se utilizó el modelo preentrenado de spaCy para generar vectores y evaluar la precisión de las asignaciones de códigos de productos. Este proceso implicó:

1. Cargar el modelo preentrenado en el entorno de trabajo y cargar el dataframe actualizado que incluye las descripciones subyacentes.
2. Generar vectores para las descripciones de las SC y las descripciones subyacentes de la BMC.
3. Calcular las similitudes de coseno entre cada descripción del cliente y todas las descripciones subyacentes de la BMC.
4. Evaluar la precisión de las asignaciones.

Evaluación Top k =3 : Para cada descripción del cliente, se calcularon las tres mayores similitudes con las descripciones subyacentes del BMC. Este proceso permitió evaluar si la descripción correcta del BMC se encontraba entre las tres más similares.

1. Calcular las similitudes de coseno para cada descripción del cliente contra todas las descripciones subyacentes del BMC.
2. Identificar los índices de las tres mayores similitudes.
3. Verificar si la descripción correcta del BMC se encontraba dentro de las tres más similares.

Este método mostró una precisión mejorada en comparación con el uso de descripciones completas y complejas.

3.3.7. Similitud Combinada: Coseno + Jaccard

Finalmente, se probó una combinación de la similitud de coseno y la similitud de Jaccard. Esta combinación busca aprovechar las fortalezas de ambas métricas para mejorar la precisión general.

Implementación de la Similitud Combinada

1. **Conversión y Precomputación**: Se convierten las descripciones a vectores y conjuntos de palabras clave.
2. **Cálculo de Similitudes**: Calculo de tanto la similitud de coseno como la de Jaccard para cada par de descripciones.
3. **Combinación de Puntuaciones**: Se combinan las puntuaciones de similitud promediándolas.
4. **Evaluación de Precisión**: Contamos el número de coincidencias correctas dentro de las tres más similares y calculamos la precisión.

Con estos enfoques, buscamos mejorar la precisión y eficiencia del modelo de asignación automática de códigos de productos, proporcionando una comparación exhaustiva de diversas métricas de similitud.

3.4. Implementación de Redes Siamesas: SBERT

Para mejorar la asignación automática de códigos de productos, se implementaron redes siamesas utilizando embeddings generados por el modelo preentrenado Sentence-BERT (SBERT). Las redes siamesas son especialmente útiles para tareas de comparación y clasificación de similitud de textos.

3.4.1. Preparación del Entorno

1. **Instalación de Dependencias** Se instalaron las bibliotecas `sentence-transformers` y `torch` necesarias para cargar y utilizar modelos de SBERT. Estas bibliotecas permiten el manejo de modelos de lenguaje natural preentrenados y la realización de operaciones con tensores, respectivamente.

2. **Carga de Datos**: Se cargaron los datos desde un archivo Excel, incluyendo las columnas relevantes para el análisis, como las descripciones de productos de las SC y las descripciones de la BMC.

3. **Carga del Modelo SBERT:** Se cargaron dos modelos preentrenados de SBERT, cada uno con características distintas.

   a) **Modelo Multilingüe:**
   - **Modelo:** `distiluse-base-multilingual-cased-v2`
   - **Características:** Este modelo es recomendado por la documentación de SBERT por su capacidad para manejar múltiples idiomas, lo que lo hace ideal para aplicaciones en contextos multilingües [31].

   b) **Modelo Especializado en Español:**
   - **Modelo:** `sentence_similarity_spanish_es`
   - **Características:** Este modelo ha ganado popularidad en la plataforma Hugging Face y está específicamente entrenado para tareas en español, ofreciendo una mayor precisión en comparación con los modelos multilingües en contextos hispanohablantes [32].

3.4.2. Generación de Embeddings con SBERT

Se utilizaron los modelos preentrenados de SBERT para generar embeddings tanto para las descripciones de las SC como para las descripciones de la BMC.

1. **Evaluación de los Modelos:** Ambos modelos fueron evaluados para determinar cuál proporcionaba mejores resultados en términos de precisión en la similitud entre descripciones. El proceso de evaluación incluyó varios pasos clave:

   - **Generación de Embeddings:** Se calcularon los embeddings para las descripciones únicas del BMC y las descripciones de las SC utilizando ambos modelos.
   - **Cálculo de Similitudes:** Se calcularon las similitudes coseno entre los embeddings generados por las SC y los embeddings únicos del BMC.

- **Comparación de Resultados:** Se compararon los resultados obtenidos de ambos modelos en términos de precisión para identificar cuál ofrecía mejores asignaciones de códigos de productos.

2. **Selección del Modelo:**Al observar que el modelo especializado en español proporcionaba mejores resultados, se decidió utilizar este modelo para el resto de las pruebas. Este modelo ofreció una mayor precisión en las asignaciones de descripciones, superando significativamente al modelo multilingüe.

3.4.3. Cálculo de Embeddings con descripciones subyacentes

Una vez seleccionado el modelo `sentence_similarity_spanish_es`, se procedió a calcular los embeddings para las descripciones de las SC y las descripciones subyacentes del BMC:

- **Embeddings para Descripciones Subyacentes del BMC:** Se generaron embeddings para todas las descripciones subyacentes del BMC.
- **Embeddings para Descripciones de las SC:** Se calcularon los embeddings para las descripciones de las SC utilizando el modelo seleccionado, permitiendo así una comparación directa con las descripciones subyacentes del BMC.

3.4.4. Cálculo y evaluación de Similitudes

Se calcularon las similitudes coseno entre los embeddings de las descripciones de las SC y los embeddings subyacentes del BMC para evaluar la precisión de las asignaciones. Este proceso incluyó la verificación de la posición de la descripción correcta en las listas ordenadas por similitud. Al igual que con en el modelo de embeddings se hizo una comparación de precisión de asignaciones correctas directas y con descripciones en el top 3 más similar.

3.4.5. Combinación de Similitud Coseno + Jaccard

Para aprovechar al máximo las ventajas de ambas métricas, se combinaron las similitudes coseno y Jaccard para obtener una puntuación combinada y evaluar la precisión de las asignaciones.

1. **Conversión a Conjuntos:** Las descripciones de las SC y las descripciones subyacentes se convirtieron en conjuntos de palabras clave para calcular la similitud Jaccard.
2. **Cálculo de Similitud Jaccard:** Se calcularon las similitudes Jaccard entre cada conjunto de palabras de las descripciones de las SC y las descripciones subyacentes únicas.
3. **Combinación de similitudes:** Se promediaron las puntuaciones para obtener una similitud combinada.

3.4.6. Evaluación de Precisión de Similitud Combinada

Se determinó la precisión para las descripciones más similares (directas y Top 3) basadas en las puntuaciones combinadas, proporcionando una evaluación más robusta de la similitud. La combinación de similitudes coseno y Jaccard mostró una mejora en la precisión de las asignaciones, evidenciando la eficacia de utilizar múltiples métricas para la comparación de textos.

3.5. Fine-tuning del Modelo SBERT

El proceso de ajuste fino del modelo SBERT se realizó para mejorar la precisión de las asignaciones de códigos de productos. Este proceso implicó varias iteraciones de entrenamiento, ajustando hiperparámetros y utilizando diferentes enfoques para la generación de ejemplos de entrenamiento. Algunos de los hiper parametros clave fueron los *epoch* (epocas) y *Loss Function* (funcion de perdida).

Epoch

En el contexto del aprendizaje automático, un *epoch* se refiere a un ciclo completo de entrenamiento en el que el modelo pasa por todo el conjunto de datos de entrenamiento una vez. Durante cada epoch, el modelo realiza una pasada hacia adelante (predicción) y una pasada hacia atrás (actualización de pesos y cálculo del error). La importancia de los epochs radica en su relación con el rendimiento del modelo: muy pocos epochs pueden resultar en un modelo subentrenado (*underfitting*), mientras que demasiados pueden llevar al sobreentrenamiento (*overfitting*) [22].

Loss Function

Una *Loss Function* es una métrica que evalúa el desempeño del modelo durante el entrenamiento, ajustando los pesos del modelo para minimizar esta pérdida. En redes siamesas se utilizan diferentes funciones de pérdida dependiendo del objetivo específico. En nuestro caso usamos primero *CosineSimilarityLoss* y después usamos *ContrastiveLoss* para comparar resultados y ver como esta afectaba la precisión del modelo.

Batch Size

El *batch size* o tamaño de lote es el número de muestras que se procesan en conjunto antes de actualizar los parámetros del modelo. En este caso, se utilizó un batch size de 16. Esto significa que en cada iteración del entrenamiento, el modelo procesa 16 descripciones de productos a la vez antes de ajustar sus pesos. Este valor es una compensación entre la eficiencia computacional y la estabilidad del modelo: lotes más grandes pueden acelerar el entrenamiento, pero pueden reducir la capacidad del modelo para generalizar en nuevos datos[26] [27].

Warmup Steps

Se utilizaron 100 pasos de calentamiento ("warmup steps"). Durante estos pasos, la tasa de aprendizaje se incrementa gradualmente desde un valor inicial muy bajo hasta el valor deseado. Este enfoque ayuda a estabilizar el entrenamiento y previene grandes oscilaciones en los gradientes que podrían dificultar la convergencia del modelo[28].

3.5.1. Primera Iteración: Fine-tuning Inicial

Preparación del Entorno

1. **Instalación de Dependencias:** Se instalaron las bibliotecas `sentence-transformers` y `torch` necesarias para el entrenamiento del modelo SBERT.

2. **Carga de Datos:** Se cargaron los datos desde un archivo Excel, incluyendo las columnas relevantes para el análisis.

Ajuste Fino del Modelo SBERT

1. **Configuración del Modelo:**
   - Se utilizó el modelo preentrenado `sentence_similarity_spanish_es`.
   - Se añadió una capa de pooling para obtener embeddings fijos de tamaño.
2. **Creación de Datos de Entrenamiento:**
   - Se generaron ejemplos de entrenamiento positivos a partir de las descripciones de las SC y sus correspondientes descripciones de BMC.
3. **Entrenamiento del Modelo:**
   - Se entrenó el modelo utilizando *CosineSimilarityLoss*, una técnica que ajusta los parámetros del modelo para maximizar la similitud entre descripciones relacionadas.
4. **Evaluación de Precisión:**
   - Se evaluó la precisión del modelo ajustado, verificando su capacidad para asignar correctamente los códigos de productos como se ha hecho previamente.

Parámetros utilizados para entrenamiento

Parámetro	**Valor**
Batch Size	16
Función de Pérdida	CosineSimilarityLoss
Epochs	1
Warmup Steps	100
Tiempo de Entrenamiento	2.71 minutos

Cuadro 3.4: Parámetros Utilizados para el Entrenamiento del Modelo SBERT Fine-tuned en la primera iteración

3.5.2. Segunda Iteración: Aumento de Epochs y Ejemplos Negativos

Para mejorar los resultados, se realizó una segunda iteración de ajuste fino con las siguientes modificaciones:

1. **Aumento de Épocas:** Se incrementó el número de epochs a tres, permitiendo al modelo aprender más profundamente de los datos.
2. **Generación de Ejemplos Negativos:** Se añadieron ejemplos negativos de entrenamiento emparejando descripciones de las SC con descripciones incorrectas de BMC, lo que ayuda al modelo a distinguir mejor entre descripciones similares y no relacionadas.

Ajuste Fino del Modelo SBERT

1. **Configuración del Modelo:** Se reutilizó el modelo `sentence_similarity_spanish_es` con una capa de pooling.
2. **Creación de Datos de Entrenamiento:** Se generaron tanto ejemplos positivos como negativos para el entrenamiento.
3. **Entrenamiento del Modelo:** Se entrenó el modelo utilizando el loss de similitud coseno.
4. **Evaluación de Precisión:** Se evaluó la precisión del modelo ajustado.

Parámetros utilizados para entrenamiento en la segunda iteración

Parámetro	**Valor**
Batch Size	16
Función de Pérdida	CosineSimilarityLoss
Epochs	3
Warmup Steps	100
Tiempo de Entrenamiento	8.13 minutos

Cuadro 3.5: Parámetros Utilizados para el Entrenamiento del Modelo SBERT Fine-tuned en la segunda iteración

3.5.3. Tercera Iteración: Uso de Contrastive Loss

Para seguir mejorando los resultados, se realizó una tercera iteración de ajuste fino utilizando *ContrastiveLoss*. El *Contrastive Loss* se ha demostrado efectivo en la tarea de emparejamiento y comparación de pares de datos similares y disímiles, optimizando la distancia entre vectores de características aprendidos. [33].

1. **Configuración del Modelo:** Se utilizó nuevamente el modelo `sentence_similarity_spanish_es` con una capa de pooling.
2. **Creación de Datos de Entrenamiento:** Se generaron ejemplos positivos y negativos para el entrenamiento.
3. **Entrenamiento del Modelo:** Se entrenó el modelo utilizando el *ContrastiveLoss*.
4. **Evaluación de Precisión:** Se evaluó la precisión del modelo ajustado.

Parámetros utilizados para entrenamiento en la tercera iteración

Parámetro	**Valor**
Batch Size	16
Función de Pérdida	ContrastiveLoss
Epochs	5
Warmup Steps	100
Tiempo de Entrenamiento	16.31 minutos

Cuadro 3.6: Parámetros Utilizados para el Entrenamiento del Modelo SBERT Fine-tuned en la tercera iteración

3.6. Generación de Datos Sintéticos

Para evaluar de manera más exhaustiva los modelos SBERT y Word2Vec (en este caso, el modelo de spaCy con la combinación de similitud Jaccard + coseno) de mayor rendimiento, se generaron datos sintéticos que ampliaron el conjunto de datos de prueba. El objetivo de esta generación fue obtener una mayor variedad y volumen de descripciones de productos para mejorar la robustez de las pruebas y la evaluación de los modelos [34, 35].

3.6.1. Técnicas Utilizadas

Reemplazo de Sinónimos

- **spaCy y WordNet**: Se utilizó spaCy junto con WordNet para identificar y reemplazar sinónimos en las descripciones de productos. Sin embargo, los resultados no fueron satisfactorios, ya que muchos reemplazos no tenían sentido en el contexto de las descripciones.

- **BERT**: Se empleó BERT para identificar y reemplazar sinónimos en las descripciones. Al igual que con spaCy, los resultados no fueron óptimos.

Back-Translation

- Se utilizó la técnica de back-translation para traducir las descripciones de productos a otro idioma y luego de vuelta al español, introduciendo variaciones que mantenían el significado original.

- **Googletrans**: Se empleó Google Translate para realizar back-translation. Esta técnica produjo resultados más aceptables en comparación con el reemplazo de sinónimos.

- **Bert2Bert**: Se usaron modelos de traducción de MarianMT para realizar back-translation, lo que también resultó en descripciones variadamente aceptables.

Generación de Texto

- **GPT-2**: Se utilizó el modelo GPT-2 para generar texto a partir de descripciones de productos. Sin embargo, los resultados no fueron útiles ya que las descripciones generadas a menudo carecían de coherencia y relevancia.

- **ChatGPT**: Se utilizó el modelo ChatGPT para generar texto a partir de descripciones de productos. Este demostró resultados ampliamente mejores que los de su predecesor, sin embargo, la selección optima de resultados fue extensa al tener que ser un trabajo de selección manual para asegurar la coherencia y relevancia se las descripciones nuevas.

3.6.2. Resultados y Selección de Técnicas

Los resultados de las técnicas utilizadas se evaluaron y compararon, concluyendo que las técnicas de back-translation, específicamente utilizando Googletrans y Bert2Bert, además del uso de generación de texto con modelos como GPT más actualizados fueron las más efectivas para generar descripciones sintéticas coherentes y útiles. Al final, se generaron 1750 descripciones sintéticas utilizando estas dos técnicas.

3.6.3. Observaciones

La generación de datos sintéticos mediante técnicas de back-translation demostró ser una estrategia efectiva para ampliar y diversificar el conjunto de datos de prueba, a pesar del trabajo manual y tiempo requeridos para seleccionar las descripciones adecuadas. Esto permitió una evaluación más robusta de los modelos SBERT y Word2Vec, destacando las ventajas y limitaciones de cada enfoque. La adición de descripciones sintéticas proporcionó un medio para explorar y validar la capacidad de los modelos en contextos más variados y complejos.

3.7. Conclusiones

A lo largo de este proyecto, se desarrollaron y evaluaron modelos avanzados de aprendizaje automático. Utilizando técnicas de procesamiento de lenguaje natural (NLP) como Word2Vec, Sentence Transformers (SBERT) y modelos preentrenados de spaCy, se implementaron métodos para generar embeddings, calcular similitudes de coseno y Jaccard, y combinarlas para mejorar la precisión de las asignaciones. Tambien realizaron ajustes finos en SBERT utilizando diferentes configuraciones de parámetros y funciones de pérdida,para demostrar como el ajuste continuo y la incorporación de métricas combinadas pueden incrementar significativamente la precisión. Además, se exploraron alternativas como la lematización, el uso de N-gramas y descripciones subyacentes para optimizar el rendimiento del modelo. Estas técnicas mostraron mejoras en la precisión, reduciendo redundancias y mejorando la eficiencia del proceso. En conclusion, la implementación y desarrollo de este estudio subraya la importancia de utilizar múltiples enfoques y técnicas avanzadas de NLP para lograr asignaciones más precisas y eficientes. Se invita al lector a continuar con el próximo capítulo, donde se presentan los resultados detallados de estas implementaciones y evaluaciones.

Capítulo 4

Resultados

En este capítulo se presentan los resultados obtenidos y la interpretación de los mismos posterior al análisis de similitud de descripciones de productos despues de haber utilizado diversos enfoques y modelos de embeddings. Se realizaron varias iteraciones utilizando diferentes técnicas y modelos, evaluando su desempeño en la tarea de emparejar descripciones de las SC con las descripciones oficiales de la BMC (Bolsa Mercantil de Colombia).

4.1. Desarrollo del Modelo con Word2Vec

Después de entrenar un modelo Word2Vec utilizando las descripciones de productos tanto de las SC como del BMC, se evaluó efectividad mediantes distintas iteraciones en las cuales los parámetros de entrenamiento del modelo variaron y se calcularon las similitudes del coseno entre los vectores de las descripciones de las SC y las descripciones del BMC.

Iteración 1

Métrica	**Valor**
Similitud promedio directa	0.6402285655820605
Top k	1
Correctamente alineados	51 de 17288
Porcentaje de alineación correcta	0.30 %
Top k	3
Correctamente alineados	141 de 17288
Porcentaje de alineación correcta	0.82 %

Cuadro 4.1: Resultados del modelo Word2Vec (Tokens)

Iteración 2

Métrica	**Valor**
Similitud promedio directa	0.6012983722794533
Top k	1
Correctamente alineados	39 de 17288
Porcentaje de alineación correcta	0.23 %
Top k	3
Correctamente alineados	105 de 17288
Porcentaje de alineación correcta	0.61 %

Cuadro 4.2: Resultados del modelo Word2Vec (Tokens)

Iteración 3

Métrica	**Valor**
Similitud promedio directa	0.5962543640411025
Top k	1
Correctamente alineados	53 de 17288
Porcentaje de alineación correcta	0.30 %
Top k	3
Correctamente alineados	138 de de 17288
Porcentaje de alineación correcta	0.80 %

Cuadro 4.3: Resultados del modelo Word2Vec (Tokens)

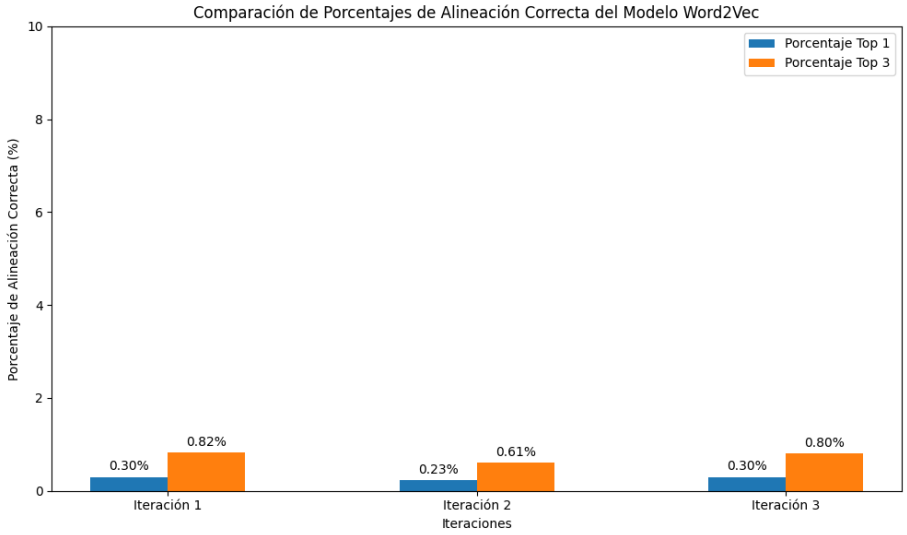

Figura 4.1: Comparación de Porcentajes de Alineación Correcta del Modelo Word2Vec.

Interpretación: El valor de similitud promedio sugiere que hay un cierto grado de similitud entre las descripciones, pero no es particularmente alto. Esto podría deberse a la variabilidad en la forma de describir los productos entre las descripciones de las SC y las descripciones oficiales del BMC. Además de esto se comprobó que la variaciones

en los parámetros no afectaron demasiado los resultados de entrenamiento como vemos en la figura 4.1, por lo cual, para el resto de las pruebas, se usó el modelo entrenado con los parámetros de la iteración 1, los cuales eran los mas balanceados y brindaba los mejores resultados.

4.1.1. Utilización de N-gramas

En un intento por mejorar la precisión utilizando N-gramas para capturar mejor las frases comunes en las descripciones de productos.

Métrica	**Valor**
Similitud promedio directa	0.5892576087382935
Top k	1
Correctamente alineados	10 de 17288
Porcentaje de alineación correcta	0.0578 %
Top k	3
Correctamente alineados	19 de 17288
Porcentaje de alineación correcta	0.1091 %

Cuadro 4.4: Resultados del modelo Word2Vec (N-gramas)

Interpretación: El uso de N-gramas no mejoró la precisión de la alineación de descripciones, lo que sugiere que el enfoque de N-gramas no es efectivo en este caso para capturar la semántica de las descripciones de productos.

4.1.2. Entrenamiento con Descripciones Combinadas

Al probar con la combinación de las descripciones de las SC y de la BMC para observar cómo esto afectaba los resultados de alineación y el desarrollo de los embeddings se obtubo:.

Métrica	**Valor**
Similitud promedio directa	0.008290658933038349
Top k	1
Correctamente alineados	5 de 17288
Porcentaje de alineación correcta	0.0289 %

Cuadro 4.5: Resultados del modelo Word2Vec (Descripciones Combinadas)

Interpretación: La combinación de descripciones no resultó en una mejora en la precisión, ya que la similitud promedio y el porcentaje de alineación correcta fueron los más bajos obtenidos hasta el momento. Esto demuestra que este enfoque no es para nada el indicada para intentar mejorar la comprensión semántica de las descripciones.

4.1.3. Entrenamiento con Lemmas

Se utilizaron lemas en lugar de tokens crudos para ver si esto mejoraba la precisión del modelo.

Métrica	**Valor**
Similitud promedio directa	0.6073812505204846
Top k	1
Correctamente alineados	46 de 17288
Porcentaje de alineación correcta	0.2661 %
Top k	3
Correctamente alineados	130 de 17288
Porcentaje de alineación correcta	0.752 %

Cuadro 4.6: Resultados del modelo Word2Vec (Lemmas)

Interpretación: El uso de lemas mejoró ligeramente la precisión en comparación con el uso de tokens crudos, pero los resultados aún fueron bajos en términos de alineación correcta y estos decrecieron ligeramente a la hora de evaluar las tres descripciones mas similares, por lo que este enfoque parece no resultar prometedor.

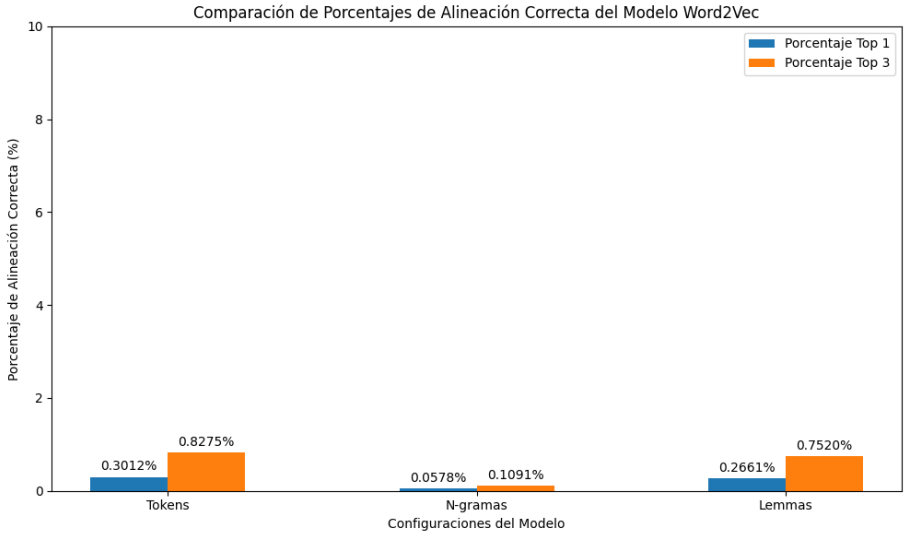

Figura 4.2: Comparación de Porcentajes de Alineación Correcta del Modelo Word2Vec.

4.1.4. Observaciones

De los distintos métodos utilizados, el que obtubo mejores resultados fue el modelo de *Word2Vec* que utilizó tokens para su entrenamiento, sin embargo estos resultados demuestran una tasa de alineación tan baja que ninguno parece ser realmente útil en el desarrollo de la tarea que se esta abordando.

4.2. Resultados del Modelo Preentrenado de spaCy

Al usar un modelo preentrenado de spaCy para generar embeddings y evaluar la similitud de las descripciones, se obtuvo:

Métrica	**Valor**
Similitud promedio directa	0.470233
Top k	1
Correctamente alineados	450 de 17288
Porcentaje de alineación correcta	2.60 %
Top k	3
Correctamente alineados	803 de 17288
Porcentaje de alineación correcta	4.64 %

Cuadro 4.7: Resultados del modelo spaCy

Interpretación: El uso del modelo preentrenado de spaCy resultó en una mejora significativa en comparación con los modelos entrenados localmente con Word2Vec. Aunque el porcentaje de alineación correcta sigue siendo bajo, se observa una clara mejora en la capacidad del modelo para identificar similitudes entre las descripciones, como s emuestra en al figura 4.3.

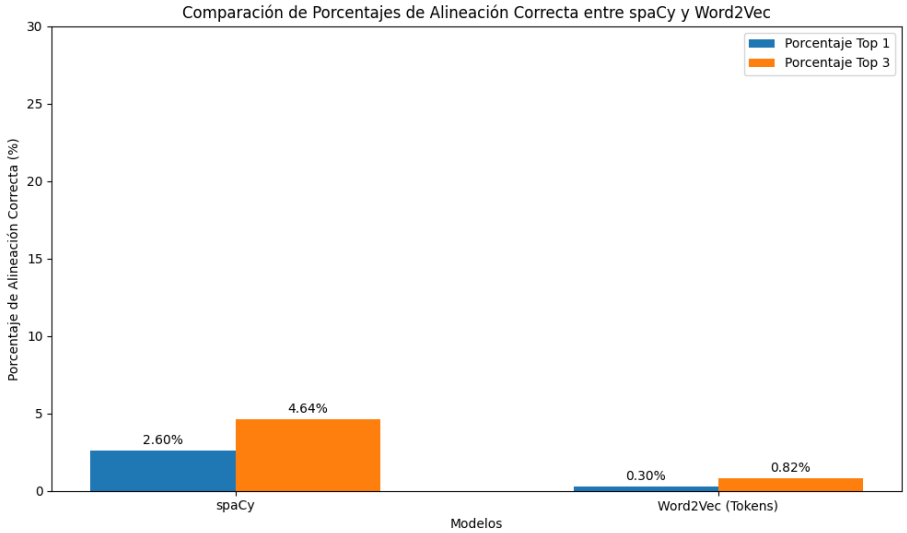

Figura 4.3: Comparación de Porcentajes de Alineación Correcta del Modelo Word2Vec y spacy.

4.2.1. Observaciones

Los resultados visibles en la figura 4.3 muestran que el modelo preentrenado de spaCy supera significativamente a los modelos Word2Vec entrenados localmente en términos de alineación correcta de descripciones de productos. La eficacia de spaCy sugiere que los modelos preentrenados, con su capacidad para captar mejor las sutilezas lingüísticas, son más adecuados para esta tarea, subrayando la importancia de utilizar técnicas más sofisticadas y entrenadas en grandes corpus para mejorar la precisión en la alineación de descripciones.

4.3. Resultados con Descripciones Subyacentes

Al abordar el problema de una manera más simplificada, se buscó reducir las probabilidades de error en la asignación, además tener mas registros de los cuales pueda valerse el modelo para identificar patrones por código o descripción de la BMC.

4.3.1. Resultados Word2Vec con Tokens

Se entrenó un modelo Word2Vec utilizando las descripciones de productos tanto de las SC como de la BMC. Después de calcular las similitudes coseno entre los vectores de las descripciones de las SC y las descripciones subyacentes del BMC y evaluar su precisión se obtubo que:

Métrica	**Valor**
Similitud promedio directa	0.7147727002047899
Top k	1
Correctamente alineados	71 de 17288
Porcentaje de alineación correcta	0.4107 %
Top k	3
Correctamente alineados	187 de 17288
Porcentaje de alineación correcta	1.0817 %

Cuadro 4.8: Resultados del modelo Word2Vec (Tokens) con Descripciones Subyacentes

Interpretación: El uso de descripciones subyacentes resultó en una similitud promedio más alta en comparación con las descripciones completas. Sin embargo, la precisión de la alineación correcta sigue siendo baja. La precisión mejora ligeramente al considerar las tres descripciones con mayor similitud, pero sigue siendo insuficiente para aplicaciones prácticas.

4.3.2. Resultados con Modelo Preentrenado de spaCy

Se utilizó un modelo preentrenado de spaCy para generar embeddings y evaluar la similitud de las descripciones subyacentes.

Métrica	**Valor**
Top k	1
Correctamente alineados	620 de 17288
Porcentaje de alineación correcta	3.59 %
Top k	3
Correctamente alineados	1451 de 17288
Porcentaje de alineación correcta	8.39 %

Cuadro 4.9: Resultados del modelo spaCy con Descripciones Subyacentes

Interpretación: Vemos una mejora en la alineación en comparación con los embeddings procesados directamente con las descripciones más complejas. Considerar las tres descripciones con mayor similitud mejora significativamente la precisión de alineación.

4.3.3. Similitud Jaccard

Se probó la similitud Jaccard, midiendo la similitud basada en la cantidad de palabras compartidas entre descripciones.

Descripciones Completas

Métrica	Valor
Top k	1
Correctamente alineados	2209 de 17288
Porcentaje de alineación correcta	12.78 %
Top k	3
Correctamente alineados	3651 de 17288
Porcentaje de alineación correcta	21.12 %

Cuadro 4.10: Resultados de Similitud Jaccard con Descripciones Completas

Interpretación: Aunque la precisión es menor en comparación con las descripciones subyacentes, la similitud Jaccard sigue siendo una métrica útil para esta tarea.

Descripciones Subyacentes

Métrica	Valor
Top k	1
Correctamente alineados	4708 de 17288
Porcentaje de alineación correcta	27.23 %
Top k	3
Correctamente alineados	6198 de 17288
Porcentaje de alineación correcta	35.85 %

Cuadro 4.11: Resultados de Similitud Jaccard con Descripciones Subyacentes

Interpretación: La similitud Jaccard resultó en una precisión mucho más alta en comparación con la similitud del coseno, lo que sugiere que esta métrica es adecuada como añadidura a la similitud coseno para la tarea de alineación de descripciones.

4.3.4. spaCy con Similitud Coseno + Jaccard

Finalmente, se probó una combinación de las similitudes del coseno y Jaccard para mejorar aún más la precisión para el modelo preentrenado de *spaCy* con descripciones subyacentes.

Métrica	Valor
Top k	1
Correctamente alineados	2024 de 17288
Porcentaje de alineación correcta	11.71 %
Top k	3
Correctamente alineados	4739 de 17288
Porcentaje de alineación correcta	27.41 %

Cuadro 4.12: Resultados Combinados de Similitud Coseno y Jaccard

Interpretación: La combinación de ambas métricas resulta en una mejora significativa en la precisión de alineación, destacando la importancia de utilizar múltiples métricas para capturar mejor la similitud entre descripciones.

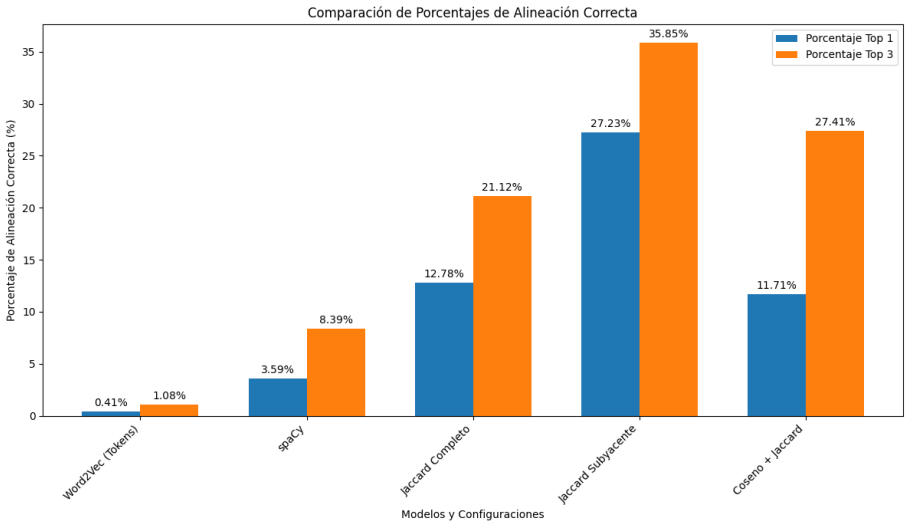

Figura 4.4: Comparación de Porcentajes de Alineación Correcta para descripciones subyacentes Word2Vec, spacy, Jaccard

4.3.5. Observaciones

Las deducciones que se pueden hacer al observar la figura 4.4 de los resultados indican que la simplificación de las descripciones subyacentes mejora la precisión de los modelos, con el modelo preentrenado de spaCy superando a Word2Vec. La similitud Jaccard resultó especialmente eficaz con descripciones subyacentes, alcanzando los mayores porcentajes de alineación correcta. La combinación de similitud coseno y Jaccard mejoró aún más la precisión, destacando la importancia de usar múltiples métricas para capturar mejor la similitud entre descripciones. Estos hallazgos sugieren que simplificar las descripciones y combinar métricas de similitud son estrategias clave para mejorar la alineación de las descripciones.

4.4. Resultados del Modelo con Redes Siamesas

Se utilizó un modelo preentrenado de SBERT para evaluar la similitud entre las descripciones de las SC y las descripciones del BMC. Se realizaron varias pruebas para medir la precisión de los emparejamientos.

4.4.1. Resultados con Descripciones Completas

Resultados Modelo Español

Métrica	Valor
Top k	1
Correctamente alineados	1697 de 17288
Porcentaje de alineación correcta	9.82 %
Top k	3
Correctamente alineados	3155 de 17288
Porcentaje de alineación correcta	18.25 %

Cuadro 4.13: Resultados del modelo SBERT con Descripciones Completas

Interpretación: El uso del modelo SBERT con descripciones completas resultó en una precisión del 9.82 % para las coincidencias exactas. Esta precisión es baja, lo que indica la necesidad de mejorar el enfoque. Considerar las tres descripciones con mayor similitud mejora la precisión al 18.25 %, lo que sigue siendo insuficiente para aplicaciones prácticas.

Resultados Modelo Multilingüe

Se utilizó un modelo preentrenado multilingüe para evaluar su precisión en la tarea de emparejamiento.

Métrica	Valor
Top k	1
Correctamente alineados	1372 de 17288
Porcentaje de alineación correcta	7.94 %
Top k	3
Correctamente alineados	2942 de 17288
Porcentaje de alineación correcta	17.02 %

Cuadro 4.14: Resultados del modelo Multilingüe

Interpretación: El modelo multilingüe resultó en una precisión del 7.94 % para las coincidencias exactas, lo que es inferior al modelo específico para español utilizado anteriormente. Por lo tanto, no se procedió con experimentos adicionales con este modelo.

4.4.2. Prueba con Descripciones Subyacentes

Se utilizaron descripciones subyacentes simplificadas en lugar de las descripciones completas del BMC para observar si esto mejoraba los resultados.

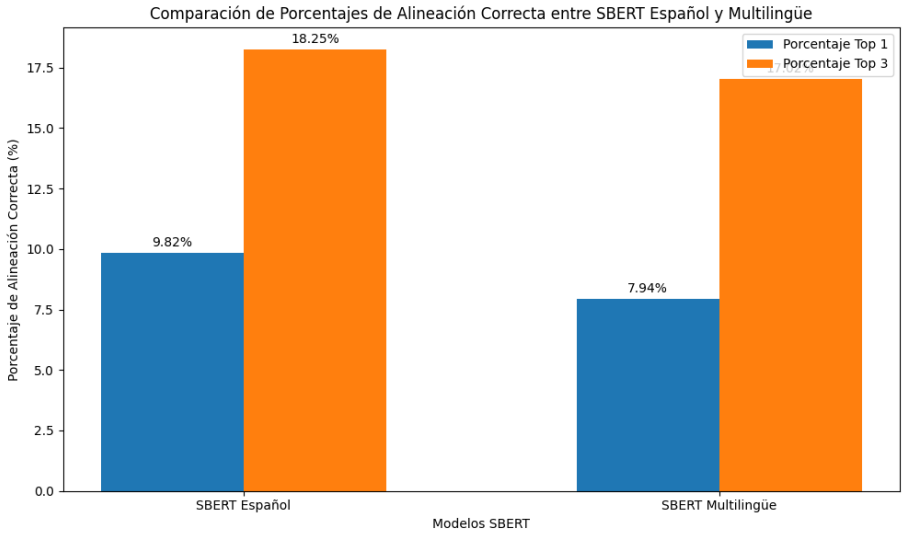

Figura 4.5: Comparación de Porcentajes de Alineación Correcta SBERT español vs SBERT multilingua

Métrica	**Valor**
Top k	1
Correctamente alineados	2669 de 17288
Porcentaje de alineación correcta	15.44 %
Top k	3
Correctamente alineados	4555 de 17288
Porcentaje de alineación correcta	26.35 %

Cuadro 4.15: Resultados del modelo SBERT con Descripciones Subyacentes

Interpretación: El uso de descripciones subyacentes resultó en una precisión del 15.44 % para las coincidencias exactas, lo que representa una mejora en comparación con el uso de descripciones completas. Considerar las tres descripciones con mayor similitud mejora la precisión al 26.35 %, lo que es significativamente mejor en comparación con el uso de descripciones completas. Esto se puede observar más claramente en la figura 4.5

4.4.3. Combinación de Similitud Coseno y Jaccard

Se combinó la similitud del coseno y la similitud de Jaccard para mejorar aún más la precisión de los emparejamientos.

Métrica	**Valor**
Top k	1
Correctamente alineados	4066 de 17288
Porcentaje de alineación correcta	23.52 %
Top k	3
Correctamente alineados	6384 de 17288
Porcentaje de alineación correcta	36.93 %

Cuadro 4.16: Resultados Combinados de Cosine Similarity y Jaccard Score

Interpretación: Como se puede observar en la figura 4.6, la combinación de ambas métricas resultó en una precisión del 23.52 % para las coincidencias exactas, lo que muestra una mejora significativa en comparación con el uso de una sola métrica, además La combinación de ambas métricas resultó en una precisión del 36.93 % para las coincidencias en el top 3, destacando la importancia de utilizar múltiples métricas para capturar mejor la similitud entre descripciones.

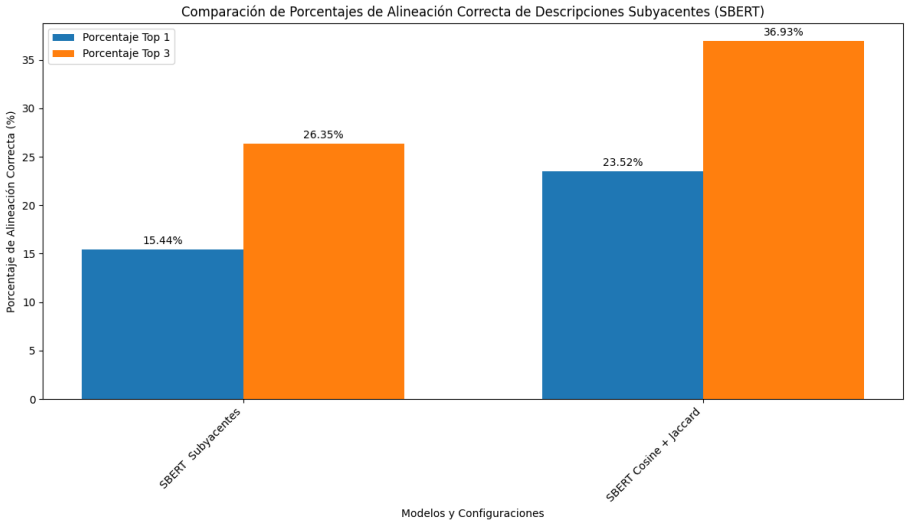

Figura 4.6: Comparación de Porcentajes de Alineación Correcta del Modelo SBERT con descripciones subyacentes

4.4.4. Observaciones

Los resultados indican que el modelos SBERT especializado en español supera al multilingües, sugiriendo que los modelos específicos del idioma captan mejor las particularidades lingüísticas. Como adición podemos observar como la simplificación de descripciones subyacentes mejora significativamente la precisión, destacando la importancia de reducir la complejidad de los datos. La combinación de métricas de similitud coseno y Jaccard demuestró ser más efectiva, subrayando la necesidad de enfoques multifacéticos para capturar mejor las similitudes. A pesar de las mejoras, los resultados sugieren que hay necesidad de un ajuste fino continuo y de exploración de técnicas avanzadas para alcanzar niveles de precisión adecuados para aplicaciones prácticas.

4.5. Desarrollo del Modelo con Redes Siamesas Fine-Tuned

Se utilizó un modelo preentrenado de SBERT para evaluar la similitud entre las descripciones de las SC y las descripciones del BMC. Se realizaron varias pruebas para medir la precisión de los emparejamientos.

4.5.1. Primera Iteración: Epochs = 1

En la primera iteración, el modelo SBERT fue fine-tuned por una sola época.

Descripciones Completas

Métrica	**Valor**
Top k	1
Correctamente alineados	1690 de 17288
Porcentaje de alineación correcta	9.77 %
Top k	3
Correctamente alineados	3071 de 17288
Porcentaje de alineación correcta	17.76 %

Cuadro 4.17: Resultados del modelo SBERT fine-tuned en la primera iteración

Interpretación: La precisión de la alineación correcta es baja (9.77 %). Considerar las tres descripciones con mayor similitud mejora la precisión al 17.76 %, pero estos resultados aún son insuficientes para aplicaciones prácticas.

Descripciones Subyacentes

Métrica	**Valor**
Top k	1
Correctamente alineados	648 de 17288
Porcentaje de alineación correcta	3.75 %
Top k	3
Correctamente alineados	1100 de 17288
Porcentaje de alineación correcta	6.36 %

Cuadro 4.18: Resultados del modelo SBERT fine-tuned con sudyacentes en la primera iteración

Interpretación: El ajuste fino del modelo SBERT utilizando descripciones subyacentes y sin ejemplos negativos resultó en una precisión baja del 3.75 %. Al considerar las tres descripciones con mayor similitud, la precisión aumentó al 6.36 %, lo cual sigue siendo insuficiente para aplicaciones prácticas.

4.5.2. Segunda Iteración: Epochs = 3 y Ejemplos Negativos

En la segunda iteración, el modelo SBERT fue fine-tuned por tres épocas, añadiendo ejemplos negativos al conjunto de entrenamiento.

Descripciones Completas

Métrica	**Valor**
Top k	1
Correctamente alineados	6017 de 17288
Porcentaje de alineación correcta	34.84 %
Top k	3
Correctamente alineados	10288 de 17288
Porcentaje de alineación correcta	59.52 %

Cuadro 4.19: Resultados del modelo SBERT fine-tuned en la segunda iteracion

Interpretación: El uso de ejemplos negativos y el aumento del número de épocas mejoró significativamente la precisión del modelo. La precisión de alineación correcta aumentó al 34.84 % y la precisión en el top 3 aumentó al 59.52 %.

Descripciones Subyacentes

Métrica	Valor
Top k	1
Correctamente alineados	8151 de 17288
Porcentaje de alineación correcta	47.14 %
Top k	3
Correctamente alineados	12055 de 17288
Porcentaje de alineación correcta	69.71 %

Cuadro 4.20: Resultados del modelo SBERT fine-tuned con sudyacentes en la segunda iteración

Interpretación: El ajuste fino del modelo SBERT con tres épocas y la inclusión de ejemplos negativos mejoró significativamente la precisión del modelo, alcanzando un 47.14 % de alineación correcta. Considerar las tres descripciones con mayor similitud mejoró la precisión al 69.71 %, mostrando una gran mejora en comparación con las pruebas sin ejemplos negativos.

4.5.3. Tercera Iteración: Uso de Contrastive Loss

Para seguir mejorando los resultados, se realizó una tercera iteración de ajuste fino utilizando *ContrastiveLoss*, que es adecuado para tareas de comparación de similitud.

Descripciones Completas

Métrica	Valor
Top k	1
Correctamente alineados	5862 de 17288
Porcentaje de alineación correcta	33.91 %
Top k	3
Correctamente alineados	10170 de 17288
Porcentaje de alineación correcta	58.81 %

Cuadro 4.21: Resultados del modelo SBERT fine-tuned en la tercera iteración

Interpretación: La utilización de *Contrastive Loss* mantuvo resultados similares en comparación con el uso de ejemplos negativos y múltiples épocas. La precisión de alineación correcta fue de 33.91 %, y la precisión en el top 3 fue de 58.81 %. Aunque no hay una mejora significativa, este enfoque sigue siendo prometedor.

Descripciones Subyacentes

Métrica	**Valor**
Top k	1
Correctamente alineados	8239 de 17288
Porcentaje de alineación correcta	47.66 %
Top k	3
Correctamente alineados	11865 de 17288
Porcentaje de alineación correcta	68.60 %

Cuadro 4.22: Resultados del modelo SBERT fine-tuned con sudyacentes en la tercera iteración

Interpretación: La inclusión de cinco épocas, ejemplos negativos y el uso de *Contrastive Loss* resultó en una precisión del 47.66 % para las alineaciones correctas. Considerar las tres descripciones con mayor similitud resultó en una precisión del 68.60 %. Aunque hay una ligera mejora en comparación con las pruebas sin *Contrastive Loss*, los resultados son consistentes con las mejoras significativas observadas al incluir ejemplos negativos.

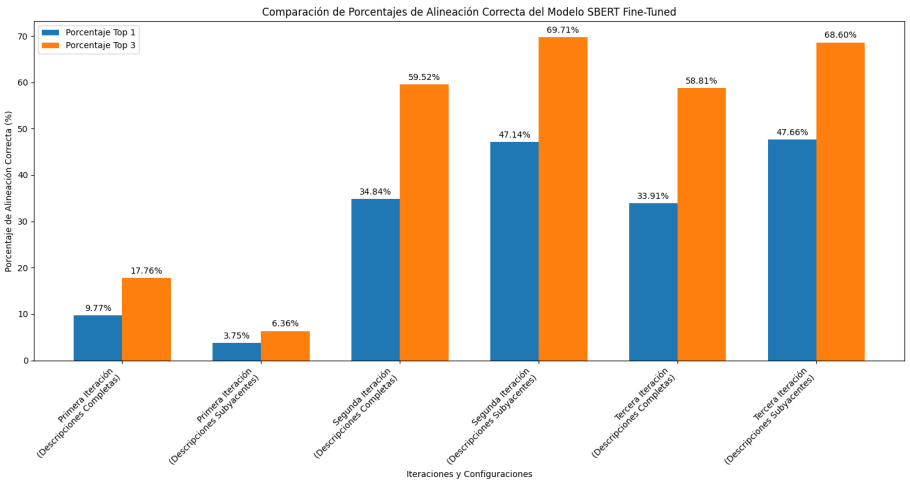

Figura 4.7: Comparación de Porcentajes de Alineación Correcta del Modelo SBERT Fine-Tuned

4.5.4. Observaciones

Los resultados, ilustrados por la figura 4.7, indican que el refinamiento del modelo SBERT con los parámetros de la segunda iteración fue el más óptimo a la hora de probar la alineación correcta de productos. Además, se mostraron mejores resultados con el uso de descripciones subyacentes, revalidando la idea de que simplificar y mejorar la categorización de estos productos puede mejorar ampliamente los resultados en su correcta asignación. Estos hallazgos subrayan la importancia de utilizar descripciones más claras y específicas, así como de ajustar los modelos de manera precisa, para lograr una mayor precisión en la asignación automática de códigos de productos.

4.6. Resultados de pruebas con datos sintéticos

Para evaluar de manera más exhaustiva los modelos con los mejores rendimientos, SBERT fine-tuned y spaCy con Jaccard + Cosine Similarity, se realizaron pruebas utilizando el conjunto de datos sintéticos creado previamente. Se generaron un total de 1750 descripciones sintéticas mediante técnicas de back-translation y modelos generativos de texto. Los resultados de estas pruebas proporcionaron una visión más amplia de la capacidad de estos modelos para manejar descripciones de productos variadas y complejas.

4.6.1. Resultados del modelo spaCy con Jaccard + Cosine Similarity

Métrica	**Valor**
Top k	1
Correctamente alineados	288 de 1750
Porcentaje de alineación correcta	16.46 %
Top k	3
Correctamente alineados	446 de 1750
Porcentaje de alineación correcta	25.49 %

Cuadro 4.23: Resultados del modelo spaCy Jaccard + Cosine Similarity con datos sintéticos

Interpretación: El modelo spaCy con la combinación de Jaccard y Cosine Similarity logró un 25.49 % de precisión al considerar las tres descripciones más similares, y un 16.46 % de precisión para coincidencias exactas. Esto demuestra una mejora comparado con el corpus original.

4.6.2. Resultados del modelo SBERT Fine-tuned

Métrica	**Valor**
Top k	1
Correctamente alineados	781 de 1750
Porcentaje de alineación correcta	44.63 %
Top k	3
Correctamente alineados	1235 de 1750
Porcentaje de alineación correcta	70.57 %

Cuadro 4.24: Resultados del modelo SBERT fine-tuned con datos sintéticos

Interpretación: El modelo SBERT fine-tuned logró una precisión del 44.63 % para coincidencias exactas y del 70.57 % al considerar las tres descripciones más similares. Estos resultados resaltan la eficacia del modelo SBERT en la identificación de coincidencias relevantes en un contexto variado y complejo.

4.6.3. Observaciones

Los resultados obtenidos mediante el uso de datos sintéticos confirman que el modelo SBERT fine-tuned tiene un rendimiento superior en comparación con el modelo spaCy con Jaccard + Cosine Similarity, como se observa en la figura 4.8. La precisión mejorada,

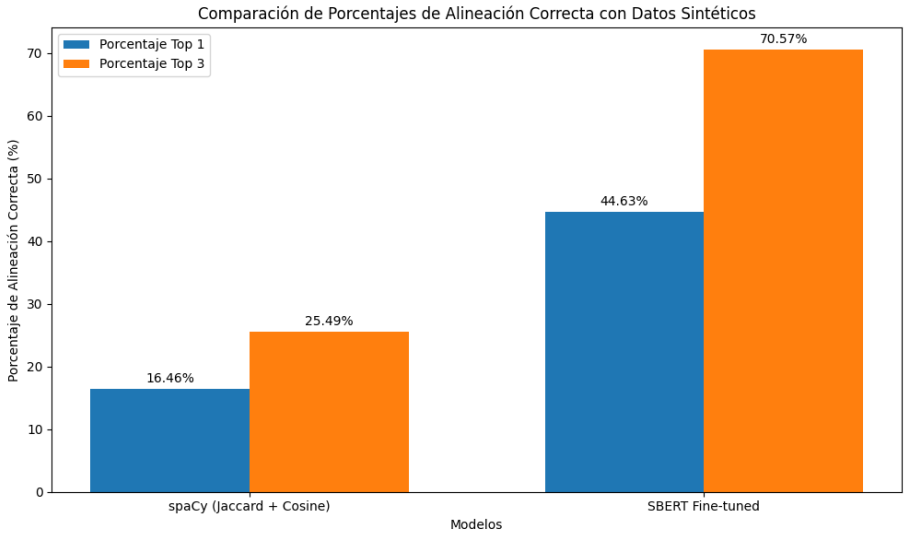

Figura 4.8: Comparación de Porcentajes de Alineación Correcta del Modelo SBERT Fine-Tuned vs spaCy Jaccard + Coseno

especialmente al considerar las tres descripciones más similares, demuestra la robustez y la capacidad del modelo SBERT para manejar descripciones variadas y complejas, lo cual es crucial para aplicaciones prácticas en la asignación automática de códigos de productos.

4.7. Conclusiones

En este capítulo, se han comparado diversos modelos y técnicas para la alineación automática de descripciones de productos, utilizando tanto datos reales como datos sintéticos. A lo largo de las pruebas, los modelos basados en SBERT fine-tuned y spaCy con Jaccard + Cosine Similarity demostraron ser los más efectivos, con SBERT fine-tuned mostrando consistentemente los mejores resultados.

Las observaciones clave son las siguientes:

- El modelo Word2Vec, aunque mostró algunas capacidades de alineación, no logró un rendimiento adecuado para aplicaciones prácticas.
- El modelo preentrenado de spaCy superó significativamente a los modelos Word2Vec, indicando la ventaja de utilizar modelos preentrenados en grandes corpus.
- La combinación de métricas de similitud, como Cosine y Jaccard, resultó en una mejora significativa en la precisión de alineación, subrayando la importancia de utilizar enfoques multifacéticos.
- El uso de datos sintéticos ayudó a evaluar de manera más exhaustiva los modelos, confirmando la superioridad del modelo SBERT fine-tuned en términos de precisión y robustez.

Estas conclusiones llevan a considerar que para mejorar aún más los resultados de la alineación de descripciones de productos, es crucial seguir explorando técnicas avanzadas y ajustes finos de los modelos SBERT. Se invita al lector a continuar con el siguiente capítulo, donde se presentarán las conclusiones finales de este estudio y las direcciones futuras de investigación y desarrollo.

Capítulo 5

Conclusiones

5.1. Conclusiones

A lo largo de la ejecución de este proyecto, se han observado ciertas limitaciones con las cuales se ha trabajado para obtener los mejores resultados posibles. A continuación, se detallan algunas de estas limitaciones y se presentan las principales conclusiones derivadas del estudio.

5.1.1. Limitaciones del Corpus

- **Variabilidad y Limitaciones de las Descripciones**: El corpus utilizado presentó variaciones significativas en la calidad y consistencia de las descripciones. Las descripciones proporcionadas por las Sociedades Comisionistas (SC) variaban ampliamente en términos de detalle y precisión, lo que dificultó la tarea de alineación. Además, el número limitado de ejemplos de descripciones de las SC por cada descripción de la BMC restringió la capacidad del modelo para generalizar adecuadamente.

- **Errores Ortográficos y Léxico Variado**: La presencia de errores ortográficos y un léxico variado en las descripciones complicaron aún más el proceso de alineación. Estos errores introdujeron ruido en los datos, afectando negativamente la precisión del modelo. La estandarización de textos y la corrección ortográfica podrían ser pasos adicionales para mejorar la calidad del corpus, sin embargo al ser un trabajo manualmente pesado se escapaba de los alcances de esta investigación.

- **Falta de Metadatos Adicionales**: La exclusividad del uso de descripciones textuales limitó el alcance del modelo. Incorporar información adicional, como categorías de productos, marcas o cualquier otro metadato relevante, podría mejorar significativamente la precisión de las coincidencias al proporcionar un contexto más rico para el modelo.

5.1.2. Resultados Principales

- **Word Embeddings y Word2Vec**: A pesar de la utilidad de los word embeddings basados en la técnica Word2Vec en muchos contextos, estos no lograron ser utilizados de manera efectiva en el ámbito de la similitud de textos para la cantidad y calidad de datos manejados en este proyecto. Sin embargo, se logró comprobar que el uso de modelos preentrenados con corpus mucho más vastos puede resultar en una herramienta poderosa y útil para identificar la similitud semántica de

oraciones. Esta efectividad se hizo más evidente al comparar con los resultados obtenidos de modelos entrenados exclusivamente con nuestro corpus.

- **Modelos Preentrenados de spaCy**: El uso de modelos preentrenados, como el de spaCy, proporcionó mejoras significativas en la precisión de alineación. Al comparar las descripciones completas y subyacentes, se observó una mejora notable, subrayando la ventaja de utilizar modelos preentrenados que capturan mejor las sutilezas lingüísticas.
- **Combinación de Métricas de Similitud**: La combinación de métricas de similitud, como Jaccard y coseno, proporcionó un aumento apreciable en la precisión del alineamiento de descripciones. Esta combinación aprovechó la intersección en el conjunto de palabras entre oraciones, lo cual es valioso en el contexto de este proyecto, dado que no se disponía de un corpus vasto y preciso para reentrenar el modelo.
- **Redes Siamesas (SBERT)**: El uso de redes siamesas (SBERT) mostró un verdadero potencial para identificar la similitud de oraciones en un ámbito técnico como la descripción de productos. Las redes siamesas proporcionaron mejores resultados en casi todos los ámbitos en comparación con los modelos entrenados con técnicas como Word2Vec. Esta mejora fue aún más evidente al realizar un refinamiento (fine-tuning) del modelo, especialmente al aumentar las épocas de entrenamiento y añadir ejemplos negativos.
- **Descripciones Subyacentes**: Una aproximación simplificada a la tarea, donde se categorizaron los productos de manera más general, mejoró ampliamente la precisión de todos los modelos utilizados. Esta estrategia demostró ser valiosa y debería contemplarse para futuras investigaciones y aplicaciones.

Siendo concisos, los resultados de este proyecto destacan la importancia de utilizar modelos preentrenados, la combinación de métricas de similitud y las redes siamesas para mejorar la precisión en la asignación de códigos de productos. Un trabajo continuo y más detallado, enfocado en el refinamiento y la incorporación de categorías de clasificación más amplias en los datos de entrenamiento, podría llevar a mejoras aún mayores.

5.2. Trabajo a Futuro

Para continuar mejorando y expandiendo los resultados obtenidos en este proyecto, se identifican varias áreas clave para futuros trabajos:

- **Ajuste Fino Continuo**: Los modelos desarrollados pueden beneficiarse de un ajuste fino continuo, explorando diferentes configuraciones de hiperparámetros y técnicas de optimización para mejorar aún más la precisión. Experimentar con diversas funciones de pérdida y estrategias de entrenamiento podría conducir a mejoras significativas. Un acercamiento a medidas de similitud como similitud coseno y/o Jaccard score pueden resultar beneficiosas a la hora de querer refinar los ejemplos de entrenamiento.
- **Integración de Datos Adicionales**: La ampliación general de ejemplos y la incorporación de datos adicionales, como descripciones más detalladas o información contextual sobre los productos, podría mejorar la calidad de los embeddings y, por lo tanto, la precisión de las asignaciones. Metadatos adicionales, como categorías de productos, marcas y atributos específicos, proporcionarían un contexto más completo para el modelo.

- **Optimización del Tiempo de Cálculo**: Aunque la precisión es un factor crucial, la experimentación de distintos modelos y frameworks optimizados y el uso de distinto hardware (como uno enfocado en GPU) nos demostró que también es importante optimizar el tiempo de cálculo y la eficiencia de los modelos para aplicaciones prácticas en entornos operativos. Técnicas de compresión de modelos y optimización de hardware pueden ser exploradas para garantizar que los modelos sean no solo precisos, sino también eficientes en términos de recursos computacionales.

- **Aplicaciones en Otros Sectores**: Los métodos y modelos desarrollados en este proyecto pueden ser adaptados y aplicados a otros sectores que requieran la clasificación y asignación automática de códigos o categorías basadas en descripciones textuales. Esto podría incluir sectores como la salud, el comercio electrónico y la gestión de inventarios, ya que estos se especializan en el uso de un lenguaje técnico de uso especifico, lo cual podría ampliar el impacto y la aplicabilidad de esta investigación.

- **Desarrollo de un Prototipo Funcional**: Crear un prototipo funcional que implemente el modelo con el mejor rendimiento, integrándolo en un sistema de producción que permita a las Sociedades Comisionistas de la BMC utilizarlo de manera efectiva y eficiente en sus operaciones diarias. Este prototipo debería incluir una interfaz de usuario amigable y herramientas para la supervisión y ajuste del modelo en tiempo real.

En conclusión, con este proyecto se ha demostrado el potencial significativo de las técnicas de NLP y aprendizaje automático para mejorar la asignación de códigos de productos, dando así una base sólida para futuras investigaciones y aplicaciones en el campo. La incorporación de mejoras adicionales y la expansión a otros dominios prometen un impacto aún mayor en la eficiencia operativa y la precisión en la asignación de códigos en diversas industrias.

Bibliografía

[1] Analytics India Magazine, "NLP case study: Identify Documents Similarity", Analytics India Magazine, [Online]. Disponible: https://analyticsindiamag.com/nlp-case-study-identify-documents-similarity/.[Accedido: 30 de octubre de 2023].

[2] C. Wang, M. Qiu, C. Shi, T. Zhang, T. Liu, L. Li, J. Wang, M. Wang, J. Huang, y W. Lin, "EasyNLP: A Comprehensive and Easy-to-use Toolkit for Natural Language Processing", arXiv:2205.00258v2 [cs.CL], 13 Mar 2023

[3] A. A. Chakrabarty, "Text Data Labelling Using Transformer Based Sentence Embeddings and Text Similarity for Text Classification", en International Journal on Natural Language Computing (IJNLC), vol. 11, no. 2, abril de 2022

[4] Z. Liu, C. Benge, S. Jiang "Ticket-BERT: Labeling Incident Management Tickets with Language Models", arXiv:2307.00108, 2023.

[5] S. Gupta, U. Vadgama and T. R. Vedhavathy, Ïdentification and Labeling of Textual Cyberbullying using BiLSTM and BERT,"2023 International Conference on Networking and Communications (ICNWC), Chennai, India, 2023, pp. 1-5, doi: 10.1109/ICNWC57852.2023.10127308.

[6] Bolsa Mercantil de Colombia, "Historia", Bolsa Mercantil de Colombia, [En línea]. Disponible: https://www.bolsamercantil.com.co/. [Accedido: 30 de octubre de 2023].

[7] P. Johri, S. K. Khatri, A. T. Al-Taani, M. Sabharwal, S. Suvanov, and A. Kumar, "Natural Language Processing: History, Evolution, Application, and Future Work", in *Proceedings of 3rd International Conference on Computing Informatics and Networks*, pp. 365-375, 2021. DOI: 10.1007/978-981-15-9712-1_31.

[8] Turing, "Word embeddings in NLP: A Complete Guide" , Turing.com. [En línea]. Disponible: https://www.turing.com/kb/guide-on-word-embeddings-in-nlp. [Accedido: 26-Nov-2023].

[9] A. Huang, "Similarity measures for text document clustering," *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC)*, Christchurch, New Zealand, 2008.

[10] scikit-learn, *sklearn.metrics.top_k_accuracy_score,* Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.top_k_accuracy_score.html, Accessed on: May 20, 2024.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin,"Attention Is All You Need" en *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA, 2017. [Online]. Disponible: arXiv:1706.03762v7 [cs.CL]

[12] Eligijus Bujokas, "Creating Word Embeddings: Coding the Word2Vec Algorithm in Python using Deep Learning", Towards Data Science. Disponible:https://towardsdatascience.com/creating-word-embeddings-coding-the-word2vec-algorithm-in-python-using-deep-learning-b337d0ba17a8/[Accedido: 26-Nov-2023].

[13] Scaler Topics, "Siamese Networks in NLP", Scaler.com. [En línea]. Disponible:https://www.scaler.com/topics/siamese-networks-in-nlp/. [Accedido: 26-Nov-2023].

[14] V. Novotný, P. Sojka, E. F. Ayetiran, y M. Štefánik, "Text classification with word embedding regularization and soft similarity measure" *arXiv preprint arXiv:2003.05019*, 2020.

[15] Analytics India Magazine,"Word2Vec vs GloVe - A Comparative Guide to Word Embedding Techniques" Analyticsindiamag.com, 2023. [En línea]. Disponible: https://analyticsindiamag.com/word2vec-vs-glove-a-comparative-guide-to-word-embedding-techniques/. [Accedido: 26-Nov-2023].

[16] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks", Ubiquitous Knowledge Processing Lab (UKP-TUDA), Department of Computer Science, Technische Universität Darmstadt, 2019. [Online]. Disponible:https://github.com/UKPLab/sentence-transformers.

[17] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora."*LREC 2010 Workshop on New Challenges for NLP Frameworks*, 2010. [Online]. Disponible: https://radimrehurek.com/gensim/

[18] Scaler Topics, "Introduction to ngrams in NLP", Scaler Topics, 2023. [Online]. Disponible: https://www.scaler.com/topics/nlp/ngrams-in-nlp/.

[19] Explosion AI, "spaCy: Industrial-Strength Natural Language Processing in Python", Explosion AI, 2023. [Online]. Disponible: https://spacy.io/.

[20] Python Software Foundation, "collections — Container datatypes", Python Software Foundation, 2023. [Online]. Disponible: https://docs.python.org/3/library/collections.html#collections.OrderedDict.

[21] Nik Piepenbreier, "Understanding Jaccard Similarity in Python: A Comprehensive Guide," Datagy, 2023. [Online]. Disponible: https://datagy.io/jaccard-similarity/.

[22] DeepAI, "Understanding Epochs in Machine Learning," DeepAI, 2023. [Online]. Disponible: https://deepai.org/machine-learning-glossary-and-terms/epoch.

[23] Kishore, "Understanding Epochs in Neural Networks: A Comprehensive Guide," Cogxta, 2024. [Online]. Disponible: https://blog.cogxta.com/understanding-epochs-in-neural-networks-a-comprehensive-guide/.

[24] Simplilearn, "Understanding Loss Functions in Machine Learning," Simplilearn, 2023. [Online]. Disponible: https://www.simplilearn.com/understanding-loss-functions-in-machine-learning-article.

[25] Simplilearn, "What is Pooling in Machine Learning?," Simplilearn, 2023. [Online]. Disponible: https://www.simplilearn.com/what-is-pooling-in-machine-learning-article.

[26] Bengio, Y., & Lecun, Y. (2007). *Scaling learning algorithms towards AI*. MIT Press. https://www.iro.umontreal.ca/~lisa/pointeurs/bengio+lecun_chapter2007.pdf

[27] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. https://www.deeplearningbook.org/

[28] Smith, S. L., Kindermans, P.-J., Ying, C., & Le, Q. V. (2018). *Don't Decay the Learning Rate, Increase the Batch Size*. arXiv preprint arXiv:1711.00489. https://arxiv.org/abs/1711.00489

[29] MachineLearningModels.org, "Understanding the Concept of Negative Examples in Machine Learning," MachineLearningModels.org, 2023. [Online]. Disponible: https://machinelearningmodels.org/negative-examples-in-machine-learning.

[30] Kishore, "The Role of Negative Examples in Training Neural Networks," Cogxta, 2024. [Online]. Disponible: https://blog.cogxta.com/negative-examples-in-neural-networks/.

[31] Hugging Face. "distiluse-base-multilingual-cased-v2." *Hugging Face*, 2023. [Online]. Disponible: https://huggingface.co/sentence-transformers/distiluse-base-multilingual-cased-v2

[32] Hugging Face. "sentence-similarity-spanish-es." *Hugging Face*, 2023. [Online]. Disponible: https://huggingface.co/hiiamsid/sentence_similarity_spanish_es

[33] Zhang, Junlei, Lan, Zhenzhong, y He, Junxian. Çontrastive Learning of Sentence Embeddings from Scratch."Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2023, pp. 3916-3932.

[34] N. Umasankar, "NLPAUG – A Python Library to Augment Your Text Data,.ªnalytics Vidhya, 19-Sep-2022. [Online]. Disponible:https://www.analyticsvidhya.com/blog/2021/08/nlpaug-a-python-library-to-augment-your-text-data/

[35] P. Tidke, "Text Data Augmentation in Natural Language Processing with Texattack,.ªnalytics Vidhya, 26-Feb-2022. [Online]. Disponible:https://www.analyticsvidhya.com/blog/2022/02/text-data-augmentation-in-natural-language-processing-with-texattack/

Apéndice

En esta sección se añadirá el código implementado para la ejecución de este proyecto. Este código ha sido añadido de tal forma que pueda ser facilmente ejecutable en un ambiente de Google Colab con las mínimas correcciones, como los parametros para su ejecución, la ruta de carga y lectura de los archivos que desean ser probados.

A. Preprocesamiento de Datos

```
import numpy as np
import pandas as pd
import os
import re
import nltk
import spacy
from google.colab import drive
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Montar Google Drive
drive.mount('/content/drive')

# Cargar dataset de subyacentes
columns = ['codigo_subyacente_caracteristica', 'descripcion', 'estado', '
    permitido_orf', 'desc_subya', 'desc_caracteristica', 'desc_unidad', '
    desc_empaque', 'desc_naturaleza']
db = pd.read_excel("/content/drive/MyDrive/Tesis/Datasets/SUBYACENTES.xlsx
    ", names=columns)

# Cargar datasets de las SC
folder_path = '/content/drive/MyDrive/Tesis/Datasets'
files = [file for file in os.listdir(folder_path)]

# Concatenar todos los dataframes de clientes en uno solo
dataframes = [pd.read_excel(os.path.join(folder_path, file)) for file in
    files]
combined_df_clientes = pd.concat(dataframes, ignore_index=True)

# Combinar las columnas 'DESC' y 'DESCRIPCION'
combined_df_clientes['DESCRIPCION_CLIENTE'] = combined_df_clientes['DESC'
    ].fillna(combined_df_clientes['DESCRIPCION'])
combined_df_clientes = combined_df_clientes.drop(columns=['DESC', '
    DESCRIPCION'])

\# Descartar columnas con mayor a de valores NaN
combined_df_clientes = combined_df_clientes.drop(columns=['FECHA HORA', '
    Nit. Cliente', 'CODIGO PROD. CLIENTE', 'DESCRIPCION PRODUCTOS'])

# Convertir las columnas de float a int
combined_df_clientes['ID'] = combined_df_clientes['ID'].fillna(0).astype('
    int')
```

```
combined_df_clientes['CODIGO BMC'] = combined_df_clientes['CODIGO BMC'].fillna(0).astype('int')

# Descartar columnas irrelevantes
combined_df_clientes = combined_df_clientes.drop(columns=['TIPO ID', 'ID', 'FACTOR CONVERSION'])

# Eliminar filas duplicadas
combined_df_clientes = combined_df_clientes.drop_duplicates()

# Integrar descripciones BMC
subyacentes_copia = db.rename(columns={'descripcion': 'DESCRIPCION_BMC', 'codigo_subyacente_caracteristica': 'CODIGO BMC'})
resultado_df = pd.merge(combined_df_clientes, subyacentes_copia[['CODIGO BMC', 'DESCRIPCION_BMC']], on='CODIGO BMC', how='left')

# Eliminar filas con valores NaN
resultado_df = resultado_df.dropna()

# Estandarizaci n de textos
resultado_df['DESCRIPCION_CLIENTE'] = resultado_df['DESCRIPCION_CLIENTE'].str.lower().str.strip().str.replace(r'\s+', ' ', regex=True)
resultado_df['DESCRIPCION_BMC'] = resultado_df['DESCRIPCION_BMC'].str.lower().str.strip().str.replace(r'\s+', ' ', regex=True)
resultado_df['DESCRIPCION_CLIENTE'] = resultado_df['DESCRIPCION_CLIENTE'].apply(lambda x: re.sub(r'(\D)(\d)', r'\1 \2', x))
resultado_df['DESCRIPCION_CLIENTE'] = resultado_df['DESCRIPCION_CLIENTE'].apply(lambda x: re.sub(r'(\d)(\D)', r'\1 \2', x))
resultado_df['DESCRIPCION_BMC'] = resultado_df['DESCRIPCION_BMC'].apply(lambda x: re.sub(r'(\D)(\d)', r'\1 \2', x))
resultado_df['DESCRIPCION_BMC'] = resultado_df['DESCRIPCION_BMC'].apply(lambda x: re.sub(r'(\d)(\D)', r'\1 \2', x))

# Reemplazo de abreviaturas
replacements = {r'\bg\b': 'gramos', r'\bgms\b': 'gramos', r'\bgrs\b': 'gramos', r'\bgr\b': 'gramos', r'\bkg\b': 'kilogramo', r'\bml\b': 'mililitros', r'\blt\b': 'litro'}
for key, value in replacements.items():
    resultado_df['DESCRIPCION_CLIENTE'] = resultado_df['DESCRIPCION_CLIENTE'].str.replace(key, value, regex=True)
    resultado_df['DESCRIPCION_BMC'] = resultado_df['DESCRIPCION_BMC'].str.replace(key, value, regex=True)

# Eliminar caracteres especiales
resultado_df['DESCRIPCION_CLIENTE'] = resultado_df['DESCRIPCION_CLIENTE'].str.replace(r'[^a-zA-Z0-9\s]', '', regex=True)
resultado_df['DESCRIPCION_BMC'] = resultado_df['DESCRIPCION_BMC'].str.replace(r'[^a-zA-Z0-9\s]', '', regex=True)

# Eliminar stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('spanish'))
resultado_df['DESCRIPCION_CLIENTE'] = resultado_df['DESCRIPCION_CLIENTE'].apply(lambda x: ' '.join([word for word in x.split() if word not in stop_words]))
resultado_df['DESCRIPCION_BMC'] = resultado_df['DESCRIPCION_BMC'].apply(lambda x: ' '.join([word for word in x.split() if word not in stop_words]))

# Tokenizaci n
nltk.download('punkt')
resultado_df['TOKENS_DESCRIPCION_CLIENTE'] = resultado_df['DESCRIPCION_CLIENTE'].apply(word_tokenize)
resultado_df['TOKENS_DESCRIPCION_BMC'] = resultado_df['DESCRIPCION_BMC'].apply(word_tokenize)

```

```
# Lemmatizaci n
!python -m spacy download es_core_news_sm
nlp = spacy.load('es_core_news_sm')
resultado_df['LEMMA_TOKENS_DESCRIPCION_CLIENTE'] = resultado_df['
    TOKENS_DESCRIPCION_CLIENTE'].apply(lambda tokens: [token.lemma_ for
    token in nlp(' '.join(tokens))])
resultado_df['LEMMA_TOKENS_DESCRIPCION_BMC'] = resultado_df['
    TOKENS_DESCRIPCION_BMC'].apply(lambda tokens: [token.lemma_ for token
    in nlp(' '.join(tokens))])

# Guardar el DataFrame limpio
!pip install openpyxl
from google.colab import files
resultado_df.to_excel('df_limpio.xlsx', engine='openpyxl', index=False)
files.download('df_limpio.xlsx')
```

Listing 1: Preprocesamiento de Datos en Python

B. Entrenamiento de Modelos Word2Vec

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
import ast
import nltk
from nltk.tokenize import word_tokenize
from gensim.models import Word2Vec
from gensim.models.phrases import Phrases, Phraser
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import PCA
from collections import OrderedDict

# Montar Google Drive
drive.mount('/content/drive')

# Cargar y preprocesar datos
columns = [
    'CODIGO BMC', 'CODIGO CLIENTE', 'DESCRIPCION_CLIENTE', '
        DESCRIPCION_BMC',
    'TOKENS_DESCRIPCION_CLIENTE', 'TOKENS_DESCRIPCION_BMC',
    'LEMMA_TOKENS_DESCRIPCION_CLIENTE', 'LEMMA_TOKENS_DESCRIPCION_BMC'
]
db = pd.read_excel("/content/drive/MyDrive/Tesis/df_limpio.xlsx", names=
    columns)

# Convertir representaciones de cadenas de listas en listas reales
db['TOKENS_DESCRIPCION_CLIENTE'] = db['TOKENS_DESCRIPCION_CLIENTE'].apply(
    ast.literal_eval)
db['TOKENS_DESCRIPCION_BMC'] = db['TOKENS_DESCRIPCION_BMC'].apply(ast.
    literal_eval)
db['LEMMA_TOKENS_DESCRIPCION_CLIENTE'] = db['
    LEMMA_TOKENS_DESCRIPCION_CLIENTE'].apply(ast.literal_eval)
db['LEMMA_TOKENS_DESCRIPCION_BMC'] = db['LEMMA_TOKENS_DESCRIPCION_BMC'].
    apply(ast.literal_eval)

# Preprocesar descripciones para entrenamiento
descripciones_procesadas = db['TOKENS_DESCRIPCION_CLIENTE'].tolist() + db[
    'TOKENS_DESCRIPCION_BMC'].tolist()

# Entrenar modelo Word2Vec
modelo_w2v = Word2Vec(sentences=descripciones_procesadas, vector_size=150,
     window=7, min_count=3, workers=4)
```

```
36
37 # Funci n para generar vectores de documento
38 def document_vector(model, doc):
39     vectors = [model.wv[word] for word in doc if word in model.wv]
40     return np.mean(vectors, axis=0) if vectors else np.zeros(model.
           vector_size)
41
42 # Generar vectores para descripciones de clientes y BMC
43 vectores_clientes = np.array([document_vector(modelo_w2v, desc) for desc
       in db['TOKENS_DESCRIPCION_CLIENTE']])
44 vectores_bmc = np.array([document_vector(modelo_w2v, desc) for desc in db[
       'TOKENS_DESCRIPCION_BMC']])
45
46 # Visualizar vectores usando PCA
47 vectores_combined = np.vstack((vectores_clientes[:100], vectores_bmc
       [:100]))
48 pca = PCA(n_components=2)
49 vectores_reducidos = pca.fit_transform(vectores_combined)
50 labels = ['SC'] * 100 + ['BMC'] * 100
51
52 plt.figure(figsize=(10, 6))
53 for i, label in enumerate(set(labels)):
54     plt.scatter(vectores_reducidos[np.array(labels) == label, 0],
55                 vectores_reducidos[np.array(labels) == label, 1],
56                 label=label)
57 plt.title('Visualizaci n de Vectores de Documento usando PCA')
58 plt.xlabel('Componente Principal 1')
59 plt.ylabel('Componente Principal 2')
60 plt.legend()
61 plt.grid(True)
62 plt.show()
63
64 # Calcular similitudes del coseno para pares alineados
65 similitudes_directas = [
66     cosine_similarity(vectores_clientes[i].reshape(1, -1), vectores_bmc[i
           ].reshape(1, -1))[0][0]
67     for i in range(len(vectores_clientes))
68 ]
69
70 print("Similitud Directa Promedio:", np.mean(similitudes_directas))
71
72 # Histograma de similitudes del coseno
73 plt.figure(figsize=(10, 6))
74 plt.hist(similitudes_directas, bins=50, color='blue', edgecolor='black',
       alpha=0.7)
75 plt.title('Distribuci n de Similitud del Coseno')
76 plt.xlabel('Similitud del Coseno')
77 plt.ylabel('Frecuencia')
78 plt.grid(True, which='both', linestyle='--', linewidth=0.5)
79 plt.axvline(x=np.mean(similitudes_directas), color='red', linestyle='--',
       label='Promedio')
80 plt.legend()
81 plt.show()
82
83 # Encontrar la descripci n BMC m s similar para cada cliente
84 similitudes = cosine_similarity(vectores_clientes, vectores_bmc)
85 indices_max_sim = similitudes.argmax(axis=1)
86 valores_max_sim = similitudes.max(axis=1)
87
88 # Verificaci n de alineaciones correctas
89 correctos = sum([1 for i in range(len(similitudes)) if similitudes[i].
       argmax() == i])
90 print(f"Alineaciones correctas: {correctos} de {len(similitudes)}")
91 print(f"Porcentaje de alineaci n correcta: {(correctos / len(similitudes)
       ) * 100}%")
92
```

```
# Calcular alineaciones correctas dentro del top 3
correctos_top3 = sum([1 for i in range(len(similitudes)) if i in np.
    argsort(similitudes[i])[-3:][::-1]])
print(f"Alineaciones correctas dentro del top 3: {correctos_top3} de {len(
    similitudes)}")
print(f"Porcentaje de alineaci n correcta dentro del top 3: {(
    correctos_top3 / len(similitudes)) * 100}%")
```

Listing 2: Entrenamiento y Evaluación con Word Embeddings en Python

B.1. Entrenamiento con N-gramas

```
  # Preprocesar y generar vectores con n-grams
phrases = Phrases(descripciones_procesadas, min_count=2, threshold=10)
bigram = Phraser(phrases)
descripciones_con_phrases = bigram[descripciones_procesadas]
modelo_w2v = Word2Vec(sentences=descripciones_con_phrases, vector_size
    =150, window=7, min_count=3, workers=4)
vectores_clientes = np.array([document_vector(modelo_w2v, desc) for desc
    in db['TOKENS_DESCRIPCION_CLIENTE']])
vectores_bmc = np.array([document_vector(modelo_w2v, desc) for desc in db[
    'TOKENS_DESCRIPCION_BMC']])
similitudes_directas = [
    cosine_similarity(vectores_clientes[i].reshape(1, -1), vectores_bmc[i
        ].reshape(1, -1))[0][0]
    for i in range(len(vectores_clientes))
]

print("Similitud Directa Promedio con N-grams:", np.mean(
    similitudes_directas))

# Verificaci n de alineaciones correctas con n-grams
similitudes = cosine_similarity(vectores_clientes, vectores_bmc)
correctos = sum([1 for i in range(len(similitudes)) if similitudes[i].
    argmax() == i])
print(f"Alineaciones correctas: {correctos} de {len(similitudes)}")
print(f"Porcentaje de alineaci n correcta: {(correctos / len(similitudes)
    ) * 100}%")
```

Listing 3: Entrenamiento y Evaluación con N-gramas

B.2. Entrenamiento con Lemmas

```


# Entrenar Word2Vec con lemas
descripciones_procesadas = db['LEMMA_TOKENS_DESCRIPCION_CLIENTE'].tolist()
     + db['LEMMA_TOKENS_DESCRIPCION_BMC'].tolist()
modelo_w2v = Word2Vec(sentences=descripciones_procesadas, vector_size=150,
     window=7, min_count=3, workers=4)
vectores_clientes = np.array([document_vector(modelo_w2v, desc) for desc
    in db['LEMMA_TOKENS_DESCRIPCION_CLIENTE']])
vectores_bmc = np.array([document_vector(modelo_w2v, desc) for desc in db[
    'LEMMA_TOKENS_DESCRIPCION_BMC']])
similitudes_directas = [
    cosine_similarity(vectores_clientes[i].reshape(1, -1), vectores_bmc[i
        ].reshape(1, -1))[0][0]
    for i in range(len(vectores_clientes))
]

print("Similitud Directa Promedio con Lemas:", np.mean(similitudes\
    _directas))
```

```
14
15 \# Verificaci n de alineaciones correctas con lemas
16 similitudes = cosine_similarity(vectores_clientes, vectores_bmc)
17 correctos = sum([1 for i in range(len(similitudes)) if similitudes[i].
       argmax() == i])
18 print(f"Alineaciones correctas: {correctos} de {len(similitudes)}")
19 print(f"Porcentaje de alineaci n correcta: {(correctos / len(similitudes)
       ) * 100}%")
20
21 \# Calcular alineaciones correctas dentro del top 3 con lemas
22 correctos_top3 = sum([1 for i in range(len(similitudes)) if i in np.
       argsort(similitudes[i])[-3:][::-1]])
23 print(f"Alineaciones correctas dentro del top 3: {correctos_top3} de {len(
       similitudes)}")
24 print(f"Porcentaje de alineaci n correcta dentro del top 3: {(correctos\
       _top3 / len(similitudes)) * 100}%")
```

Listing 4: Entrenamiento y Evaluación con Lemmas

B.3. Entrenamiento con Descripciones Combinadas

```
1 # Combinar descripciones y reentrenar Word2Vec
2 desc_clientes = db['DESCRIPCION_CLIENTE'].tolist()
3 desc_bmc = db['DESCRIPCION_BMC'].tolist()
4 descripciones_combinadas = [f"{desc_clientes[i]} {desc_bmc[i]}" for i in
      range(len(desc_bmc))]
5 tokens = [word_tokenize(desc) for desc in descripciones_combinadas]
6 modelo_w2v = Word2Vec(sentences=descripciones_combinadas, vector_size=150,
       window=7, min_count=3, workers=4)
7 vectores_clientes = np.array([document_vector(modelo_w2v, desc) for desc
      in db['TOKENS_DESCRIPCION_CLIENTE']])
8 vectores_bmc = np.array([document_vector(modelo_w2v, desc) for desc in db[
      'TOKENS_DESCRIPCION_BMC']])
9 similitudes_directas = [
10     cosine_similarity(vectores_clientes[i].reshape(1, -1), vectores_bmc[i
           ].reshape(1, -1))[0][0]
11     for i in range(len(vectores_clientes))
12 ]
13
14 print("Similitud Directa Promedio con Descripciones Combinadas:", np.mean(
      similitudes_directas))
15
16 # Verificaci n de alineaciones correctas con descripciones combinadas
17 similitudes = cosine_similarity(vectores_clientes, vectores_bmc)
18 correctos = sum([1 for i in range(len(similitudes)) if similitudes[i].
      argmax() == i])
19 print(f"Alineaciones correctas: {correctos} de {len(similitudes)}")
20 print(f"Porcentaje de alineaci n correcta: {(correctos / len(similitudes)
      ) * 100}%")
```

C. Entrenamiento de Modelo con spaCy

```
1 import spacy
2 import numpy as np
3 from sklearn.metrics.pairwise import cosine_similarity
4 import matplotlib.pyplot as plt
5 from sklearn.decomposition import PCA
6 from collections import OrderedDict
7
8 # Cargar el modelo preentrenado en espa ol
```

```
nlp = spacy.load('es_core_news_md')

# Funci n para obtener el vector de un texto
def get_vector(text):
    return nlp(text).vector if isinstance(text, str) else np.zeros(nlp.
        vocab.vectors.shape[1])

# Asegurarse de que todas las descripciones sean cadenas
db['DESCRIPCION_CLIENTE'] = db['DESCRIPCION_CLIENTE'].astype(str)
db['DESCRIPCION_BMC'] = db['DESCRIPCION_BMC'].astype(str)

# Precalcular vectores para descripciones de clientes
vectores_cliente = np.array([get_vector(desc) for desc in db['
    DESCRIPCION_CLIENTE']])

# Obtener descripciones  nicas  de BMC y mapear cada descripci n  nica  a
     sus  ndices
descripciones_bmc_unicas = list(OrderedDict.fromkeys(db['DESCRIPCION_BMC'
    ]))
indices_bmc_unicos = {desc: i for i, desc in enumerate(
    descripciones_bmc_unicas)}
vectores_bmc_unicos = np.array([get_vector(desc) for desc in
    descripciones_bmc_unicas])

# Calcular la similitud entre cada descripci n de cliente y todas las
    descripciones  nicas  de BMC
resultados_similitud = []
correctos = 0

for idx, vector_cliente in enumerate(vectores_cliente):
    # Calcular la similitud del coseno para el vector del cliente contra
        todos los vectores BMC
    similarities = cosine_similarity(vector_cliente.reshape(1, -1),
        vectores_bmc_unicos).flatten()

    # Encontrar los  ndices  de las tres puntuaciones de similitud m s
        altas
    top_3_indices = np.argsort(-similarities)[:3]
    max_similarity = similarities[top_3_indices[0]]

    # Almacenar los resultados
    descripcion_cliente = db['DESCRIPCION_CLIENTE'][idx]
    descripcion_bmc_similar = descripciones_bmc_unicas[top_3_indices[0]]
    correct_match_index = indices_bmc_unicos[db['DESCRIPCION_BMC'][idx]]
    is_correct_match = correct_match_index in top_3_indices

    if is_correct_match:
        correctos += 1

    resultados_similitud.append((descripcion_cliente,
        descripcion_bmc_similar, max_similarity, is_correct_match))

# Calcular y imprimir el total y porcentaje de coincidencias correctas
total = len(resultados_similitud)
porcentaje_correctos = (correctos / total) * 100

print(f"Correctamente alineados (dentro del top 3): {correctos} de {total}
    ")
print(f"Porcentaje de alineaci n correcta (top 3): {porcentaje_correctos
    :.2f}%")

# Asegurarse de que vectores_clientes[i] y vectores_bmc[i]  est n
    alineados correctamente.

# Calcular similitudes del coseno directamente para parejas alineadas
```

```
vectores_bmc = np.array([get_vector(desc) for desc in db['DESCRIPCION_BMC']])
similitudes_directas = [cosine_similarity(vectores_cliente[i].reshape(1, -1), vectores_bmc[i].reshape(1, -1))[0][0] for i in range(len(vectores_cliente))]

# similitudes_directas contiene ahora la similitud de coseno para cada par alineado de descripciones
print("Similitud promedio directa:", np.mean(similitudes_directas))

# Visualizaci n con PCA para vectores spaCy
vectores_clientes_sample = vectores_cliente[:100]
vectores_bmc_sample = vectores_bmc[:100]

vectores_combined = np.vstack((vectores_clientes_sample, vectores_bmc_sample))

pca = PCA(n_components=2)
vectores_reducidos = pca.fit_transform(vectores_combined)

labels = ['SC'] * len(vectores_clientes_sample) + ['BMC'] * len(vectores_bmc_sample)

plt.figure(figsize=(10, 6))
for i, label in enumerate(set(labels)):
    plt.scatter(vectores_reducidos[np.array(labels) == label, 0],
                vectores_reducidos[np.array(labels) == label, 1],
                label=label)
plt.title('Visualizaci n de Vectores de Documento utilizando PCA')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.legend()
plt.grid(True)
plt.show()
```

Listing 5: Entrenamiento y Evaluacion con spaCy en Python

D. Word2vec con descripciones subyacentes

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
import ast
from nltk.tokenize import word_tokenize
from gensim.models import Word2Vec
from sklearn.metrics.pairwise import cosine_similarity
from collections import OrderedDict

drive.mount('/content/drive')

columns = ['CODIGO BMC', 'CODIGO CLIENTE', 'DESCRIPCION_CLIENTE', 'DESCRIPCION_BMC', 'TOKENS_DESCRIPCION_CLIENTE', 'TOKENS_DESCRIPCION_BMC', 'LEMMA_TOKENS_DESCRIPCION_CLIENTE', 'LEMMA_TOKENS_DESCRIPCION_BMC', 'DESCRIPCION_SUBYACENTE', 'TOKENS_DESCRIPCION_SUBYACENTE']
db = pd.read_excel("/content/drive/MyDrive/Tesis/df_limpio_2.xlsx", names=columns)
db.head()

# Convertir las cadenas de la columna 'tokens' a listas
```

```
19 db['TOKENS_DESCRIPCION_CLIENTE'] = db['TOKENS_DESCRIPCION_CLIENTE'].apply(
       ast.literal_eval)
20 db['TOKENS_DESCRIPCION_BMC'] = db['TOKENS_DESCRIPCION_BMC'].apply(ast.
       literal_eval)
21 db['LEMMA_TOKENS_DESCRIPCION_CLIENTE'] = db['
       LEMMA_TOKENS_DESCRIPCION_CLIENTE'].apply(ast.literal_eval)
22 db['LEMMA_TOKENS_DESCRIPCION_BMC'] = db['LEMMA_TOKENS_DESCRIPCION_BMC'].
       apply(ast.literal_eval)
23 db['TOKENS_DESCRIPCION_SUBYACENTE'] = db['TOKENS_DESCRIPCION_SUBYACENTE'].
       apply(ast.literal_eval)
24
25 # ENTRENAMIENTO CON TOKENS
26
27 # Aplicar el preprocesamiento a las descripciones
28 descripciones_procesadas = db['TOKENS_DESCRIPCION_CLIENTE'].tolist() + db[
       'TOKENS_DESCRIPCION_BMC'].tolist()
29
30 # Entrenar el modelo Word2Vec con las descripciones combinadas y
       procesadas
31 modelo_w2v = Word2Vec(sentences=descripciones_procesadas, vector_size=150,
        window=7, min_count=3, workers=4)
32
33 # Una vez que el modelo est  entrenado, generamos vectores para cualquier
        descripci n
34 def document_vector(model, doc):
35     # Asegurarse de que el documento est  procesado (tokenizado y limpio)
36     vectors = [model.wv[word] for word in doc if word in model.wv]
37     if len(vectors) == 0:
38         return np.zeros(model.vector_size)  # Retorna un vector de ceros
               si no hay vectores v lidos
39     else:
40         return np.mean(vectors, axis=0)
41
42 # Generar vectores para las descripciones de los clientes
43 vectores_clientes = np.array([document_vector(modelo_w2v, desc) for desc
       in db['TOKENS_DESCRIPCION_CLIENTE']])
44 # Generar vectores para las descripciones simplificadas del BMC
45 vectores_sub = np.array([document_vector(modelo_w2v, desc) for desc in db[
       'TOKENS_DESCRIPCION_SUBYACENTE']])
46
47 # Funci n para convertir listas en tuplas
48 def list_to_tuple(lists):
49     return tuple(lists)
50
51 # Convertir cada lista de tokens a una tupla
52 tuples_sub = map(list_to_tuple, db['TOKENS_DESCRIPCION_SUBYACENTE'])
53
54 # Crear un OrderedDict para eliminar duplicados y mantener el orden
55 descripciones_subyacentes_unicas = OrderedDict.fromkeys(tuples_sub)
56
57 # Generar vectores para las descripciones simplificadas  nicas  del BMC
58 vectores_sub_unicos = np.array([document_vector(modelo_w2v, desc) for desc
       in descripciones_subyacentes_unicas])
59
60 # Ahora tenemos un array de vectores de BMC sin duplicados que podemos
       usar para calcular la similitud
61 num_vectores_sub_originales = len(vectores_sub)
62 num_vectores_sub_unicos = len(vectores_sub_unicos)
63 print("N mero de vectores subyacentes originales: " + str(
       num_vectores_sub_originales))
64 print("N mero de vectores subyacentes  nicos : " + str(
       num_vectores_sub_unicos))
65
66 # Nombres de las categor as
67 categorias = ['Subyacentes Originales', 'Subyacentes  nicos ']
68
```

```
# Cantidades para cada categor a
cantidades = [num_vectores_sub_originales, num_vectores_sub_unicos]

plt.figure(figsize=(8, 5))
plt.bar(categorias, cantidades, color=['blue', 'green'])

plt.title('Comparaci n del Tama o de Vectores Subyacentes')
plt.xlabel('Categor a')
plt.ylabel('N mero de Vectores')
plt.show()

# Calcular similitudes del coseno directamente para parejas alineadas
similitudes_directas = [cosine_similarity(vectores_clientes[i].reshape(1, -1), vectores_sub[i].reshape(1, -1))[0][0] for i in range(len(vectores_clientes))]

# similitudes_directas contiene ahora la similitud de coseno para cada par alineado de descripciones
print("Similitud promedio directa:", np.mean(similitudes_directas))

# Crear un histograma para visualizar la distribuci n de las similitudes
plt.figure(figsize=(10, 6))
plt.hist(similitudes_directas, bins=50, color='blue', edgecolor='black', alpha=0.7)
plt.title('Distribuci n de las Similitudes del Coseno')
plt.xlabel('Similitud del Coseno')
plt.ylabel('Frecuencia')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.axvline(x=np.mean(similitudes_directas), color='red', linestyle='--', label='Promedio')
plt.legend()
plt.show()

# Asumiendo que vectores_clientes y vectores_sub_unicos est n ya definidos y preparados

# "similitudes" es una matriz de arrays que guarda la similitud del coseno de cada comparaci n
# ejemplo:
# [[cs(desc_client1,desc_bmc1),...cs(desc_client1,desc_bmcN)],
#  [cs(desc_client2,desc_bmc1),...cs(desc_client2,desc_bmcN)],...
#  [cs(desc_clientN,desc_bmc1),...cs(desc_clientN,desc_bmcN)]]

similitudes = cosine_similarity(vectores_clientes, vectores_sub)

# Con las similitudes calculadas, proceder a encontrar la descripci n del BMC m s similar para cada cliente
indices_max_sim = similitudes.argmax(axis=1)  #  ndices  de las descripciones del BMC m s similares para cada cliente

# Opcionalmente, calcular la similitud m xima para verificar la correspondencia
valores_max_sim = similitudes.max(axis=1)

# Verificaci n
correctos = 0
total = len(similitudes)

for i in range(total):
    # El  ndice  de la descripci n del BMC con la mayor similitud
    indice_max_sim = similitudes[i].argmax()
    # Verificar si este  ndice  coincide con el  ndice  de la descripci n del BMC alineada
    if indice_max_sim == i:
        correctos += 1

```

```
print(f"Correctamente alineados: {correctos} de {total}")
# Calcular el porcentaje de correctos
porcentaje_correctos = (correctos / total) * 100
print(f"Porcentaje de alineaci n correcta: {porcentaje_correctos}%")

# Verificaci n en el top 3
correctos_top3 = 0

for i in range(len(similitudes)):
    indices_top3_sim = np.argsort(similitudes[i])[-3:][::-1]  # Ordenar los  ndices  de las similitudes de mayor a menor y tomar los top 3
    if i in indices_top3_sim:  # Verificar si el  ndice  correcto (i) est  dentro de los top 3  ndices  de similitud
        correctos_top3 += 1

porcentaje_correctos_top3 = (correctos_top3 / len(similitudes)) * 100
print(f"Correctamente alineados dentro de los top 3: {correctos_top3} de {len(similitudes)}")
print(f"Porcentaje de alineaci n correcta dentro de los top 3: {porcentaje_correctos_top3}%")

# Obtener las 3 descripciones m s similares  nicas  para la descripci n del cliente i
correctos_top3 = 0

for i in range(len(similitudes)):
    indices_sorted = np.argsort(similitudes[i])[::-1]  # Ordenar en orden descendente
    unique_descriptions = set()
    top3_indices = []

    for idx in indices_sorted:  # Obtener las top 3 descripciones  nicas
        desc = db['DESCRIPCION_SUBYACENTE'].iloc[idx]
        if desc not in unique_descriptions:
            unique_descriptions.add(desc)
            top3_indices.append(idx)
        if len(top3_indices) == 3:
            break

    if i in top3_indices:  # Verificar si el  ndice  correcto (i) est  dentro de los top 3  ndices  de similitud
        correctos_top3 += 1

porcentaje_correctos_top3 = (correctos_top3 / len(similitudes)) * 100
print(f"Correctamente alineados dentro de los top 3  nicos : {correctos_top3} de {len(similitudes)}")
print(f"Porcentaje de alineaci n correcta dentro de los top 3  nicos : {porcentaje_correctos_top3}%")
```

Listing 6: Entrenamiento y Evaluacion Word2Vec con subyacentes

E. spaCy con Descripciones Subyacentes

```
import spacy
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from collections import OrderedDict

# Cargar el modelo preentrenado en espa ol
nlp = spacy.load('es_core_news_md')

# Funci n para obtener el vector de un texto
def get_vector(text):
```

```
    return nlp(text).vector if isinstance(text, str) else np.zeros(nlp.
        vocab.vectors.shape[1])

# Asegurarse de que todas las descripciones sean cadenas
db['DESCRIPCION_CLIENTE'] = db['DESCRIPCION_CLIENTE'].astype(str)
db['DESCRIPCION_SUBYACENTE'] = db['DESCRIPCION_SUBYACENTE'].astype(str)

# Precalcular vectores para descripciones de clientes
vectores_cliente = np.array([get_vector(desc) for desc in db['
    DESCRIPCION_CLIENTE']])

# Obtener descripciones subyacentes  nicas  y mapear cada descripci n
     nica  a sus  ndices
descripciones_sub_unicas = list(OrderedDict.fromkeys(db['
    DESCRIPCION_SUBYACENTE']))
indices_sub_unicos = {desc: i for i, desc in enumerate(
    descripciones_sub_unicas)}
vectores_sub = np.array([get_vector(desc) for desc in
    descripciones_sub_unicas])

# Calcular la similitud entre cada descripci n de cliente y todas las
    descripciones subyacentes  nicas
resultados_similitud = []
correctos = 0

for idx, vector_cliente in enumerate(vectores_cliente):
    # Calcular la similitud del coseno para el vector del cliente contra
        todos los vectores subyacentes
    similarities = cosine_similarity(vector_cliente.reshape(1, -1),
        vectores_sub)

    # Encontrar el  ndice  de la mayor puntuaci n de similitud
    max_index = np.argmax(similarities)
    max_similarity = similarities[0, max_index]

    # Verificar si la coincidencia es correcta
    correct_match_index = indices_sub_unicos[db['DESCRIPCION_SUBYACENTE'][
        idx]]
    is_correct_match = (max_index == correct_match_index)

    if is_correct_match:
        correctos += 1

# Calcular y mostrar el total y porcentaje de coincidencias correctas
total = len(vectores_cliente)
porcentaje_correctos = (correctos / total) * 100

print(f"Correctamente alineados: {correctos} de {total}")
print(f"Porcentaje de alineaci n correcta: {porcentaje_correctos:.2f}%")

# Calcular la similitud entre cada descripci n de cliente y todas las
    descripciones subyacentes  nicas
resultados_similitud = []
correctos_top3 = 0

for idx, vector_cliente in enumerate(vectores_cliente):
    # Calcular la similitud del coseno para el vector del cliente contra
        todos los vectores subyacentes
    similarities = cosine_similarity(vector_cliente.reshape(1, -1),
        vectores_sub).flatten()

    # Obtener los  ndices  de las tres mayores similitudes
    top_three_indices = np.argsort(similarities)[-3:][::-1]

    # Verificar si la coincidencia correcta est  en el top 3
```

```
    correct_match_index = indices_sub_unicos[db['DESCRIPCION_SUBYACENTE'].
        iloc[idx]]
    is_correct_in_top3 = correct_match_index in top_three_indices

    if is_correct_in_top3:
        correctos_top3 += 1

# Calcular y mostrar la precisi n del top 3
accuracy_top3 = (correctos_top3 / len(vectores_cliente)) * 100
print(f"Correctamente alineados dentro del top 3: {correctos_top3} de {len
    (vectores_cliente)}")
print(f"Porcentaje de alineaci n correcta dentro del top 3: {
    accuracy_top3:.2f}%")
```

E.1. Jaccard Score

```

from sklearn.metrics import jaccard_score

# Funci n para convertir texto a conjunto de palabras clave (tokens)
def text_to_set(text):
    doc = nlp(text.lower())
    return set(token.lemma_ for token in doc if not token.is_stop and not
        token.is_punct and token.is_alpha)

# Precomputar conjuntos de palabras para cada descripci n
conjuntos_cliente = [text_to_set(desc) for desc in db['DESCRIPCION_CLIENTE
    ']]
conjuntos_sub = [text_to_set(desc) for desc in descripciones_sub_unicas]

# Calcular la similitud de Jaccard
correctos_top3_jaccard = 0

for idx, conjunto_cliente in enumerate(conjuntos_cliente):
    jaccard_similarities = np.array([len(conjunto_cliente & conjunto_sub)
        / len(conjunto_cliente | conjunto_sub) for conjunto_sub in
        conjuntos_sub])

    # Obtener los  ndices  de las tres mayores similitudes
    top_three_indices = np.argsort(jaccard_similarities)[-3:][::-1]

    # Verificar si la coincidencia correcta est  en el top 3
    correct_match_index = indices_sub_unicos[db['DESCRIPCION_SUBYACENTE'].
        iloc[idx]]
    is_correct_in_top3 = correct_match_index in top_three_indices

    if is_correct_in_top3:
        correctos_top3_jaccard += 1

# Calcular y mostrar la precisi n del top 3 con Jaccard
accuracy_top3_jaccard = (correctos_top3_jaccard / len(conjuntos_cliente))
    * 100
print(f"Correctamente alineados dentro del top 3 (Jaccard): {
    correctos_top3_jaccard} de {len(conjuntos_cliente)}")
print(f"Porcentaje de alineaci n correcta (Jaccard) dentro del top 3: {
    accuracy_top3_jaccard:.2f}%")


# Calcular la precisi n del top 1 con Jaccard
correctos_top1_jaccard = 0

for idx, conjunto_cliente in enumerate(conjuntos_cliente):
    jaccard_similarities = np.array([len(conjunto_cliente & conjunto_sub)
        / len(conjunto_cliente | conjunto_sub) for conjunto_sub in
```

```
        conjuntos_sub])

    # Obtener el  ndice  de la mayor similitud
    top_index = np.argmax(jaccard_similarities)

    # Verificar si la descripci n correcta est  en el top 1
    correct_match_index = indices_sub_unicos[db['DESCRIPCION_SUBYACENTE'].
        iloc[idx]]
    is_correct_top1 = correct_match_index == top_index

    if is_correct_top1:
        correctos_top1_jaccard += 1

# Calcular y mostrar la precisi n del top 1 con Jaccard
accuracy_top1_jaccard = (correctos_top1_jaccard / len(conjuntos_cliente))
    * 100
print(f"Correctamente alineados dentro del top 1 (Jaccard): {
    correctos_top1_jaccard} de {len(conjuntos_cliente)}")
print(f"Porcentaje de alineaci n correcta (Jaccard) dentro del top 1: {
    accuracy_top1_jaccard:.2f}%")
```

Listing 7: Evaluación con Jaccard Score

E.2. Jaccard Score + Cosine similarity

```
# Calcular puntuaciones de similitud combinadas
correctos_top3_combinados = 0

for idx, (vector_cliente, conjunto_cliente) in enumerate(zip(
    vectores_cliente, conjuntos_cliente)):
    cosine_sims = cosine_similarity(vector_cliente.reshape(1, -1),
        vectores_sub).flatten()
    jaccard_sims = np.array([len(conjunto_cliente & conjunto_sub) / len(
        conjunto_cliente | conjunto_sub) for conjunto_sub in conjuntos_sub
        ])

    # Combinar las puntuaciones promedi ndolas
    combined_scores = (cosine_sims + jaccard_sims) / 2

    # Determinar las tres mejores coincidencias basadas en las
        puntuaciones combinadas
    top_three_indices = np.argsort(combined_scores)[-3:][::-1]

    # Verificar si la coincidencia correcta est  en el top 3
    correct_match_index = indices_sub_unicos[db['DESCRIPCION_SUBYACENTE'].
        iloc[idx]]
    is_correct_in_top3 = correct_match_index in top_three_indices

    if is_correct_in_top3:
        correctos_top3_combinados += 1

# Calcular y mostrar la precisi n del top 3 con puntuaciones combinadas
accuracy_top3_combinados = (correctos_top3_combinados / len(
    vectores_cliente)) * 100
print(f"Correctamente alineados dentro del top 3 (combinado): {
    correctos_top3_combinados} de {len(vectores_cliente)}")
print(f"Porcentaje de alineaci n correcta (combinado) dentro del top 3: {
    accuracy_top3_combinados:.2f}%")

# Calcular la precisi n del top 1 con puntuaciones combinadas
correctos_top1_combinados = 0

for idx, (vector_cliente, conjunto_cliente) in enumerate(zip(
    vectores_cliente, conjuntos_cliente)):
```

```
    cosine_sims = cosine_similarity(vector_cliente.reshape(1, -1),
        vectores_sub).flatten()
    jaccard_sims = np.array([len(conjunto_cliente & conjunto_sub) / len(
        conjunto_cliente | conjunto_sub) for conjunto_sub in conjuntos_sub
        ])

    # Combinar las puntuaciones promedi ndolas
    combined_scores = (cosine_sims + jaccard_sims) / 2

    # Determinar la mejor coincidencia basada en las puntuaciones
        combinadas
    top_one_index = np.argmax(combined_scores)

    # Verificar si la coincidencia correcta est  en el top 1
    correct_match_index = indices_sub_unicos[db['DESCRIPCION_SUBYACENTE'].
        iloc[idx]]
    is_correct_in_top1 = correct_match_index == top_one_index

    if is_correct_in_top1:
        correctos_top1_combinados += 1

# Calcular y mostrar la precisi n del top 1 con puntuaciones combinadas
accuracy_top1_combinados = (correctos_top1_combinados / len(
    vectores_cliente)) * 100
print(f"Correctamente alineados dentro del top 1 (combinado): {
    correctos_top1_combinados} de {len(vectores_cliente)}")
print(f"Porcentaje de alineaci n correcta (combinado) dentro del top 1: {
    accuracy_top1_combinados:.2f}%")
```

Listing 8: Evaluación con Jaccard Score + Cosine Similarity

F. Entrenamiento de modelos SBERT

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
from sentence_transformers import SentenceTransformer, util
from google.colab import drive

drive.mount("/content/drive")

columns =  ['CODIGO BMC','CODIGO CLIENTE','DESCRIPCION_CLIENTE','
    DESCRIPCION_BMC','TOKENS_DESCRIPCION_CLIENTE','TOKENS_DESCRIPCION_BMC'
    ,'LEMMA_TOKENS_DESCRIPCION_CLIENTE','LEMMA_TOKENS_DESCRIPCION_BMC']
db = pd.read_excel("/content/drive/MyDrive/Tesis/df_limpio.xlsx",names =
    columns)

# Usamos un modelo preentrenado

model = SentenceTransformer('hiiamsid/sentence_similarity_spanish_es')

# Descripciones BMC  nicas  y sus embeddings
unique_bmc_descriptions = db['DESCRIPCION_BMC'].unique()
unique_bmc_embeddings = model.encode(unique_bmc_descriptions,
    convert_to_tensor=True)

# Calcular embeddings para las descripciones de clientes
client_embeddings = model.encode(db['DESCRIPCION_CLIENTE'].tolist(),
    convert_to_tensor=True)

# Calcular similitudes de coseno solo para descripciones BMC  nicas
```

```
cosine_similarities = util.cos_sim(client_embeddings,
    unique_bmc_embeddings)

# Crear un DataFrame de similitudes de coseno con descripciones BMC
     nicas  como columnas
similarity_df = pd.DataFrame(cosine_similarities.numpy(), columns=
    unique_bmc_descriptions, index=db['DESCRIPCION_CLIENTE'].tolist())

# Encontrar la mejor coincidencia para cada descripci n de cliente
top_match = similarity_df.idxmax(axis=1)

actual_bmc_descriptions = db['DESCRIPCION_BMC'].tolist()

def check_actual_is_top(actual, top):
    return actual == top

results = [check_actual_is_top(actual_bmc_descriptions[i], top_match.iloc[
    i]) for i in range(len(db))]

db['Actual_is_Top'] = results

# Calcular la precisi n o porcentaje de coincidencias correctas
accuracy = sum(results) / len(results) * 100
print(f"Accuracy: {accuracy:.2f}%")

# Encontrar las 3 mejores coincidencias para cada descripci n de cliente
top_matches = similarity_df.apply(lambda x: list(x.nlargest(3).index),
    axis=1)

def check_actual_in_top(actual, top):
    return actual in top

results = [check_actual_in_top(actual_bmc_descriptions[i], top_matches.
    iloc[i]) for i in range(len(db))]

db['Actual_in_Top3'] = results

# Calcular la precisi n o porcentaje de coincidencias correctas
accuracy = sum(results) / len(results) * 100
print(f"Accuracy: {accuracy:.2f}%")
```

Listing 9: Entrenamiento y Evaluación de modelo SBERT

F.1. SBERT y Descripciones subyacentes

```
    columns =  ['CODIGO BMC','CODIGO CLIENTE','DESCRIPCION_CLIENTE','
        DESCRIPCION_BMC','TOKENS_DESCRIPCION_CLIENTE','
        TOKENS_DESCRIPCION_BMC','LEMMA_TOKENS_DESCRIPCION_CLIENTE','
        LEMMA_TOKENS_DESCRIPCION_BMC', 'DESCRIPCION_SUBYACENTE', '
        TOKENS_DESCRIPCION_SUBYACENTE']
db = pd.read_excel("/content/drive/MyDrive/Tesis/df_limpio_2.xlsx",names =
     columns)

# Descripciones Subyacentes  nicas  y sus embeddings
unique_sub_descriptions = db['DESCRIPCION_SUBYACENTE'].unique()
unique_sub_embeddings = model.encode(unique_sub_descriptions,
    convert_to_tensor=True)

# Calcular similitudes de coseno solo para descripciones BMC  nicas
cosine_similarities = util.cos_sim(client_embeddings,
    unique_sub_embeddings)

similarity_df = pd.DataFrame(cosine_similarities.numpy(), columns=
    unique_sub_descriptions, index=db['DESCRIPCION_CLIENTE'].tolist())
```

```
12
13 # Encontrar la mejor coincidencia para cada descripci n de cliente
14 top_match = similarity_df.idxmax(axis=1)
15
16 # Asumir que 'DESCRIPCION_SUBYACENTE' en tu DataFrame db contiene las
       descripciones subyacentes reales para cada cliente
17 actual_sub_descriptions = db['DESCRIPCION_SUBYACENTE'].tolist()
18
19 results = [check_actual_is_top(actual_sub_descriptions[i], top_match.iloc[
       i]) for i in range(len(db))]
20
21 db['Actual_is_Top'] = results
22
23 # Calcular la precisi n o porcentaje de coincidencias correctas
24 accuracy = sum(results) / len(results) * 100
25 print(f"Accuracy: {accuracy:.2f}%")
26
27 # Encontrar las 3 mejores coincidencias para cada descripci n de cliente
28 top_matches = similarity_df.apply(lambda x: list(x.nlargest(3).index),
       axis=1)
29
30 results = [check_actual_in_top(actual_sub_descriptions[i], top_matches.
       iloc[i]) for i in range(len(db))]
31
32 db['Actual_in_Top3'] = results
33
34 # Calcular la precisi n o porcentaje de coincidencias correctas
35 accuracy = sum(results) / len(results) * 100
36 print(f"Accuracy: {accuracy:.2f}%")
```

Listing 10: Evaluación de SBERT con Descripciones subyacentes

F.2. SBERT con Jaccard score + Cosine similarity

```
1
2 # Funci n para calcular la similitud de Jaccard
3 def jaccard_similarity(set1, set2):
4     intersection = len(set1.intersection(set2))
5     union = len(set1.union(set2))
6     return intersection / union if union != 0 else 0
7
8 # Calcular las similitudes de Jaccard
9 client_sets = [set(desc.split()) for desc in db['DESCRIPCION_CLIENTE']]
10 unique_sub_sets = [set(desc.split()) for desc in unique_sub_descriptions]
11
12 jaccard_similarities = np.zeros_like(cosine_similarities)
13
14 for i, client_set in enumerate(client_sets):
15     for j, sub_set in enumerate(unique_sub_sets):
16         jaccard_similarities[i, j] = jaccard_similarity(client_set,
               sub_set)
17
18 # Combinar similitudes de coseno y Jaccard
19 combined_scores = (cosine_similarities + jaccard_similarities) / 2
20
21 # Encontrar las 3 mejores coincidencias basadas en las puntuaciones
       combinadas
22 top_3_indices = np.argsort(-combined_scores, axis=1)[:, :3]
23 top_matches = [[unique_sub_descriptions[j] for j in row] for row in
       top_3_indices]
24
25 # Verificar si la descripci n subyacente actual est  en el top 3
26 actual_in_top3 = [db['DESCRIPCION_SUBYACENTE'].iloc[i] in top_matches[i]
       for i in range(len(db))]
```

```
27
28 # Calcular la precisi n general
29 accuracy = np.mean(actual_in_top3) * 100
30
31 # Mostrar resultados
32 db['Actual_in_Top3'] = actual_in_top3
33 print(f"Combined Accuracy: {accuracy:.2f}%")
34
35 # Obtener solo el top match
36 top_indices = np.argmax(combined_scores, axis=1)
37 top_matches = [unique_sub_descriptions[j] for j in top_indices]
38
39 # Verificar si la descripci n subyacente actual es el top match
40 actual_in_top = [db['DESCRIPCION_SUBYACENTE'].iloc[i] == top_matches[i]
       for i in range(len(db))]
41
42 # Calcular la precisi n general
43 accuracy = np.mean(actual_in_top) * 100
44
45 # Mostrar resultados
46 db['Actual_in_Top'] = actual_in_top
47 print(f"Top Match Accuracy: {accuracy:.2f}%")
```

G. SBERT Fine-tunned

```
1
2 import time
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 import torch
7 from sentence_transformers import SentenceTransformer, InputExample,
      losses, models, util
8 from torch.utils.data import DataLoader
9 from google.colab import drive
10
11 # Montar Google Drive
12 drive.mount("/content/drive")
13
14 # Cargar datos
15 columns =  ['CODIGO BMC','CODIGO CLIENTE','DESCRIPCION_CLIENTE','
      DESCRIPCION_BMC','TOKENS_DESCRIPCION_CLIENTE','TOKENS_DESCRIPCION_BMC'
      ,'LEMMA_TOKENS_DESCRIPCION_CLIENTE','LEMMA_TOKENS_DESCRIPCION_BMC']
16 db = pd.read_excel("/content/drive/MyDrive/Tesis/df_limpio.xlsx", names=
      columns)
17
18 # Fine-tuning del modelo SBERT
19 model_name = 'hiiamsid/sentence_similarity_spanish_es'
20 word_embedding_model = models.Transformer(model_name)
21 pooling_model = models.Pooling(word_embedding_model.
      get_word_embedding_dimension())
22 model = SentenceTransformer(modules=[word_embedding_model, pooling_model])
23
24 # Crear datos de entrenamiento
25 train_examples = [InputExample(texts=[row['DESCRIPCION_CLIENTE'], row['
      DESCRIPCION_BMC']], label=1.0) for _, row in db.iterrows()]
26
27 # DataLoader para agrupar nuestros datos
28 train_dataloader = DataLoader(train_examples, shuffle=True, batch_size=16)
29 train_loss = losses.CosineSimilarityLoss(model)
30
31 # Fine-tune el modelo
32 num_epochs = 3
```

```
start_time = time.time()
model.fit(train_objectives=[(train_dataloader, train_loss)], epochs=
    num_epochs, warmup_steps=100)
end_time = time.time()

# Calcular tiempo de entrenamiento
training_time = end_time - start_time
print(f"Tiempo de entrenamiento: {training_time/60:.2f} minutos")

# Guardar el modelo
model.save('/content/drive/MyDrive/Tesis/fine_tuned_sbert_model_1')

# Cargar el modelo fine-tuned
model = SentenceTransformer('/content/drive/MyDrive/Tesis/
    fine_tuned_sbert_model_1')

# Descripciones BMC  nicas  y sus embeddings
unique_bmc_descriptions = db['DESCRIPCION_BMC'].unique()
unique_bmc_embeddings = model.encode(unique_bmc_descriptions,
    convert_to_tensor=True)

# Calcular embeddings para descripciones de clientes
client_embeddings = model.encode(db['DESCRIPCION_CLIENTE'].tolist(),
    convert_to_tensor=True)

# Calcular similitudes de coseno solo para descripciones BMC  nicas
cosine_similarities = util.cos_sim(client_embeddings,
    unique_bmc_embeddings)

# Crear un DataFrame de similitudes de coseno con descripciones BMC
    nicas  como columnas
similarity_df = pd.DataFrame(cosine_similarities.numpy(), columns=
    unique_bmc_descriptions, index=db['DESCRIPCION_CLIENTE'].tolist())

# Encontrar la mejor coincidencia para cada descripci n de cliente
top_match = similarity_df.idxmax(axis=1)

# Asumir que 'DESCRIPCION_BMC' en tu DataFrame db contiene las
    descripciones BMC reales para cada cliente
actual_bmc_descriptions = db['DESCRIPCION_BMC'].tolist()

def check_actual_is_top(actual, top):
    return actual == top

results = [check_actual_is_top(actual_bmc_descriptions[i], top_match.iloc[
    i]) for i in range(len(db))]

db['Actual_is_Top'] = results

# Calcular la precisi n general
accuracy = sum(results) / len(results) * 100
print(f"Precisi n: {accuracy:.2f}%")

# Encontrar las 3 mejores coincidencias para cada descripci n de cliente
top_matches = similarity_df.apply(lambda x: list(x.nlargest(3).index),
    axis=1)

def check_actual_in_top(actual, top):
    return actual in top

results = [check_actual_in_top(actual_bmc_descriptions[i], top_matches.
    iloc[i]) for i in range(len(db))]

db['Actual_in_Top3'] = results

# Calcular la precisi n general
```

```
accuracy = sum(results) / len(results) * 100
print(f"Precisi n: {accuracy:.2f}%")
```

Listing 11: Entrenamiento y Evaluación de SBERT Fine-tunned

G.1. Función que añade ejemplos de entrenamiento negativos

```
    # Create training data
train_examples = []
for _, row in db.iterrows():
    # Positive example (correct pair)
    train_examples.append(InputExample(texts=[row['DESCRIPCION_CLIENTE'],
        row['DESCRIPCION_BMC']], label=1.0))
    # Negative example (incorrect pair)
    random_bmc = db['DESCRIPCION_BMC'].sample().values[0]
    train_examples.append(InputExample(texts=[row['DESCRIPCION_CLIENTE'],
        random_bmc], label=0.0))
```

H. Resumen y Comparación de resultados general

Correct Matches	**Descripciones completas**		**Subyacentes**	
Accuracy Top	**k = 1**	**k = 3**	**k = 1**	**k = 3**
W2V (Tokens) iter1	0.3 %	0.82 %	0.41 %	1.08 %
W2V (Tokens) iter2	0.23 %	0.61 %	x	x
W2V (Tokens) iter3	0.30 %	0.80 %	x	x
W2V (Lemas)	0.47 %	0.75 %	x	x
W2V (N-grams)	0.058 %	x	x	x
spaCy	2.38 %	4.40 %	3.59 %	8.39 %
Jaccard	12.78 %	21.12 %	27.23 %	35.85 %
spaCy (J+CS)	x	x	11.71 %	27.41 %
SBERT	9.82 %	18.25 %	15.44 %	26.35 %
SBERT (J+CS)	x	x	23.52 %	36.93 %
SBERT (Fine-tuned) iter 1	9.77 %	17.76 %	3.75 %	6.36 %
SBERT (Fine-tuned) iter 2	34.84 %	59.52 %	47.14 %	69.71 %
SBERT (Fine-tuned) iter 3	33.91 %	58.81 %	47.66 %	68.60 %

Cuadro 1: Accuracy Comparison