Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Asociación de direcciones con Procesamiento de Lenguaje Natural y Apache Spark

Address matching with Natural Language Processing and Apache Spark

Enrique Álvarez Mesa

La Laguna, 23 de mayo de 2024

Dña. **Isabel Sánchez Berriel**, con N.I.F. 42.885.838-S profesora Contratada Doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

Dña. **Luz Marina Moreno de Antonio**, con N.I.F. 45.457.492-Q profesora Contratada Doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutora

C E R T I F I C A (N)

Que la presente memoria titulada:

“Asociación de direcciones con Procesamiento de Lenguaje Natural y Apache Spark”

ha sido realizada bajo su dirección por D. **Enrique Álvarez Mesa**,

con N.I.F. 43.836.362-H.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 23 de mayo de 2024

Agradecimientos

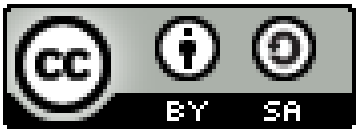
A mi familia por educarme y apoyarme para llegar a donde estoy hoy.

A mi pareja por ser un pilar fundamental en mi vida e impulsarme a ser mejor.

A mis amigos por estar siempre a mi lado.

A Isabel y Luz Marina por haberme guiado en todo este camino.

Licencia



Resumen

El presente trabajo aborda el desafío de asociar direcciones postales utilizando técnicas de procesamiento de lenguaje natural y aprendizaje automático, con un enfoque especial en el uso de Apache Spark para manejar grandes volúmenes de datos. Se emplea una base de datos proporcionada por el Instituto Canario de Estadística (ISTAC), la cual contiene más de dos millones de registros con información detallada sobre direcciones en la Comunidad Autónoma de Canarias.

El proyecto se divide en varias etapas fundamentales. En primer lugar, se realiza un exhaustivo proceso de limpieza y preprocesamiento de los datos para garantizar su calidad y coherencia. Esto incluye la gestión de valores nulos, la extracción del tipo de vía y la generación de nuevos datos para aumentar la frecuencia de ciertas direcciones.

Del mismo modo, se lleva a cabo el estudio de Apache Spark y su API de Python, PySpark, y la configuración del entorno para optimizar la asignación de memoria y el procesamiento distribuido. Se implementan modelos preentrenados para calcular representaciones vectoriales de las direcciones y buscar similitudes semánticas, además de diferentes enfoques, como la integración de Chroma DB para el almacenamiento y consulta eficiente de las representaciones vectoriales.

A lo largo del proyecto, se enfrentan desafíos significativos relacionados con la gestión de recursos y la eficiencia computacional. Se experimenta con diferentes configuraciones de Apache Spark para optimizar el rendimiento del sistema y se aborda la necesidad de manejar grandes volúmenes de datos sin superar las limitaciones de memoria.

Finalmente, se realiza un análisis exhaustivo de los resultados obtenidos, identificando áreas de mejora y posibles direcciones para futuras investigaciones

Palabras clave: asociación de direcciones, procesamiento de lenguaje natural, aprendizaje automático, Apache Spark, similitud semántica, representación vectorial.

Abstract

The present work addresses the challenge of address matching using natural language processing and machine learning techniques, with a special focus on using Apache Spark to handle large volumes of data. A database provided by the Canary Islands Institute of Statistics (ISTAC) is used, which contains more than two million records with detailed information about addresses in the Autonomous Community of the Canary Islands.

The project is divided into several fundamental stages. First, an exhaustive data cleaning and preprocessing process is carried out to ensure its quality and consistency. This includes managing null values, extracting the type of street, and generating new data to increase the frequency of certain addresses.

Subsequently, Apache Spark is initialized using its Python API, PySpark, configuring the environment to optimize memory allocation and distributed processing. Pretrained models are implemented to calculate embeddings of addresses and search for semantic similarities, in addition to various approaches such as integrating Chroma DB for efficient storage and querying of embeddings.

Throughout the project, significant challenges related to resource management and computational efficiency are faced. Different configurations of Apache Spark are experimented with to optimize system performance, and the need to handle large volumes of data without exceeding memory limitations is addressed.

Finally, a comprehensive analysis of the results obtained is performed, identifying areas for improvement and possible directions for future research.

Keywords: address matching, natural language processing, machine learning, Apache Spark, semantic similarity, embeddings.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1 Introducción

En este proyecto se busca aplicar al problema de asociación de direcciones postales (*Address matching*) un sistema de cómputo distribuido, como es *Apache Spark*. Con esto se pretende continuar con la línea de trabajo que inició en los proyectos *Machine learning and NLP approaches in address matching* (Syne L., 2022) y *Procesamiento de Lenguaje Natural en direcciones postales* (Cabrera J., 2023) sobre la asociación de direcciones utilizando técnicas de Procesamiento del Lenguaje Natural (PLN) y Aprendizaje Automático (AA) o Machine Learning (ML).

Se pretende estudiar la capacidad de cómputo y sus tiempos, así como la precisión técnicas como la similitud del coseno y el uso de diferentes representaciones textuales, utilizando modelos del lenguaje para resolver este problema. Para ello, se cuenta con una base de datos (BDD) proporcionada por el Instituto Canario de Estadística (*ISTAC*), con información sobre direcciones postales georreferenciadas de la Comunidad Autónoma de Canarias.

La BDD proporcionada tiene el potencial para poder aplicarse a este proyecto, donde se pretende que dada una dirección postal proporcionada por un ciudadano, identificar su equivalente en la BDD, incluso si su escritura difiere. Las novedades principales que se incorporan en este trabajo son, el uso del cómputo distribuido y el uso de distancias semánticas basadas en representaciones en forma de vectores numéricos para computar la semejanza con las direcciones georreferenciadas.

1.1 Estado del arte

La ley de Estadística de la Comunidad Autónoma de Canarias se creó el 28 de enero de 1991, estableciendo en su artículo 32 la construcción de un banco de datos administrativos para fines estadísticos. Este banco se nutre prioritariamente de los ficheros administrativos de la Comunidad Autónoma de Canarias, los cuales deben ser remitidos al Instituto Canario de Estadística (ISTAC) por todos los departamentos titulares de los mismos y necesarios para el ejercicio de la función estadística.

El propósito de este banco de datos es mejorar la eficiencia de la actividad estadística de interés de la Comunidad Autónoma de Canarias. Durante la ejecución del Plan Estadístico de Canarias 2018-2022, se impulsó el Sistema de Datos Integrados (iDatos) para su desarrollo. Este sistema tiene como objetivo permitir la producción de estadísticas multifuentes mediante la elaboración de datos maestros organizados en directorios y

registros que faciliten el enlace de fuentes diversas.

A partir de este banco de datos, el ISTAC llevó a cabo la georreferenciación utilizando cuatro tipos diferentes: determinística, pseudodeterminística, probabilística y aleatoria.

La georreferenciación determinística consistió en las relaciones por códigos de ciertos campos de la BDD y las coincidencias por dirección. Se realizó un proceso de record linkage, que comparó estos campos con otros registros ya georreferenciados y con los mismos códigos.

La pseudodeterminística se aplicó únicamente en la fuente del Padrón Municipal de Habitantes (PMH). Consistió en asignar una aproximación del portal, es decir, un portal cercano al real pero no exacto.

La georreferenciación probabilística utilizó las APIs de diversos proveedores (ArcGIS, Bing, Cartociudad, Google Map, Google Place, Here, Komoot). Se llamó a cada dirección con cada una de estas APIs, lo que resultó en varios puntos para una misma dirección. Se implementó un método de selección para elegir el punto más adecuado para cada dirección.

La georreferenciación aleatoria asignó un punto aleatorio a una dirección que no se pudo georreferenciar. Este punto podría estar en la misma vía, municipio, provincia o en Canarias. La selección del punto aleatorio se realizó mediante el paquete "pps" de R, que proporcionó candidatos cuyo valor probabilístico estaba en proporción al tamaño. Los candidatos variaron según la fuente de datos, por ejemplo, el número de personas en cada portal para PMH, la población activa para demandantes y llamamientos, y el número de cuentas coincidentes en los dos primeros dígitos de la CNAE[1] para cuentas de cotización. (González et al., & Hernández, 2021)

En (Comber & Arribas-Bel, 2019) se presenta una comparación práctica entre dos técnicas de aprendizaje automático, Word2Vec y Conditional Random Fields (CRFs), aplicadas a la asociación de direcciones.

La asociación de direcciones es crucial en la integración de datos espaciales, donde la falta de identificadores únicos dificulta la unión directa de registros. Tradicionalmente, se utilizan métodos basados en texto para resolver estos problemas, pero los avances en el aprendizaje automático han abierto nuevas posibilidades. Este estudio evalúa las ventajas y limitaciones de Word2Vec, que convierte palabras en vectores en un espacio continuo, y CRFs, que modelan las dependencias contextuales entre palabras.

Los resultados del estudio indican que ambas técnicas tienen fortalezas en diferentes aspectos de la asociación de direcciones. Word2Vec es útil para capturar similitudes semánticas y contextuales, mientras que CRFs son efectivos para modelar dependencias secuenciales entre elementos de las direcciones. La combinación de ambas metodologías podría mejorar significativamente la precisión del asociación de direcciones.

[1] Clasificación Nacional de Actividades Económicas

A partir de este trabajo, donde se pretende resolver la asociación de direcciones mediante aprendizaje automático, y junto al ISTAC al compartir su banco de datos, se creó una línea de trabajo donde aparecieron proyectos que ofrecieron nuevas técnicas para resolver este problema de la georreferenciación en Canarias.

El primero de ellos fue *Machine learning and NLP approaches in address matching* (Syne, 2022) donde se expuso la posibilidad de aplicar algoritmos de ML para resolver este problema de asociación de direcciones sobre un subconjunto muy reducido de la base de datos del ISTAC, en concreto 16.024 direcciones. Los modelos utilizados fueron *word2vec*, *fastText*, *doc2vec* y *BERT*, destacando que se entrenó una versión propia para *word2vec* y *fastText*, además de usar el modelo pre entrenado. Sin embargo, con *doc2vec* sólo se usó el modelo propio entrenado. Un resumen de los datos obtenidos se puede comprobar en la tabla 1.1:

Model	**Accuracy**	**Precision**	**Recall**
Trained wor2vec	0.675	0.023	0.93
Pre-trained wor2vec	0.789	0.034	0.92
Trained FastText	0.552	0.017	0.94
Pre-trained FastText	0.976	0.233	0.848
Trained doc2vec	0.996	0.694	0.848
BERT	0.997	0.80	0.848

Tabla 1.1: Resultados del trabajo de Syne L.

Por otro lado, en el trabajo *Procesamiento de Lenguaje Natural en información postal* (Cabera J, 2023), se experimenta con técnicas de Procesamiento de Lenguaje Natural, haciendo uso modelos de clasificación como *Random Forest*, *Naive Bayes Gaussiano* y *XGBoost* e incluso el algoritmo de agrupamiento *K-means* para resolver este problema asignando coincidencias binarias (*match* y *no match*) si las direcciones comparadas son o no equivalentes.

Según su estudio, no se hace uso del todo el banco de datos para tratar el problema, sino que lo acota a un municipio en concreto, debido a la cantidad de datos y a los limitados recursos hardware disponibles. Tras realizar diferentes pruebas, consigue obtener las siguientes precisiones haciendo uso de un sobre-muestreo aplicando SMOTE para obtener un balanceo ideal sobre las clases (*match* y *no match*):

- Random Forest Classifier: 99,3%, en un tiempo de ejecución de 2792 segundos.
- Gaussian NB: 94,7%, en un tiempo de ejecución de 375,3 segundos.
- XGBClassifier: 98,7%,en un tiempo de ejecución de 672,45 segundos.

Destacando además la mejor precisión obtenida con el algoritmo de clasificación

k-medias, haciendo uso de 2 grupos y 10 ejecuciones por centroide, fue de un 72,6% en un tiempo de ejecución de menos de un segundo.

El artículo "GeoRoBERTa: A Transformer-based Approach for Semantic Address Matching" (Guermazi et al., 2023) presenta un enfoque novedoso para la asociación semántica de direcciones utilizando la arquitectura Transformer, específicamente la variante RoBERTa. Este enfoque aprovecha dos tipos de conocimiento geográfico durante la fase de emparejamiento, lo que lo hace relevante para el problema de asociación de direcciones en Canarias.

En primer lugar, el artículo destaca el uso de RoBERTa, un modelo preentrenado de lenguaje basado en la arquitectura Transformer, que ha demostrado ser altamente efectivo para capturar la semántica y el contexto en datos textuales. Al entrenar RoBERTa para la adaptación (fine-tuning) a un conjunto de datos específico de direcciones, el modelo puede aprender representaciones vectoriales contextualizadas de direcciones que capturan su significado y similitud semántica.

Además, el artículo introduce dos tipos de conocimiento geográfico que se incorporan al enfoque propuesto: los etiquetados geográficos (GT) y la codificación de geohash (GH). Los etiquetados geográficos permiten al modelo capturar información adicional sobre la ubicación geográfica de las direcciones, mientras que la codificación de geohash proporciona una representación espacial de las direcciones que puede mejorar la detección de similitudes semánticas entre ellas.

En cuanto a la evaluación, el estudio presenta resultados prometedores en términos de precisión y rendimiento computacional. Por ejemplo, el enfoque GeoRoBERTa supera a otros métodos basados en Word2vec y fastText en términos de medida F en dos conjuntos de datos reales de direcciones en francés y senegalés. Además, se realiza un análisis de ablación para evaluar el impacto de cada tipo de conocimiento geográfico en el rendimiento del modelo, lo que proporciona información valiosa sobre cómo mejorar aún más el enfoque propuesto.

Finalmente, cabe destacar que el artículo “Deep Transfer Learning Model for Semantic Address Matching“ (Xu et al., 2022) propone un modelo de asociación semántica de direcciones (DSAMM) que se basa en dos componentes principales: el Modelo Semántico de Direcciones (ASM) y el Modelo de Aprendizaje de Transferencia Profunda. El ASM se desarrolla mediante el preentrenamiento de un modelo de lenguaje sobre un amplio corpus de direcciones, lo que le permite comprender la semántica subyacente de las direcciones, incluidos los patrones de caracteres y las relaciones entre ellos.

Posteriormente, el ASM se perfecciona aún más utilizando datos etiquetados y técnicas de aprendizaje profundo. El DSAMM utiliza el ASM como base y se refina mediante el aprendizaje de transferencia profunda, siendo entrenado con un conjunto de datos etiquetado y ajustándose para mejorar su precisión en la tarea de coincidencia de direcciones.

Los resultados experimentales muestran que el DSAMM supera a otros métodos existentes en términos de precisión, recuperación y puntuación F1. Además, el artículo identifica posibles mejoras para futuras investigaciones, como considerar la jerarquía de las direcciones y la introducción de información geográfica adicional.

El objetivo principal de este trabajo es desarrollar y evaluar un sistema para la asociación de direcciones geográficas, utilizando técnicas avanzadas de procesamiento de datos y aprendizaje automático. El proyecto se centra en preprocesar una extensa base de datos de direcciones proporcionada por el ISTAC, implementando modelos de embeddings para representar semánticamente las direcciones y facilitando la búsqueda y comparación de similitudes. El alcance del trabajo incluye la limpieza y normalización de datos, la integración de herramientas como Apache Spark para el manejo y consulta de grandes volúmenes de datos, y la realización de un análisis exhaustivo de resultados para validar la efectividad del sistema desarrollado.

Capítulo 2 Procesamiento de Lenguaje Natural

2.1 Definición

El procesamiento del lenguaje natural (PLN) es un campo de la inteligencia artificial (IA) que se centra en comprender el lenguaje humano mediante técnicas de lingüística computacional y aprendizaje automático. Un componente central del PLN es el concepto de "pipeline", que se refiere a una serie ordenada de pasos o etapas que se aplican secuencialmente al procesar una pieza de información. Cada etapa en el pipeline descompone una tarea compleja del PLN en pasos más manejables y específicos, lo que permite un procesamiento eficiente y sistemático del lenguaje.

Estas etapas pueden variar dependiendo de la tarea específica que se esté abordando, pero en general, las principales son:

- **Análisis morfológico o léxico**. Consiste en el análisis interno de las palabras que forman oraciones para extraer lemas, rasgos flexivos, unidades léxica compuestas. Es esencial para la información básica: categoría sintáctica y significado léxico.
- **Análisis sintáctico**. Consiste en el análisis de la estructura de las oraciones de acuerdo con el modelo gramatical empleado (lógico o estadístico).
- **Análisis semántico**. Proporciona la interpretación de las oraciones, una vez eliminadas las ambigüedades morfosintácticas.
- **Análisis pragmático**. Incorpora el análisis del contexto de uso a la interpretación final. Aquí se incluye el tratamiento del lenguaje figurado (metáfora e ironía) así como el conocimiento del mundo específico necesario para entender un texto especializado.
- **Tokenización**. Divide el texto en unidades más pequeñas, como palabras o caracteres, conocidas como "tokens". Esto facilita el análisis posterior del texto.
- **Normalización**. Consiste en estandarizar el texto de diversas formas, como la conversión a minúsculas, la eliminación de puntuación, la corrección ortográfica y la lematización (reducción de palabras a su forma base o lema).

El PLN se utiliza ampliamente en una variedad de aplicaciones, destacando las siguientes

tareas:

- **Clasificación de oraciones completas:** Determinar el sentimiento de una revisión, identificar si un correo electrónico es spam, evaluar la gramaticalidad de una oración o determinar la relación lógica entre dos oraciones.
- **Clasificación de cada palabra en una oración:** Identificar los componentes gramaticales de una oración (sustantivos, verbos, adjetivos) o reconocer entidades nombradas como personas, ubicaciones u organizaciones.
- **Generación de contenido de texto:** Completar un mensaje con texto generado automáticamente, rellenar los espacios en blanco en un texto utilizando palabras predichas.
- **Extracción de respuestas de un texto:** Obtener respuestas a preguntas específicas dentro de un contexto proporcionado.
- **Generación de nuevas oraciones a partir de un texto de entrada:** Traducir un texto a otro idioma, resumir la información contenida en un texto de manera concisa.
- **Extracción de información:** Identificar y extraer información específica de documentos de texto, como nombres de personas, fechas, ubicaciones o eventos.
- **Detección de idioma:** Determinar automáticamente el idioma en el que está escrito un texto.
- **Corrección gramatical y ortográfica:** Identificar y corregir errores gramaticales y ortográficos en un texto.
- **Análisis de redes sociales:** Analizar la actividad y el contenido en redes sociales para entender tendencias, opiniones o comportamientos de los usuarios.
- **Generación de resúmenes automáticos de documentos largos:** Resumir grandes volúmenes de texto en versiones más cortas y concisas, manteniendo la información más relevante.
- **Reconocimiento de voz en tiempo real:** Convertir el habla en texto de manera precisa y en tiempo real, como en aplicaciones de transcripción o asistentes virtuales. (DeepLearning.AI, 2023)

2.2 Similitud semántica

En este trabajo, el análisis semántico es crucial para comparar direcciones postales, ya que permite superar las variaciones de formato, abreviaciones y errores tipográficos, logrando una correspondencia más precisa y efectiva que los métodos basados solo en coincidencias textuales exactas.

La similitud semántica, en este contexto, es una medida de la relación entre dos unidades de información en términos de su significado. En el campo del PLN, se refiere a la

cuantificación de la semejanza en el significado entre palabras, frases, oraciones o documentos. Cuanto más similares sean estas piezas de información, menor será la distancia semántica entre ellas.

Por ejemplo, en la figura 2.1 se tiene pares de palabras, donde los colores más fuertes indican una mayor similitud.

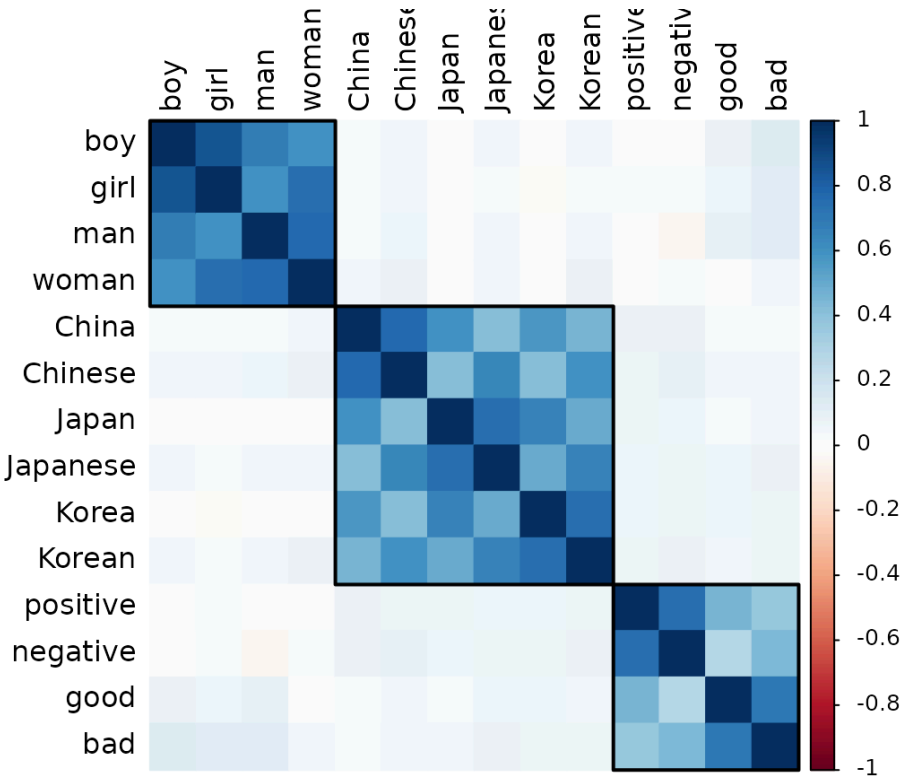


Figura 2.1: Ejemplo gráfico de la similitud del coseno usando pares de palabras
Fuente: Bao, s.f.

Existen diferentes formas de calcular la similitud entre dos piezas de información, pero a continuación se describen algunas de las más importantes.

2.2.1 Similitud del coseno

La similitud del coseno es una métrica usada para medir qué tan similares son dos documentos independientemente de su tamaño. Véase la figura 2.2:

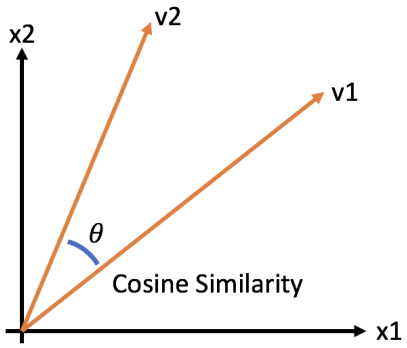


Figura 2.2: Representación de la similitud del coseno
Fuente: Datacamp, s.f.

Esta media es ventajosa debido a que incluso si dos documentos similares están alejados por la distancia Euclidiana (debido a sus tamaños) no afectará negativamente, debido a que se normalizan los vectores al dividir por su norma. Así pues, la fórmula se define como:

$$Cos(\theta) = \frac{A \cdot B}{||A|| \times ||B||}$$

El valor resultante de la similitud del coseno varía entre −1 y 1:

- Un valor de 1 indica que los vectores son idénticos en dirección, lo que implica máxima similitud semántica.
- Un valor de 0 indica que los vectores son ortogonales, lo que sugiere que no hay similitud semántica.
- Un valor de −1 indica que los vectores son opuestos en dirección, lo que implica máxima disimilitud semántica. (Miesle, 2023)

2.2.2 Distancia Manhattan

La distancia Manhattan o la distancia L1 es una métrica en la cual la distancia entre dos puntos es la suma de las diferencias absolutas de sus coordenadas cartesianas. La fórmula sería:

$$d(A,\ B) = \sum_{i=1}^{n} \left| x_i + \ y_i \right|$$

Es como si se sumara las distancias que recorremos en una cuadrícula, moviéndose solo hacia arriba/abajo o izquierda/derecha. Por ejemplo, en un plano con $p1(x1,\ y1)$ y $p2(x2,\ y2)$ se podría ilustrar cómo la figura 2.3:

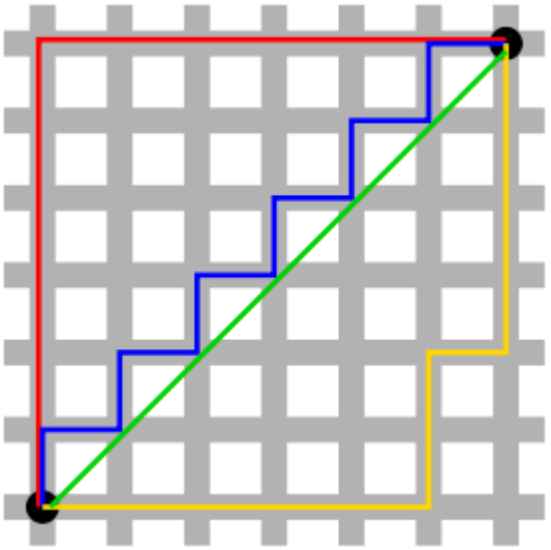


Figura 2.3: Representación de la distancia Manhattan
Fuente: Wikipedia, 2023

2.2.3 Distancia Euclidiana

La distancia Euclidiana mide la longitud del segmento que conecta dos puntos. Es la forma más obvia de representar la distancia entre dos puntos y para calcularla puede usarse el teorema de Pitágoras, de este modo su fórmula sería:

$$d = \sqrt{(x_2 - x_1)2 + (y_2 - y_1)2}$$

Se puede imaginar como si se usase una regla para medir la distancia. Véase la figura 2.4:

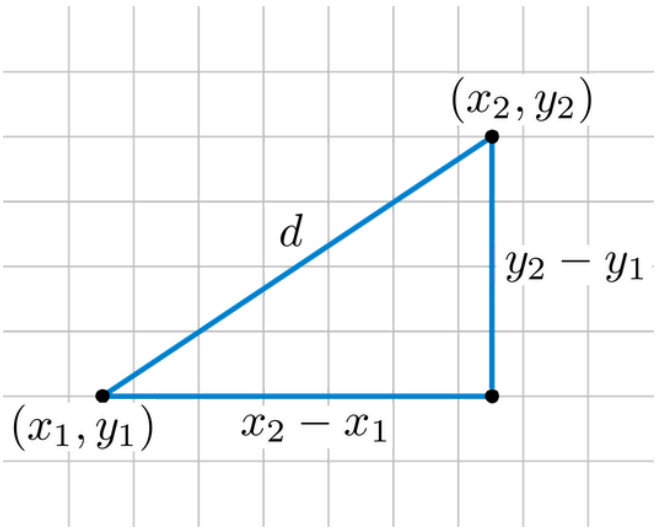


Figura 2.4: Representación de la distancia Euclidiana
Fuente: Gupta, 2019

2.2.4 Distancia de Minkowski

La distancia Minkowski es una generalización de las distancias Euclidiana y Manhattan. . Es un parámetro crucial en el análisis de datos y en diversas aplicaciones de aprendizaje automático y procesamiento de señales.

$$D(X, Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Al variar p, se puede ajustar la métrica para que sea más adecuada a las características del problema específico que se esté abordando. Como por ejemplo:

- Cuando p **= 1**, obtenemos la distancia de Manhattan.
- Cuando p **= 2**, obtenemos la distancia Euclidiana.
- Cuando p **=** ∞, obtenemos la distancia de Chebyshev. (Gupta, 2019)

Una representación de estas variaciones se puede encontrar en la figura 2.5:

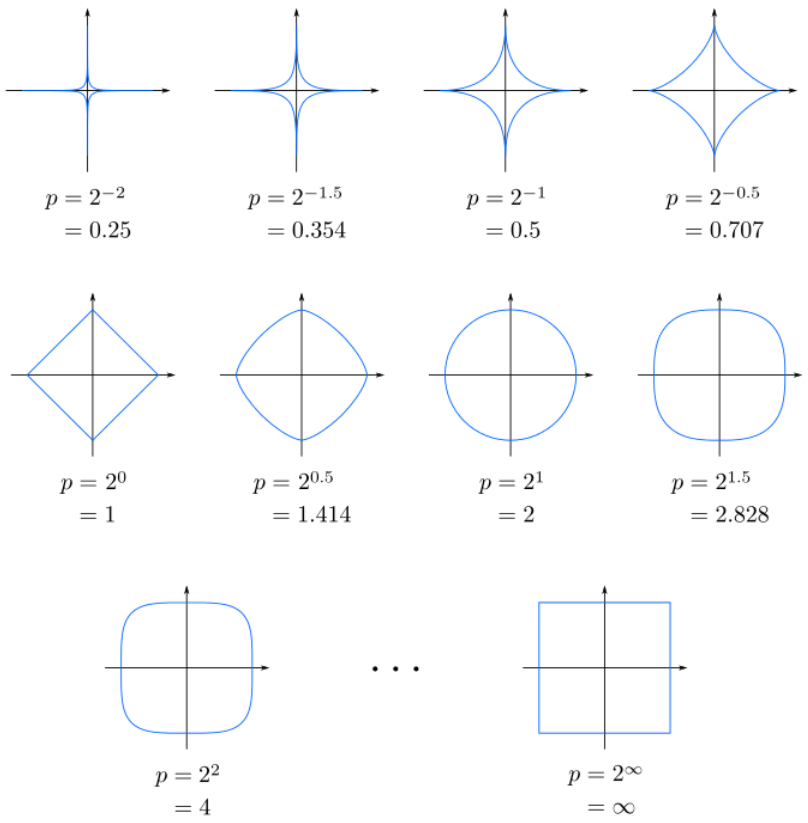


Figura 2.5: Representación de la distancia de Minkowski
Fuente: Chatterjee, s.f.

Como se pudo observar, para calcular la similitud semántica entre dos piezas de

información es necesario obtener una representación numérica de la información, por lo que este proceso se combina con las técnicas de representación vectorial de textos que se explican en el siguiente apartado.

2.3 Representación vectorial

La representación vectorial o embedding es una técnica fundamental en el procesamiento de datos, utilizada para representar información en un espacio vectorial, ya sea una palabra, un texto, una imagen o un audio. Esta técnica permite condensar la información relevante de manera que sea más fácil de procesar y analizar.

En la figura 2.6 se muestra el funcionamiento de un modelo de embedding, donde se observa cómo la información se transforma en vectores en un espacio n-dimensional.

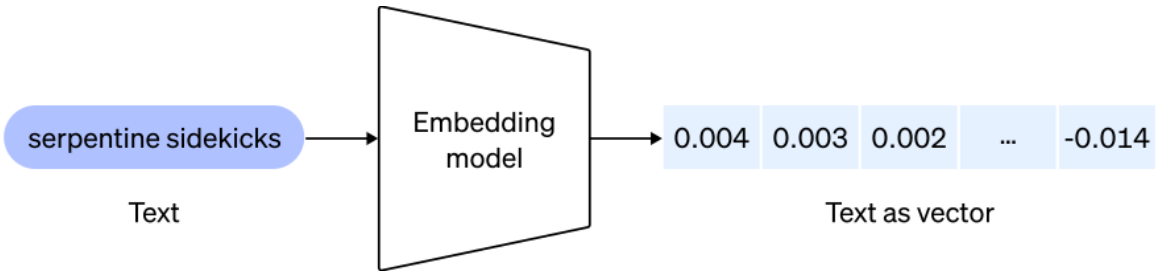


Figura 2.6: Funcionamiento de un modelo de embedding
Fuente: Espejel, s.f.

Ahora si, dados un par de representaciones vectoriales y uno de los tipos para calcular la similitud semántica anteriores, se puede cuantificar qué tan similares son dichas piezas de información.

Es importante señalar que el tamaño del vector de representación influye en la precisión de la representación en el espacio vectorial. Cuanto mayor sea el tamaño del vector, más precisa será la representación y más cerca estarán las piezas de información similares. Por lo tanto, la distancia entre dos vectores en el espacio vectorial es una medida de su relación.

En la figura 2.7 se presenta un ejemplo de word embedding en dos dimensiones, donde se puede apreciar visualmente cómo las palabras con significados similares están más cercanas en el espacio vectorial.

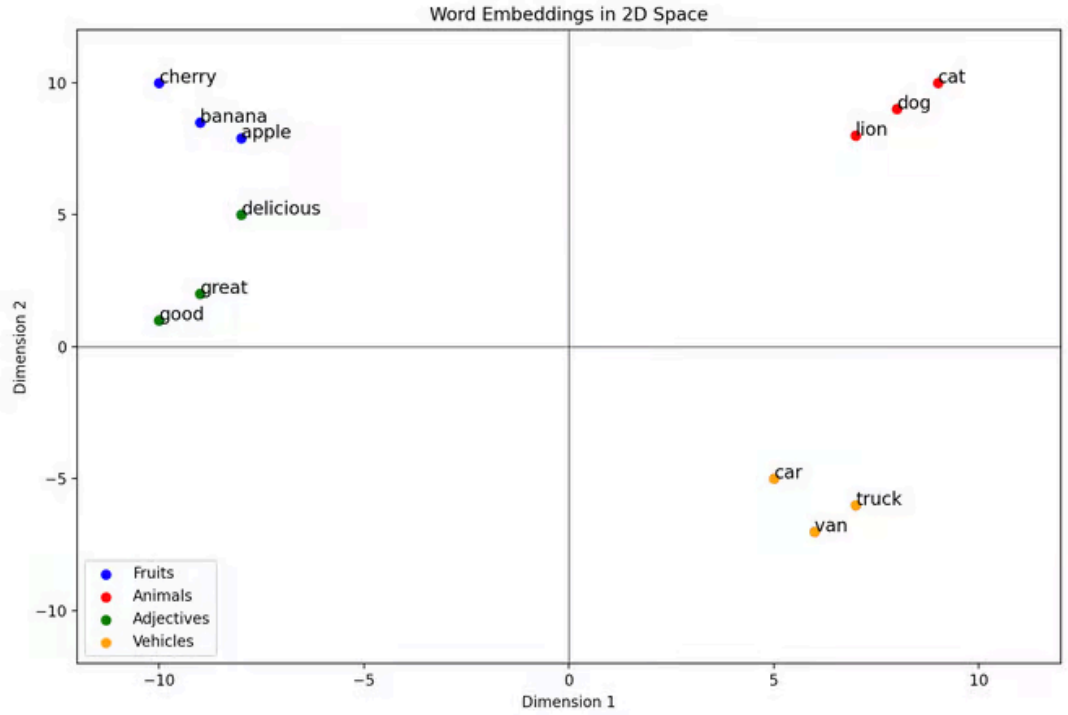


Figura 2.7: Ejemplo de word embedding en dos dimensiones
Fuente: Hoffman, 2023

En este trabajo se hará usó de esta técnica para generar embeddings de direcciones postales a partir de redes neuronales. Posteriormente, se calculará la similitud entre pares de direcciones usando la similitud del coseno.

2.3.1 TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) es una técnica de extracción de características que pondera la importancia de las palabras en un documento en relación con su aparición en un corpus. La puntuación TF-IDF de un término es el producto de TF y IDF. Se compone de dos componentes principales:

- **Frecuencia de Término (TF)**: Mide la importancia de una palabra dentro de un documento específico.

$$TF(palabra,\ documento) = \frac{Número\ de\ ocurrencias\ de\ la\ palabra\ en\ el\ documento}{Número\ total\ de\ palabras\ del\ documento}$$

- **Frecuencia Inversa de Documentos (IDF)**: Mide cuán importante es una palabra en todo el corpus.

$$IDF(palabra,\ corpus) = log(\frac{Número\ de\ documentos\ en\ el\ corpus}{Número\ de\ documentos\ que\ incluyen\ la\ palabra})$$

Una palabra es importante si aparece muchas veces en un documento. Sin embargo, esto genera un problema: palabras como "el" y "la" aparecen frecuentemente. Por lo tanto, su puntuación TF siempre será alta. Se resuelve este problema utilizando la IDF, que es alta

si la palabra es rara y baja si la palabra es común en todo el corpus.

TF-IDF se puede usar para obtener una representación vectorial de un documento, donde cada dimensión corresponde con una palabra. Un ejemplo ilustrativo lo podemos encontrar en la figura 2.8:

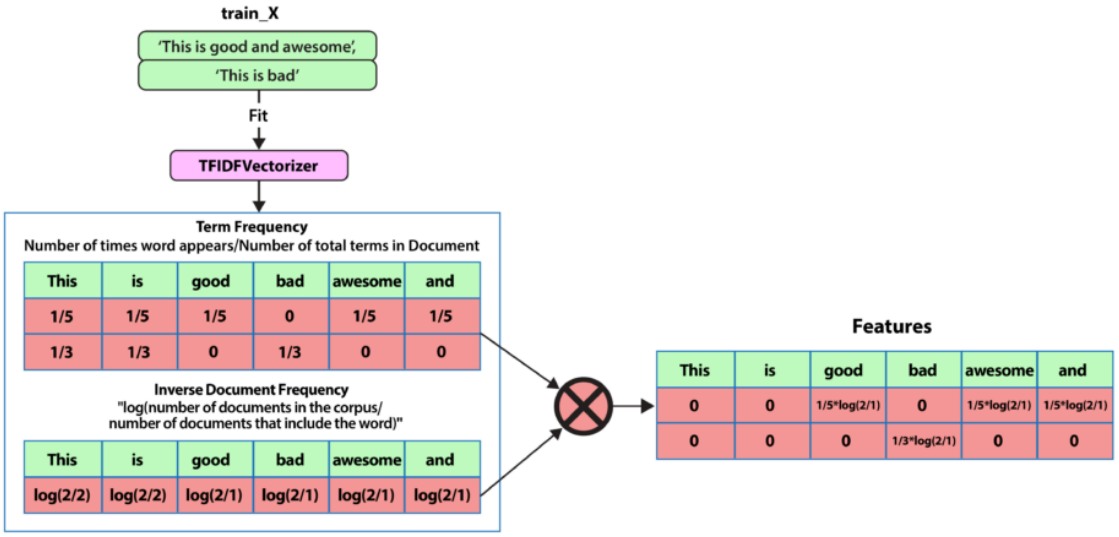


Figura 2.8: Ejemplo de una representación usando TF-IDF
Fuente: DeepLearning.AI, 2023

2.3.2 Word2vec

Word2vec es una familia de modelos que se utilizan para generar word embeddings. Utiliza redes neuronales poco profundas de dos capas. Puede emplear dos arquitecturas principales: el modelo continuo de bolsa de palabras (CBOW) y el modelo continuo skip-gram.

1. **Modelo Continuous Bag-of-Words (CBOW)**: Predice una palabra dada su contexto circundante sin importar el orden de las palabras en dicho contexto. Es más rápido y eficiente, especialmente para palabras frecuentes.

   Ejemplo:

   Supongamos que tenemos la oración: "El gato duerme en el sofá". Si elegimos una ventana de contexto de 2 palabras, el objetivo es predecir una palabra objetivo dada las palabras de contexto circundantes.

   Para la palabra objetivo "duerme", el contexto circundante son las palabras "El", "gato", "en" y "el".

   - **Contexto:** ["El", "gato", "en", "el"]
   - **Palabra objetivo:** "duerme"

   El modelo CBOW intentará aprender a predecir "duerme" utilizando las palabras de su contexto.

2. **Modelo Continuous Skip-Gram**: Predice las palabras contextuales a partir de una palabra objetivo, considerando cada palabra de contexto de manera independiente. Este modelo es más eficaz para palabras infrecuentes.

   Ejemplo:

   Usando la misma oración: "El gato duerme en el sofá", y una ventana de contexto de 2 palabras.

   Para la palabra objetivo "duerme", el contexto circundante son las palabras "El", "gato", "en" y "el".

   - **Palabra objetivo:** "duerme"
   - **Contexto:** ["El", "gato", "en", "el"]

   El modelo Skip-gram intentará aprender a predecir las palabras "El", "gato", "en" y "el" utilizando la palabra objetivo "duerme".

Se puede encontrar una representación de estas dos arquitecturas en la figura 2.9:

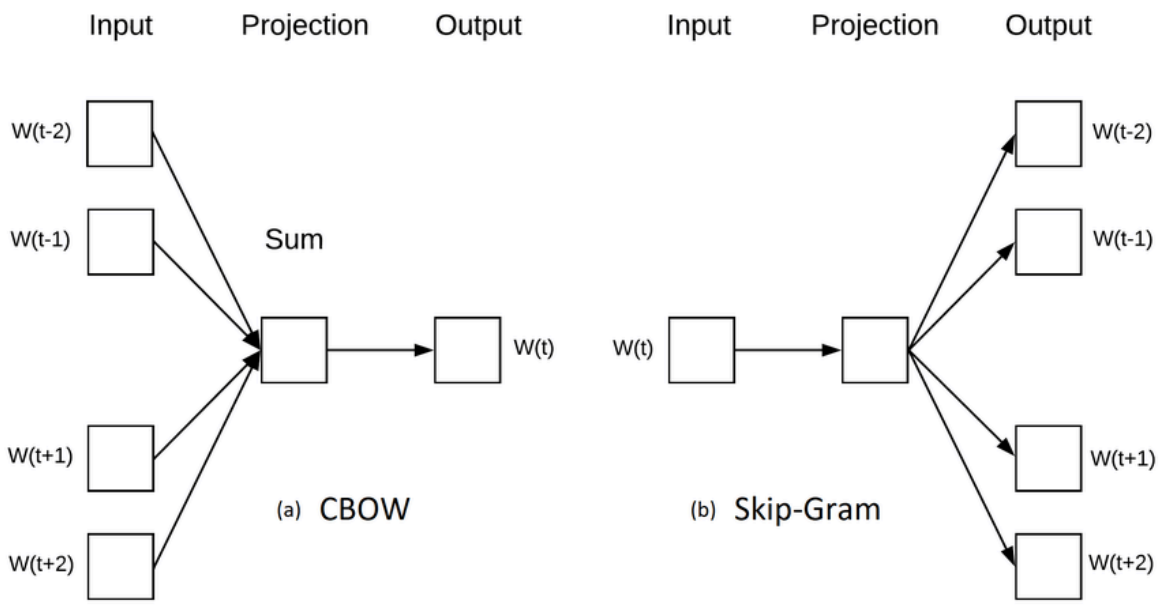


Figura 2.9: Arquitecturas Continuous Bag-of-Words y Skip-Grams
Fuente: Cano, 2018

Los resultados del entrenamiento pueden ser sensibles a la parametrización. Los siguientes son algunos parámetros importantes en el entrenamiento de Word2vec:

- **Algoritmo de entrenamiento**: Word2vec puede ser entrenado utilizando **softmax jerárquico**, que es eficaz para palabras infrecuentes, o **muestreo negativo**, que funciona mejor para palabras frecuentes y con vectores de baja dimensionalidad.
- **Submuestreo**: Las palabras de alta frecuencia pueden ser submuestreadas para acelerar el entrenamiento, ya que a menudo proporcionan poca información útil.
- **Dimensionalidad**: La calidad de los embeddings aumenta con la dimensionalidad,

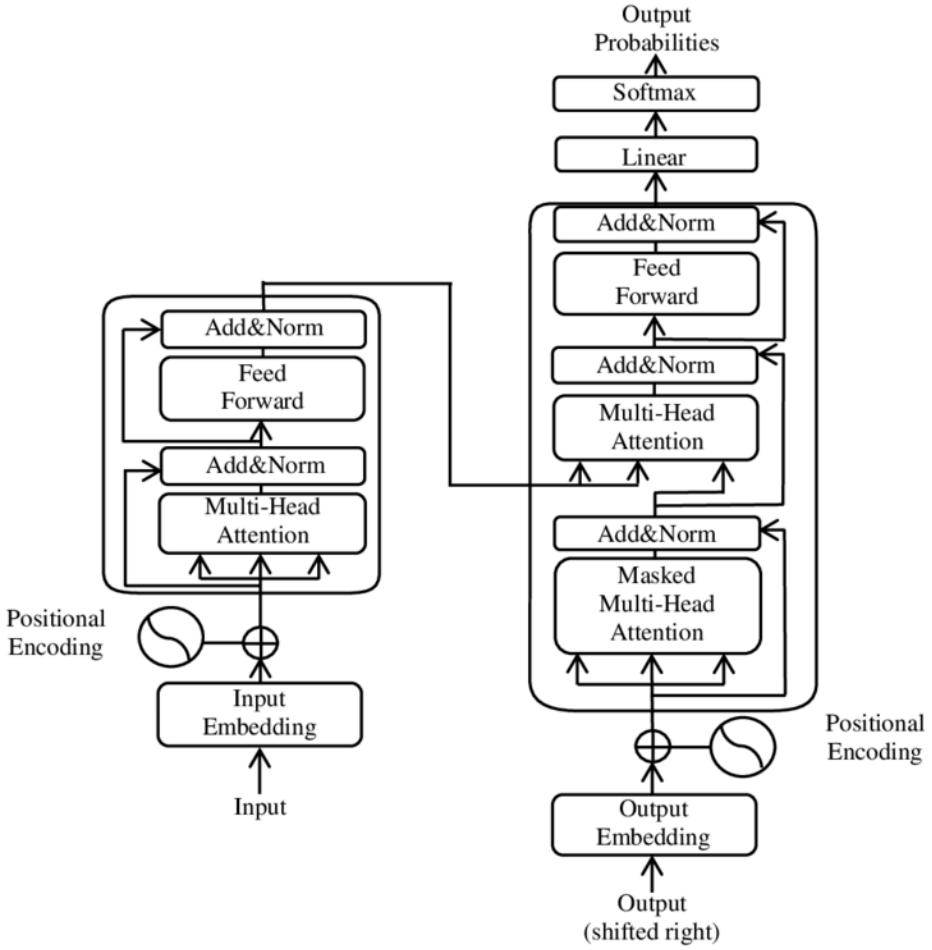
generalmente establecida entre 100 y 1,000 dimensiones.

- **Ventana de contexto**: Determina cuántas palabras antes y después de una palabra se consideran en su contexto. Se recomienda una ventana de 10 palabras para *skip-gram* y de 5 palabras para CBOW.

2.3.3 Transformers

Los Transformers fueron introducidos en 2017 a través del paper "Attention is All You Need"[2] por Vaswani et al. Esta arquitectura marcó una revolución en el procesamiento del lenguaje natural (NLP), superando a las redes recurrentes LSTM (Long Short-Term Memory) en tareas de traducción automática. La principal innovación fue el uso del mecanismo de atención que permite enfocarse en diferentes partes de la secuencia de entrada de manera paralela, en lugar de procesar secuencias de forma secuencial como las LSTM.

La arquitectura de los Transformers es fundamentalmente un modelo de aprendizaje profundo que se compone de múltiples capas de atención y transformación, como se muestra en la figura 2.10:



[2] https://arxiv.org/pdf/1706.03762

Figura 2.10: Arquitectura de los Transformers
Fuente: Bagnato, 2022

Según esta arquitectura, la secuencia de pasos a realizar sería:

1. **Entrada de Datos:**

   Secuencia de entrada: La arquitectura de Transformer toma una secuencia de datos como entrada. Esto puede ser texto en NLP, imágenes en tareas de visión por computador, o cualquier otro tipo de secuencia de datos.

   Embeddings y Positional Encoding: Cada elemento de la secuencia se representa inicialmente como un embedding que capturan su significado semántico. Además, se añade información sobre la posición de cada elemento en la secuencia mediante el positional encoding.

2. **Encoder:**

   Self-Attention: En cada capa del encoder, el modelo realiza self-attention para calcular la importancia relativa de cada elemento en la secuencia con respecto a todos los demás elementos. Esto permite al modelo aprender relaciones complejas entre los elementos de la secuencia.

   Operaciones Lineales y No Lineales: Después de la atención, se aplican operaciones lineales y funciones de activación no lineales para transformar los datos. Esto ayuda al modelo a capturar representaciones más complejas y abstractas de la secuencia.

   Feedforward Network: Finalmente, los datos se pasan a una red neuronal feedforward que aplica transformaciones adicionales. Esta red feedforward generalmente consiste en dos capas densas separadas por una función de activación no lineal, como la función ReLU (Rectified Linear Unit).

   Skip Connections y Layer Normalization: Para facilitar el entrenamiento de redes profundas, se utilizan skip connections que permiten que la información fluya directamente de una capa a otra sin transformaciones adicionales. Además, se aplica layer normalization para estabilizar la distribución de activaciones en cada capa y mejorar la eficiencia del modelo.

3. **Decoder:**

   Cross-Attention: En el decoder, el modelo realiza cross-attention, donde utiliza la información del encoder para generar una secuencia de salida. Esto permite al modelo incorporar información contextual de la secuencia de entrada al generar la secuencia de salida.

   Masked Self-Attention: Durante el entrenamiento, se utiliza masked self-attention para simular la generación secuencial de la secuencia de salida. Esto evita que el modelo vea información futura durante el entrenamiento y garantiza que solo pueda

generar una palabra a la vez en la secuencia de salida.

Operaciones Lineales y No Lineales: Al igual que en el encoder, se aplican operaciones lineales y funciones de activación no lineales para transformar los datos en el decoder.

4. **Salida:**

   Capa de Salida: La salida final del modelo pasa por una capa lineal seguida de una función softmax. El softmax convierte los puntajes en probabilidades, asignando una probabilidad a cada posible elemento de salida.

   Selección de Elementos: Finalmente, se selecciona el elemento con la mayor probabilidad como la predicción final del modelo. En el caso del NLP, esto podría ser la siguiente palabra en una secuencia de texto, mientras que en la visión por computador podría ser la clase de objeto más probable en una imagen. (Bagnato, 2022)

A modo resumen, se puede decir que la arquitectura de los Transformers se basa en un modelo encoder-decoder, donde:

- **Encoder**: Toma la secuencia de entrada y la transforma en una representación interna que encapsula la información relevante.
- **Decoder**: Utiliza esta representación para generar la secuencia de salida, como la traducción de un texto.

Y las principales características de los transformers son:

- **Mecanismo de Auto-Atención**: Procesa todas las palabras de una vez en lugar de una por una, lo que acelera la velocidad de entrenamiento y reduce el costo de inferencia en comparación con las redes neuronales recurrentes (RNNs).
- **Paralelizable**: La capacidad de procesar múltiples palabras simultáneamente permite un entrenamiento más rápido y eficiente.

Entre los modelos más relevantes creados a partir de esta arquitectura están:

- **BERT (Bidirectional Encoder Representations from Transformers)**: Desarrollado por Google, BERT es un modelo de lenguaje bidireccional pre-entrenado que ha establecido nuevos estándares en una variedad de tareas de procesamiento del lenguaje natural (NLP). BERT utiliza una arquitectura de Transformer y ha demostrado un rendimiento sobresaliente en tareas como la comprensión de lectura, la traducción automática y el análisis de sentimientos.
- **GPT (Generative Pre-trained Transformer)**: La serie GPT, desarrollada por OpenAI, consta de varios modelos (GPT-1, GPT-2, GPT-3 y GPT-4) que son conocidos por su capacidad para generar texto coherente y de alta calidad. Estos modelos utilizan una arquitectura de Transformer entrenada en grandes corpus de texto para generar texto de manera autónoma.

- **T5 (Text-To-Text Transfer Transformer)**: T5 es un modelo de lenguaje desarrollado por Google que sigue el paradigma "text-to-text", donde todas las tareas de procesamiento del lenguaje natural se formulan como problemas de generación de texto. T5 ha demostrado un rendimiento sólido en una variedad de tareas de NLP, desde la traducción automática hasta la generación de resúmenes.
- **RoBERTa (Robustly optimized BERT approach)**: RoBERTa, desarrollado por Facebook, es una versión optimizada de BERT que utiliza técnicas de pre-entrenamiento más avanzadas y un conjunto de datos de entrenamiento más grande. RoBERTa ha demostrado un rendimiento superior a BERT en una variedad de tareas de NLP, incluida la comprensión de lectura y la clasificación de texto.

Capítulo 3 Tecnologías

En este apartado se describen las principales tecnologías que fueron utilizadas o valoradas para el desarrollo de este proyecto.

3.1 IAAS

Este trabajo fue desarrollado en una máquina virtual proporcionada por el servicio IaaS (Infraestructura como servicio) de la Universidad de La Laguna, la cual cuenta con *Ubuntu 22.04.3 LTS* como sistema operativo y con las especificaciones hardware que se muestran en la tabla 3.1:

Hardware	**Características**
CPU	Intel Xeon E312XX - 8 sockets virtuales - 1 núcleo por socket - 1 hilo por núcleo
Memoria RAM	32GB
Almacenamiento	100GB

Tabla 3.1: Características de la máquina virtual

3.2 Apache Spark

Apache Spark es un framework multilingüe de cómputo distribuido que permite realizar ciencia de datos, ingeniería de datos y aprendizaje automático en máquinas de un solo nodo o en clústeres. Entre los lenguajes que soporta están Python, Scala, SQL, R y Java. Como prerrequisito, Spark necesita que se tenga instalado en el sistema los lenguajes:

- **Java**: Spark está construido para ejecutarse en la JVM. La JVM proporciona una plataforma independiente del hardware que permite que el código Java, Scala y otros lenguajes compatibles con la JVM se ejecuten en cualquier sistema que tenga la JVM instalada

- **Scala**: Spark está escrito principalmente en este lenguaje. Scala es un lenguaje de programación que combina la programación funcional y orientada a objetos, lo que facilita la escritura de código conciso y eficiente.

3.2.1 Arquitectura

Apache Spark sigue una arquitectura maestro/esclavo con un administrador de clúster (Cluster Manager), distinguiendo así tres componentes principales:

- **Controlador del programa (Driver program)**
- **Administrador de clústeres (Cluster manager)**
- **Nodo trabajador (Worker node)**

En un clúster de Apache Spark, el programa principal de la aplicación Spark (que generalmente contiene el método main en Java) se conoce como **controlador de programa** o driver program. Este controlador del programa contiene un objeto de SparkContext, que se puede configurar con información como la memoria de los ejecutores, el número de ejecutores, etc. El **administrador de clústeres** se encarga de gestionar los recursos disponibles (nodos) en el clúster.

Cuando se crea el objeto SparkContext, se conecta al administrador de clústeres para negociar la asignación de ejecutores. El administrador del clúster asigna algunos o todos los ejecutores disponibles en los nodos al SparkContext según la demanda. Es importante tener en cuenta que múltiples aplicaciones Spark pueden ejecutarse en un solo clúster. (TutorialKart, s.f.)

En la figura 3.1 se puede encontrar una representación gráfica de esta arquitectura:

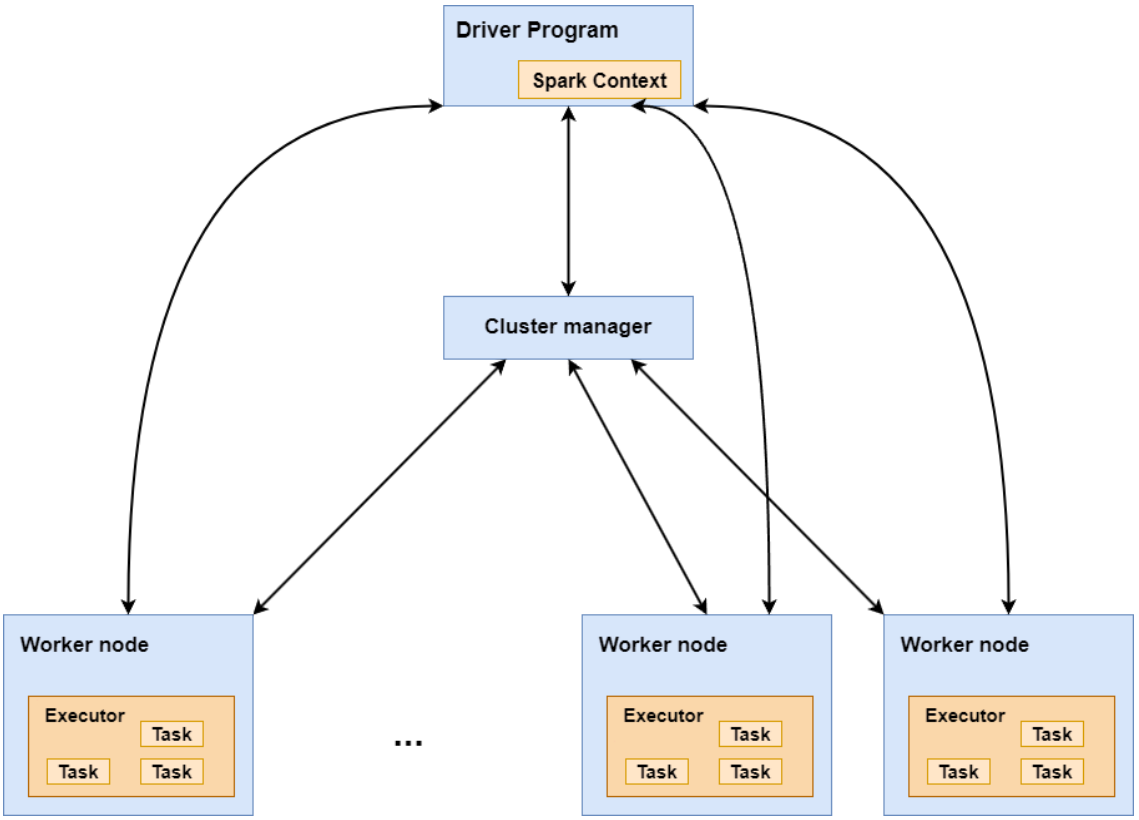


Figura 3.1: Arquitectura de Apache Spark
Fuente: Elaboración propia

Además, hay varias cosas útiles a tener en cuenta sobre esta arquitectura:

1. Cada aplicación obtiene sus propios procesos de ejecutor, que permanecen activos durante toda la duración de la aplicación y ejecutan tareas en varios hilos. Esto tiene el beneficio de aislar las aplicaciones entre sí, tanto en el lado de la programación de tareas (cada controlador programa sus propias tareas) como en el lado del ejecutor (las tareas de diferentes aplicaciones se ejecutan en JVM diferentes). Sin embargo, esto también significa que los datos no se pueden compartir entre diferentes aplicaciones Spark (instancias de SparkContext) sin escribirlos en un sistema de almacenamiento externo.
2. Spark es independiente del administrador de clúster subyacente. Mientras pueda adquirir procesos de ejecutor y estos se comuniquen entre sí, es relativamente fácil ejecutarlo incluso en un administrador de clúster que también admita otras aplicaciones (por ejemplo, Mesos/YARN/Kubernetes).
3. El controlador de programa debe escuchar y aceptar conexiones entrantes de sus ejecutores durante toda su vida útil. Como tal, el programa controlador debe ser accesible en red desde los nodos del trabajador.
4. Dado que el programa controlador programa tareas en el clúster, debe ejecutarse cerca de los nodos del trabajador, preferiblemente en la misma red de área local. Si deseas enviar solicitudes al clúster de forma remota, es mejor abrir un RPC al

controlador y hacer que este envíe operaciones desde un lugar cercano que ejecutar un controlador lejos de los nodos del trabajador. (Apache Spark, Cluster Mode Overview - Spark 3.5.1 Documentation, s.f.)

3.2.2 Configuración

Spark proporciona tres maneras de configurar el sistema:

- **Propiedades de Spark**: controla la mayoría de parámetros y puede ser especificado en tiempo de ejecución haciendo uso del objeto `SparkConf` o a través del sistema de propiedades de Java.
- **Variables de entorno**: puede ser usado para establecer los ajustes por máquina, como la dirección IP, a través del archivo `conf/spark-env.sh` en cada nodo.
- **Inicio de sesión**: puede ser configurado a través de `log4j2.properties`.

Como se verá en el apartado de Implementación, en este proyecto se hace uso de una combinación usando las propiedades de Spark y las variables de entorno.

Propiedades de Spark

Las propiedades de aplicación en Spark son configuraciones clave que afectan el comportamiento y el rendimiento de las aplicaciones. En la tabla 3.2 se muestran algunas de las propiedades más importantes y su significado:

Configuración	Definición
spark.app.name	Define el nombre de la aplicación Spark, que aparecerá en la interfaz de usuario y en los datos de registro
spark.driver.cores	Especifica el número de núcleos a utilizar para el proceso del controlador, solo en modo de clúster
spark.driver.maxResultSize	Limita el tamaño total de los resultados serializados de todas las particiones para cada acción de Spark, como collect, en bytes
spark.driver.memory	Define la cantidad de memoria a utilizar para el proceso del controlador, donde se inicializa SparkContext
spark.executor.memory	Indica la cantidad de memoria a utilizar por

	proceso de executor
spark.local.dir	Especifica el directorio para el espacio de "temporales" en Spark, incluyendo archivos de salida de mapas y RDDs almacenados en disco
spark.master	Define el administrador de clúster al que conectarse
spark.submit.deployMode	Especifica el modo de implementación del programa del controlador de Spark, ya sea "client" o "cluster"

Tabla 3.2: Propiedades de configuración de Spark

Variables de entorno

En el ecosistema de Apache Spark, la configuración adecuada juega un papel fundamental en el rendimiento y la eficiencia de las aplicaciones. Una parte crucial de esta configuración se realiza a través de variables de entorno, que permiten ajustar diversos aspectos del comportamiento de Spark. Veamos en detalle cómo se utilizan y configuran estas variables.

1. **Origen de las Variables de Entorno:**

   Las variables de entorno se leen desde el archivo `conf/spark-env.sh` en el directorio de instalación de Spark. En entornos Windows, este archivo se llama `conf/spark-env.cmd`.

   Es importante destacar que, por defecto, el archivo `conf/spark-env.sh` no existe. Para crearlo, se puede copiar `conf/spark-env.sh.template` y asegurarse de que sea ejecutable.

2. **Utilidad en Modos de Ejecución:**

   En modos Standalone y Mesos, `spark-env.sh` puede contener información específica de la máquina, como nombres de host.

   Además, este archivo se utiliza al ejecutar aplicaciones locales de Spark o scripts de envío, lo que lo convierte en una herramienta versátil para la configuración en diferentes contextos.

3. **Variables Clave:**

   En la tabla 3.3 se pueden encontrar, de manera resumida, las variables de configuración más importantes:

Variable de entorno	Descripción
JAVA_HOME	Localización donde Java está instalado (si no está en el PATH por defecto)
PYSPARK_PYTHON	Ejecutable binario de Python que se usará para PySpark tanto en el controlador como en los nodos de trabajo.
PYSPARK_DRIVER_PYTHON	Ejecutable binario de Python que se usará solo para PySpark en el controlador
SPARKR_DRIVER_R	Especifica el ejecutable de R a utilizar en la shell de SparkR
SPARK_LOCAL_IP	Dirección IP de la máquina a la que se va a enlazar
SPARK_PUBLIC_DNS	Nombre de host que el programa Spark anunciará a otras máquinas

Tabla 3.3: Variables de entorno para la configuración de Spark

4. **Configuración Avanzada:**

   Además de estas variables, `spark-env.sh` también permite configurar aspectos avanzados, como el número de núcleos por máquina y la memoria máxima en scripts de clúster independientes de Spark.

   Dado que `spark-env.sh` es un script de shell, algunas variables se pueden establecer programáticamente, lo que brinda flexibilidad adicional en la configuración.

5. **Consideraciones Especiales para YARN:**

   Es importante tener en cuenta que al ejecutar Spark en modo de clúster sobre YARN, las variables de entorno deben establecerse de manera diferente.
   En lugar de `spark-env.sh`, las variables se definen utilizando la propiedad `spark.yarn.appMasterEnv`. La propiedad `[NombreVariableDeEntorno]` en el archivo *`conf/spark-defaults.conf`*. Esto garantiza que las configuraciones se propaguen correctamente al proceso Application Master de YARN.

Inicio de sesión

Spark usa log4j para iniciar sesión cuando se use el modo clúster, de modo una guía para su configuración sería:

1. **Ubicación y Plantilla**: Se puede configurar el registro de eventos agregando un archivo `log4j2.properties` en el directorio *conf* de la instalación de Spark. Es recomendable empezar copiando el archivo existente `log4j2.properties.template` ubicado allí.
2. **Registro MDC por defecto**: Por defecto, Spark añade un registro al MDC (Contexto de Diagnóstico Mapeado) llamado `mdc.taskName`, que normalmente muestra algo como tarea 1.0 en etapa 0.0. Se puede incluir `%X{mdc.taskName}` en el patrón de diseño para imprimirlo en los registros.
   Además, se puede hacer uso de la propiedad `spark.sparkContext.setLocalProperty(s"mdc.$nombre","valor")` para añadir datos específicos del usuario al MDC. La clave en el MDC será la cadena `"mdc.$nombre"`. (Apache Spark, s.f.)

3.2.3 PySpark

PySpark es la API de Python sobre Spark, la cual nos permite procesar en tiempo real, datos a gran escala en un entorno distribuido, aprovechando así las capacidades de Spark.

PySpark es compatible con todas las características de Spark, como Spark SQL, DataFrames, Structured Streaming, Machine Learning (MLlib) y Spark Core. (Spark, 2024)

En este proyecto se usó la API de Python sobre Spark por la experiencia previa con este lenguaje sobre otros proyectos.

3.2.4 Comparación con Pandas en paralelo

En el trabajo realizado por Jorge Cabrera (Cabrera, 2023) se hace uso del paquete de Pandas. Este proporciona estructuras de datos rápidas, flexibles y expresivas diseñadas para hacer que el trabajo con datos "relacionales" o "etiquetados" sea fácil e intuitivo. Su objetivo es ser el bloque de construcción de alto nivel fundamental para hacer un análisis práctico de datos en Python. (Pandas Development Team, 2008).

Una de las principales limitaciones de Pandas es que es principalmente monohilo, las operaciones se ejecutan en un solo núcleo o hilo de procesamiento. Esto puede limitar su rendimiento en sistemas con múltiples núcleos o en conjuntos de datos muy grandes.

A raíz de esta limitación surgen otros paquetes como Pandarallel, esta es una librería de Python simple y eficiente para paralelizar las operaciones de Pandas en todas las CPU disponibles. Con un cambio de código de una línea, permite a cualquier usuario de Pandas aprovechar su CPU multinúcleo (Pandaral·lel, s.f.).

Por otra lado, Spark tiene su propia API de Pandas, lo que permite aprovechar todas

ventajas que esta herramienta ofrece, como la escalabilidad a múltiples máquinas en un clúster o la paralelización de las operaciones haciendo uso de todos los núcleos disponibles.

Pandarallel y otros paquetes puede ser una alternativa al ofrecer de una manera fácil y rápida paralelizar las operaciones de DataFrames de Pandas. Sin embargo, al usar Spark se pueden establecer los cimientos para escalar el procesamiento en múltiples máquinas.

3.3 Librerías

3.3.1 Hugging Face

Hugging Face se destaca como una empresa pionera en el ámbito tecnológico, especializándose en el desarrollo de avanzadas herramientas y plataformas basadas en Inteligencia Artificial (IA) para el Procesamiento del Lenguaje Natural (NLP). En su núcleo, la empresa se centra en la creación de soluciones innovadoras que aprovechan el poder del aprendizaje profundo y el procesamiento de lenguaje natural para abordar una amplia gama de desafíos en este campo.

Su plataforma principal, **Hugging Face Hub**, sirve como un recurso integral para la comunidad de investigadores, desarrolladores y entusiastas de NLP, ofreciendo acceso a una rica variedad de conjuntos de datos compartidos y modelos de aprendizaje profundo de código abierto. Estos recursos son fundamentales para la experimentación, investigación y desarrollo de nuevas aplicaciones y soluciones en áreas como la comprensión del lenguaje natural, la generación de texto, el análisis de sentimientos y la traducción automática.

En particular, la biblioteca **Transformers**[3] de Hugging Face ha ganado un reconocimiento destacado en la industria del Procesamiento del Lenguaje Natural. Estos transformadores, que incluyen modelos como BERT, GPT y muchos otros, han demostrado un rendimiento excepcional en una variedad de tareas de procesamiento de lenguaje natural, estableciéndose como referentes en la comunidad y siendo ampliamente utilizados en aplicaciones del mundo real.

3.3.2 SentenceTransformers

SentenceTransformers es una librería de Python que proporciona una interfaz fácil de usar para generar vectores de representación de alta calidad para oraciones y documentos utilizando modelos basados en transformadores preentrenados. Estos modelos, como BERT, RoBERTa y DistilBERT, han demostrado un rendimiento excepcional en una variedad de tareas de procesamiento de lenguaje natural (PLN).

La librería SentenceTransformers proporciona una API simple para cargar modelos preentrenados, generar embeddings de texto y entrenar modelos personalizados en

[3] Hugging Face (s.f.). *Transformers*. https://huggingface.co/docs/transformers/index

nuevos conjuntos de datos, lo que la convierte en una herramienta flexible y potente para investigadores y desarrolladores en el campo del PLN.

En concreto, se utilizó el modelo **all-MiniLM-L6-v2**[4] el cual genera representaciones de texto en 384 dimensiones y puede usarse para tareas como agrupación o búsqueda semántica. Este modelo es 5 veces más rápido que el mejor modelo, **all-mpnet-base-v2**, pero aún así mantiene un alto rendimiento. (Sentence-Transformers, s.f.)

3.3.3 OpenAI API

OpenAI es una empresa de investigación cuyo objetivo es la búsqueda de una inteligencia artificial general que beneficie a la humanidad. De momento han conseguido grandes avances en la inteligencia artificial generativa, siendo pioneros con varios modelos:

- **GPT**: para la generación de texto
- **DALL-E**: cuyo objetivo es principal es la generación de imágenes
- **Whisper**: para transcribir audio a texto y hacer traducciones al inglés.
- **Sora**: es capaz de generar vídeos realistas.

Gracias a la Cátedra Big Data, Open Data y Blockchain (BOB)[5] desarrollada en conjunto por la universidad de La Laguna y Cajasiete, se puedo utilizar la API que nos proporciona OpenAI para hacer uso de sus modelos. En concreto, se utilizó **text-embedding-3-small** para calcular los embeddings de cada una de las direcciones de la BDD. Por defecto codifica cada dirección a un vector de tamaño 1.536 pero se hizo una reducción a 256 debido a los grandes tiempos de carga al operar con una dimensión tan grande. (OpenAI, 2024)

Este modelo nos permite generar 62.500 páginas por cada dólar gastado, siendo cada página de aproximadamente 800 tokens. Esto se traduce en un gasto real final de 0.03 euros por calcular aproximadamente 1.800.000 embeddings.

3.4 Bases de datos vectoriales

Una base de datos vectorial es una infraestructura que permite almacenar datos en una representación matemática, un vector. Para ello, se usa algún modelo de embedding para obtener la representación vectorial de la información.

3.4.1 Chroma DB

Chroma DB es una base de datos vectoriales de código abierto que facilita:

- Almacenar embeddings y sus metadatos.

[4] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[5] Universidad de La Laguna (s.f.). *Cátedra Cajasiete BOB*. https://www.ull.es/catedras/catedrabob/

- Insertar documentos y consultas.
- Buscar embeddings por similitud.

Chroma es una plataforma que incluye un SDK cliente en Python, otro en JavaScript/TypeScript y una aplicación de servidor. Además, Chroma es compatible con varios lenguajes de programación como Java, Ruby, C# y otros, pero solo en modo cliente, lo que permite su integración en una amplia variedad de aplicaciones. En la figura 3.2 se puede encontrar una representación del funcionamiento.

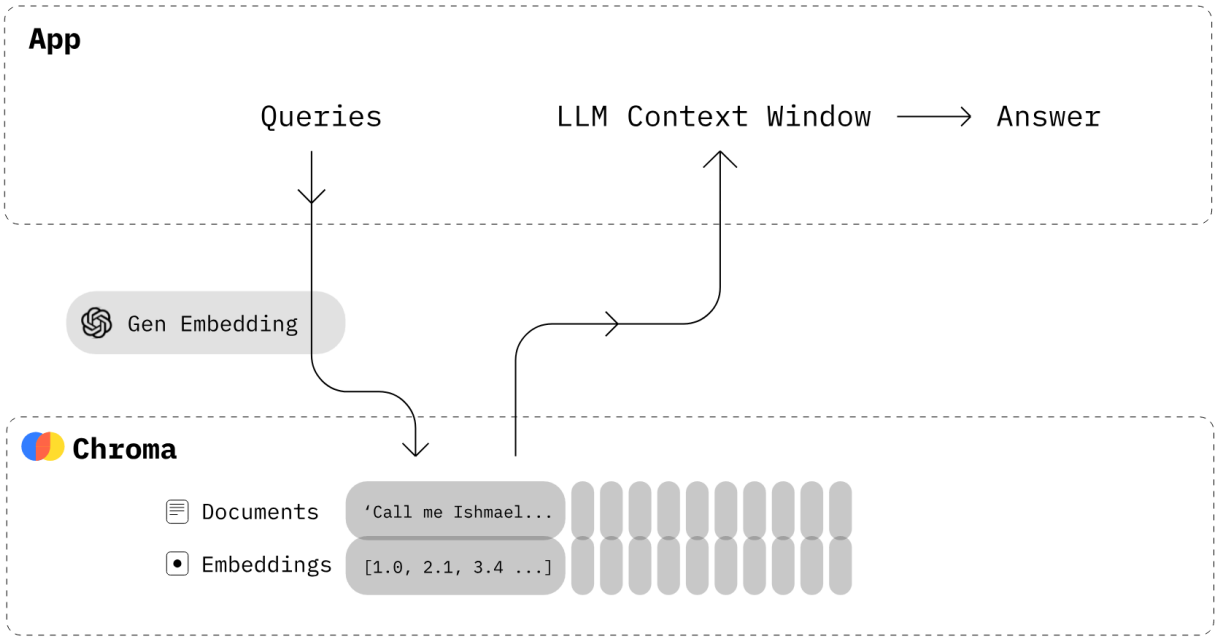


Figura 3.2: Descripción de Chroma DB
Fuente: (Chroma, s.f.)

Por defecto, Chroma usa el modelo **all-MiniLM-L6-v2** para calcular los embeddings pero es compatible con otros proveedores, entre los que se destaca OpenAI, Hugging Face o Google Generative AI. Además, permite crear funciones propias para realizar este cálculo, haciendo uso del protocolo "EmbeddingFunction", siendo posible realizar una pull request para que pase formar parte, de manera oficial, de la herramienta. (Chroma, s.f.)

3.4.2 FAISS

Facebook AI Similarity Search (FAISS) es una librería eficiente para la búsqueda por similitud y clusterización de vectores densos escrito en C++ y cuenta con una interfaz completa para Python, que incluye envoltorios para las versiones 2 y 3 del lenguaje.

Contiene algoritmos que buscan en conjuntos de vectores de cualquier tamaño, hasta los que posiblemente no quepan en la memoria RAM. Además de código de soporte para la evaluación y ajuste de parámetros.

Cabe destacar las siguientes características:

- **Identificación de k-vecinos más cercanos**: No solo encuentra el vecino más cercano, sino también el segundo, tercer y así sucesivamente hasta el k-ésimo vecino más cercano.
- **Procesamiento por lotes**: Permite buscar varios vectores a la vez, lo que mejora la eficiencia en comparación con la búsqueda secuencial.
- **Equilibrio entre precisión y velocidad**: Ofrece opciones para ajustar la precisión de los resultados en función de la velocidad de búsqueda o el consumo de memoria.
- **Soporte para diferentes distancias**: Además de la búsqueda euclidiana, también admite otras métricas de distancia como L1, Linf, etc.
- **Búsqueda de rango**: Permite encontrar todos los elementos dentro de un radio dado del punto de consulta.
- **Almacenamiento del índice en disco**: La capacidad de almacenar el índice en el disco en lugar de la RAM mejora la gestión de datos y la escalabilidad.
- **Indexación de vectores binarios**: Admite la indexación de vectores binarios, no solo de vectores de punto flotante.
- **Filtrado de vectores de índice**: Permite ignorar un subconjunto de vectores de índice según un predicado definido por los IDs de vector. (Facebook Research, s.f.)

A pesar de que tanto Chroma DB como FAISS son proyectos de código abierto y permiten un uso gratuito, se optó por utilizar el primero debido a su mayor simplicidad, con sentencias más sencillas y una curva de aprendizaje menos pronunciada.

Capítulo 4 Implementación

4.1 Descripción de los datos

La base de datos proporcionada por el ISTAC fue creada a partir de datos administrativos de la Comunidad Autónoma de Canarias. Esta contiene un total de 2.234.214 filas y 24 columnas. El esquema se muestra en la tabla 4.1.

Dato	**Descripción**
uuid_idt	Identificador único universal
longitud	Coordenada correspondiente a longitud de la dirección
latitud	Coordenada correspondiente a la latitud de la dirección
link_type, link_description, link_quality, link_quality_rank, link_active, link_active_in, link_active_out	Información correspondiente al enlazamiento
tvia_nn	Tipo de vía no normalizado
nvia_nn	Nombre de vía no normalizado
numer_nn	Número de portal no normalizado
tvia	Tipo de vía (calle, avenida, urbanización, etc)
cvia	Código de vía
nvia	Nombre de la vía

numer	Número de la dirección correspondiente
kmt	Punto kilométrico
hmt	Punto kilométrico, hectómetros
nomedif	Nombre del edificio
codmun	Código postal asociado al municipio
nommun	Nombre del municipio
direccion	Dirección (*tvia+nvia+numer+nommun*)

Tabla 4.1: Descripción de la base de datos

Esta información es utilizada para los propios análisis que realiza el institución, sin embargo, en este proyecto se filtra únicamente por los datos que aportan información relevante, que son: *uuid_idt*, *latitud*, *longitud*, *tvia*, *nvia*, *numer*, *codmun*, *nommun* y *direccion*.

4.2 Inicialización de Apache Spark

El primer paso de la implementación consistió en incorporar Apache Spark utilizando su API de Python, PySpark. Se configuró el entorno con algunos parámetros básicos iniciales pues a medida que se avanzaba en el programa, fue necesario realizar ajustes.

A continuación, se detallan las configuraciones y los comandos ejecutados para establecer el entorno de Spark, que fueron fundamentales para el manejo y procesamiento de grandes volúmenes de datos en este proyecto:

```
from pyspark.sql.session import SparkSession

spark = (

    SparkSession.builder.appName("Address Matching")

    .config("spark.executor.memory", "2g")

    .config("spark.driver.memory", "20g")

    .config("spark.driver.maxResultSize", "4g")

    .config("spark.local.dir", "/spark-tmp") \

    .getOrCreate()

)
```

En la tabla 4.2 se puede encontrar un resumen de los valores de configuración.

Clave	Valor
spark.executor.memory	2gb
spark.driver.memory	20gb
spark.driver.maxResultSize	4gb
spark.local.dir	/spark-tmp

Tabla 4.2: Valores de inicialización de Spark

En el apartado de configuración se comentó que estos parámetros como la asignación de memoria del driver y ejecutores, pueden configurarse en el archivo **spark/conf/spark-env.sh** pero comodidad y para realizar ajustes más rápidamente, se optó por este método. Cabe destacar que en este archivo se especificó que la aplicación se ejecutará en local con el máximo número de núcleos del sistema, que son 8:

…

```
spark.master          local[*]
```

…

spark.driver.maxResultSize limita el tamaño total de los resultados serializados de todas las particiones para cada acción de Spark en bytes. Este valor se tuvo que aumentar debido a que a se abortaban ciertas tareas, como al realizar la operación *collect*, pues le faltaba memoria al driver para almacenar los datos recopilados.

En algunas situaciones durante la ejecución, se lanzaban errores de falta de memoria como el siguiente:

```
… java.io.IOException: No space left on device at
java.io.FileOutputStream.writeBytes(Native Method) at
java.io.FileOutputStream.write(FileOutputStream.java:326 …
```

Una de las causas puede ser que Spark no tenía espacio en la ruta por defecto para crear archivos temporales, los cuales son creados durante la ejecución. Para solucionar esto, se creó un directorio en una ruta alternativa con almacenamiento suficiente, siendo especificado con el parámetro **spark.local.dir**.

A continuación, se realizó la lectura de los datos que se encontraban en un fichero .csv para almacenarlos en una estructura de datos de Spark. A este data frame así generado se le aplicó las tareas de preprocesado que se describen en el siguiente apartado

aprovechando las funcionalidades de procesamiento distribuído de esta tecnología.

4.3 Preprocesado

Cabe destacar que en esta fase de preprocesado de las direcciones, se presentan desafíos específicos debido a su estructura textual y geográfica.

El objetivo principal del preprocesado es normalizar, limpiar y transformar las diferentes columnas y direcciones para que puedan ser utilizadas eficazmente en algoritmos de AA y análisis de similitud A continuación, se describen las diferentes etapas del preprocesado que se han implementado.

4.3.1 Limpieza de los datos

Valores nulos

La base de datos fue proporcionada como un archivo con extensión CSV (Comma Separated Value) por lo que para realizar la exploración de datos de manera más cómoda se usó la herramienta de bases de datos DBeaver[6]. Tras esta investigación se puede ver que el ISTAC dejó campos sin valores. La cantidad de valores nulos encontrados por cada columna se muestran en la tabla 4.3.

Columna	**Valor**
uuid_idt	0
latitud	0
longitud	0
tvia	580.247
nvia	460.106
numer	449.887
codmun	0
nommun	0
direccion	0

Tabla 4.3: Representación de los valores vacíos de dataset

Estos campos sin datos posiblemente son la causa de que no fueron capaces de extraer el valor de la dirección en cuestión debido a que la dirección no es representativa o un error. En la tabla 4.4 se muestra un ejemplo.

[6] DBeaver. Recuperado el 04 de mayo de 2024, de https://dbeaver.io/

tvia	*nvia*	*numer*	*codmun*	*nommun*	*direccion*
	ALFREDO KRAUS 2	0	35001	Agaete	ALFREDO KRAUS 2 AGAETE
		14	35001	Agaete	14 AGAETE
	CALLE LAGUETE BL 5	0	35001	Agaete	CALLE LAGUETE BL 5 AGAETE

Tabla 4.4: Ejemplo de campos con valor nulo

Por lo general, estas filas con el campo *tvia* vacío, son un indicativo que la dirección es pobre, en lo que a información se refiere, o está incompleta por lo que se procedió a dar valor a estos campos con *_U*. Este valor es utilizado por el ISTAC para indicar que el campo es desconocido (unknown).

Extracción del tipo de vía

Anteriormente se observó que hay casos donde la columna *tvia* no tenía un valor asociado a pesar de que este estuviese especificado en la dirección de la columna *direccion*. Comúnmente, el tipo de vía de las direcciones aparece al comienzo, por lo que se puso en práctica una estrategia para extraer el tipo de vía de la dirección y asignarlo al campo correspondiente en el caso de que esté vacío.

Antes de la extracción y asignación del tipo de vía de la dirección, se tenían 580.357 filas con valor desconocido y tras este proceso el valor se redujo hasta 119.382.

Limpieza de entradas con valor *_U*

Finalmente, se procedió a eliminar cualquier fila que contenga en algunos de sus campo el valor *_U*, dejando así un dataset de 2.114.832 filas

4.3.2 Aumento de los datos

Tras calcular y realizar una análisis sobre cuántas direcciones hay asociadas por cada *uuid_idt*, en la tabla 4.5 se observa que hay una gran cantidad que únicamente tiene asociada una sola dirección, aportando así poca información a la hora de utilizar estos datos con los modelos de representación.

Número de direcciones asociadas por *uuid_idt*	Frecuencia
1	220.643
2	117.020

3	67.701
4	41.582
5	27.379
6	18.218
7	12.863
8	9.688
9	7.133
10 o más	34.916

Tabla 4.5: Número de direcciones por uuid_idt antes el aumento de datos

Para evitar la pérdida de información, se propuso generar nuevas direcciones a partir de la ya existentes usando las siguientes ideas:

- Añadir errores ortográficos, como si de un ser humano se tratase, cambiando la posición de dos letras adyacentes de una palabra.
- Cambiar el tipo de vía de una dirección por otra de las que aparecen en la BDD.

Por ejemplo, a partir de la dirección *CALLE CASTRILLO 48 P02 LAS PALMAS DE GRAN CANARIA*, se obtuvo *URBANIZACION CASTRILLO 48 P02 LAS APLMAS DE GRAN CANARIA*. Donde se le cambió el tipo de vía y se le introdujo un error ortográfico (*LAS APLMAS*).

Inicialmente, la generación de direcciones se utilizó para los casos con 5 o menos direcciones asociadas por *uuid_idt*. Este criterio fue posteriormente ajustado para incluir aquellos casos con una cantidad menor o igual al valor especificado en la ejecución. A partir del conjunto de direcciones resultantes, se incrementó el número de direcciones mediante la selección aleatoria de un subconjunto del 50%.

Con un valor máximo de 20 direcciones por *uuid_idt*, el total de filas aumentó de 2.114.725 a 2.267.802 tras la generación de nuevas direcciones. Esto permitió una distribución más equilibrada de direcciones por *uuid_idt*, tal como se muestra en la tabla 4.6.

Número de direcciones asociadas por *uuid_idt*	Frecuencia
1	110.507
2	168.560
3	33.874
4	79.492
5	13.706
6	42.891
7	6.424
8	25.482
9	3511
10 o más	72.659

Tabla 4.6: Número de direcciones por uuid_idt tras el aumento de datos

4.3.3 División de los datos

Después de finalizar el proceso de limpieza y ampliación de datos, es esencial dividir el conjunto de datos en dos partes distintas: una para entrenar el modelo y otra para validar los resultados obtenidos.

Para llevar a cabo esta división de manera efectiva, primero se recopilaron todos los valores de *uuid_idt* presentes en el conjunto de datos. Luego, para cada *uuid_idt*, se realizaron divisiones equitativas de sus direcciones asociadas, con una proporción del 80 y 20 por ciento para el conjunto de datos de entrenamiento y validación respectivamente. Esto se traduce en un dataset de entrenamiento con un total de 1.690.969 filas y un conjunto de validación con 420.496 filas.

```
# Obtener los uuids únicos y convertirlos a una lista de Python
unique_uuids = [row[0] for row in
addresses_df.select("uuid_idt").distinct().collect()]


# Crear DataFrame del 80%
train_df = addresses_df.sampleBy(
    "uuid_idt", fractions={uuid: 0.8 for uuid in unique_uuids},
seed=42
```

```
)


# Crear DataFrame del porcentaje restante restando el DataFrame
del 80% al original

test_df = addresses_df.subtract(train_df)
```

4.4 Procedimiento de la asociación de direcciones

Una vez preprocesados los datos y dividido el dataframe en conjuntos de entrenamiento y validación, se inició la búsqueda de soluciones para la asociación de direcciones. En un inicio, se consideró utilizar MLlib, la biblioteca de ML de Spark, para aprovechar modelos de clasificación como RandomForest (Apache Spark, RadomForest), además de desarrollar un modelo propio beneficiándose del cómputo distribuido y la escalabilidad que Spark ofrece.

Tras probar la ejecución con este modelo, se evidenció que los recursos del sistema no eran suficientes para soportar la carga de utilizar todos los datos, por lo que se propuso descubrir el límite que era capaz de soportar. Al realizar varios experimentos, se llegó a la conclusión de que la ejecución del modelo daría error si los datos utilizados eran superiores al 15% del dataset original. Esta cantidad de datos no era suficiente para que el modelo pudiese clasificar las direcciones con una tasa de acierto superior al 2%.

Así pues, teniendo en cuenta la gran cantidad de datos y las limitaciones de los recursos del sistema, se propuso probar la asociación de direcciones utilizando diferentes modelos preentrenados, que sería una estrategia que requeriría menos coste computacional

Con este nuevo enfoque definido, se procedió a calcular los embeddings para cada una de las direcciones en ambos conjuntos, utilizando los modelos descritos en la tabla 4.7.

Modelo	**Tamaño del embedding**
word2vec	200
text-embedding-3-small	256
all-MiniLM-L6-v2	384

Tabla 4.7: Modelos usados en el proyecto

Estos embeddings tienen la propiedad de que direcciones similares tendrán una distancia del coseno similar, por lo que el algoritmo que se plantea debe obtener la representación vectorial de la dirección que se quiera asociar a alguna de la base de datos, para posteriormente extraer aquellas con embeddings más similares. Es un algoritmo simple, pero costoso computacionalmente dado el volumen de la información que se maneja. Se

realiza, una operación *inner join*, la cual selecciona registros que tienen valores coincidentes en ambas tablas (direcciones georreferencias, frente a las que se van a asociar), de forma que cada una de las direcciones para asociar se enfrenta a todas las direcciones ya georreferencias, computándose la similitud del coseno con cada emparejamiento. Esta operación produce como resultado un dataframe de tamaño $m \times n$ (siendo m y n el tamaño de los dataframes de las tablas mencionadas). Finalmente, como solución se seleccionan las 3 direcciones más próximas. Véase el método del cálculo de similitudes con Spark.

En el proceso de validación se produce un problema de coste computacional en términos de tiempo y memoria, ya que se necesita contabilizar los casos de acierto, que requiere una operación count entre otras. Esta operación permite contar el tamaño del dataframe pero para ello, es necesario recopilar todos los datos en el driver de Apache Spark. El principal problema es que este componente no tenga suficiente memoria asignada por lo que daría un error como el siguiente:

…

```
Uncaught error from thread [spark-akka.actor.default-dispatcher-3] shutting down JVM since 'akka.jvm-exit-on-fatal-error' is enabled for ActorSystem[spark]

java.lang.OutOfMemoryError: Java heap space
```

…

Se podría pensar que la solución es asignar más memoria a este componente, lo cual es correcto, pero si no se realiza con cierto sentido, podría generar más problemas al dejar sin recursos a la propia máquina, generando así otros fallos como el siguiente, el cual es causado porque el OOM Killer mata el proceso de Java:

```
***"ERROR:py4j.java_gateway:Error while sending or receiving.

Traceback (most recent call last):

  File "..., in send_command

    raise Py4JError("Answer from Java side is empty")

Py4JError: Answer from Java side is empty
```

El OOM killer es un mecanismo en el kernel de Linux que se activa cuando el sistema se queda sin memoria y debe liberar recursos para seguir operando. Este mecanismo selecciona y termina procesos específicos para liberar memoria y evitar que el sistema se bloquee por falta de recursos por lo que desactivar esta herramienta puede ocasionar problemas en la ejecución (Chase, 2013).

Tras realizar variaciones en la configuración de Apache Spark, intentando ajustar el valor de memoria de los diferentes componentes (drivers y ejecutores), se optó por usar una

base de datos de embeddings.

4.5 Chroma DB

Chroma DB se utilizó en este proyecto como una alternativa para almacenar los embeddings de las direcciones, facilitando además el cálculo de la similitud con consultas más simples. Integrar Chroma es sencillo: basta con instalar su librería y crear una colección para almacenar la información para luego realizar consultas.

Tras crear los dataframes de entrenamiento y validación, se recopiló la información y se convirtió en listas para su almacenamiento en dos colecciones separadas (validación y entrenamiento).

La información se almacenó en Chroma siguiendo la siguiente estructura, de modo que cada valor de las listas corresponde con una fila del dataframe de Spark:

- **Documentos**: Lista con las direcciones.
- **Embeddings**: Lista con los embeddings correspondientes a cada dirección.
- **ID**: Lista de identificadores numéricos únicos, que fueron generados a partir de un iterador, con valores desde 0 hasta el tamaño del dataframe.
- **Metadatos**: Lista de objetos que almacenan el *uuid_idt*, la latitud y la longitud de cada dirección.

Al intentar cargar todos los datos en las colecciones correspondientes con la sentencia *add*, se obtuvo el siguiente error:

```
/chromadb/db/mixins/embeddings_queue.py", line 127, in
submit_embeddings raise ValueError( ValueError:  Cannot submit more
than 41,666 embeddings at once. Please submit your embeddings in
batches of size 41,666 or less.
```

Este error se debe a las limitaciones que tiene esta librería al usar SQLite como base de datos encargada de almacenar la información. Por lo que se procedió a crear un método para cargar los datos de manera gradual, en lotes de 41.666.

Una vez cargadas las dos colecciones con los datos, se procede a identificar los mejores resultados, usando la sentencia *query*[7], para cada dirección de la colección de validación utilizando la distancia del coseno que ya viene definida con Chroma. Esta sentencia recibe un embedding y retorna un diccionario con los n-resultados más similares.

A continuación, se inspeccionaron los metadatos de los resultados en busca de la igualdad entre el *uuid_idt* de la dirección de validación y los resultantes, contabilizando así los casos con match o no match. Véase el método de evaluación en Croma DB.

Sin embargo, se observó que los tiempos de extracción y carga de Chroma DB formaban

[7] https://docs.trychroma.com/getting-started#5.-query-the-collection

un cuello de botella por lo que se optó por reducir el problema por municipios, y ya que se había puesto en práctica el funcionamiento de esta librería, se podrían realizar comparaciones con Apache Spark.

Para identificar el municipio con más direcciones asociadas, se realizó un estudio detallado. El resultado mostró que el municipio de San Cristóbal de La Laguna tiene el mayor número de direcciones, con un total de 206.730. La figura 4.1 ilustra la distribución de direcciones por municipios, destacando aquellos con un mayor volumen de datos.

Figura 4.1: Municipios de la BDD con más direcciones
Fuente: Elaboración propia

Finalmente, tras ejecutar el programa haciendo uso de los diferentes modelos, y distintas frecuencias de direcciones por *uuid_idt*, se procedió a realizar un análisis de resultados.

A modo de resumen de los experimentos realizados se presenta un esquema que muestra la ejecución del programa en la figura 4.2:

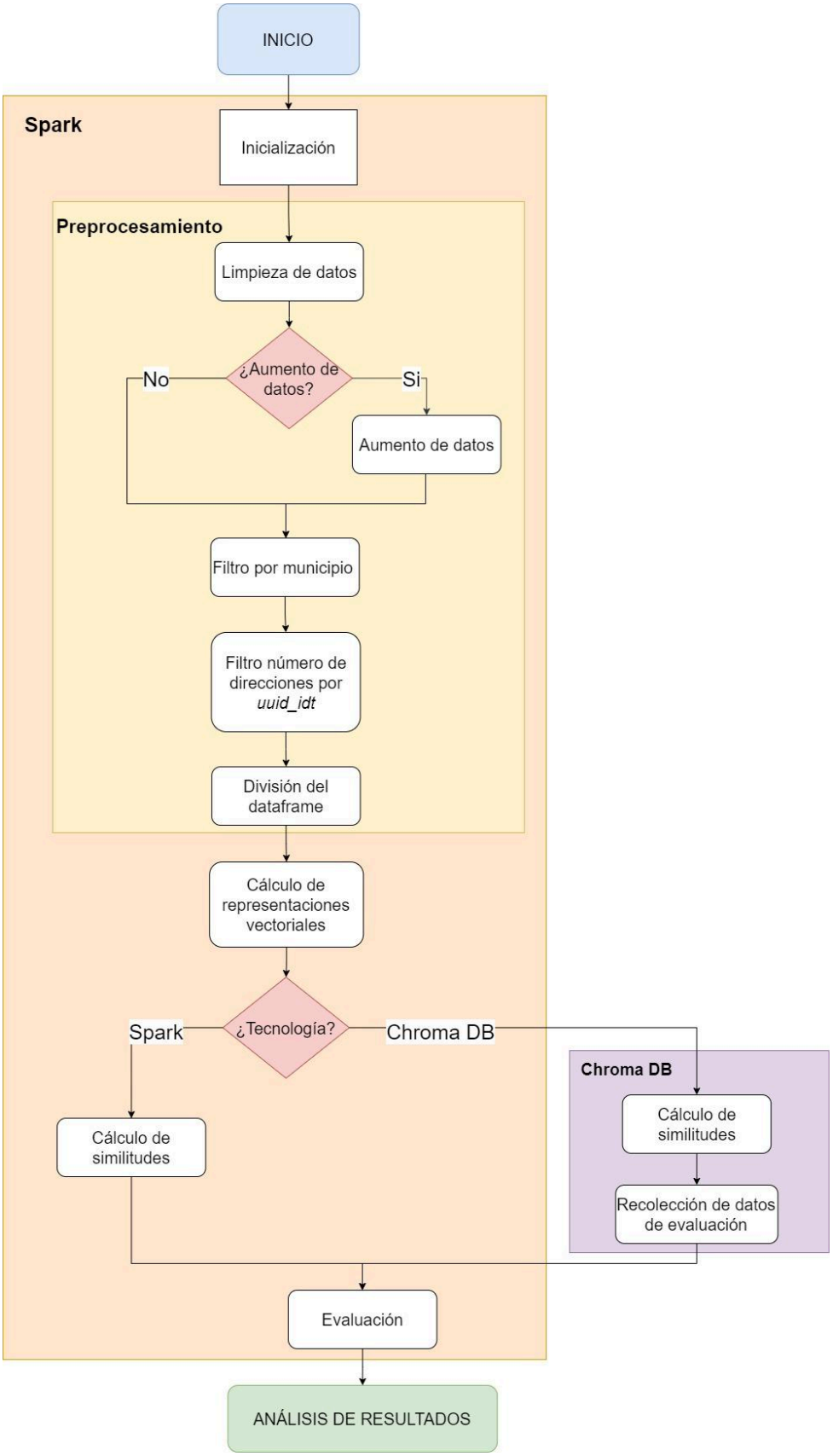


Figura 4.2: Diagrama de flujo del proyecto
Fuente: Elaboración propia

Capítulo 5 Resultados

La solución que se ha expuesto en esta memoria requiere la toma de decisiones respecto a diferentes aspectos bien sean parámetros, estrategias o tecnologías. En este capítulo se analizan los resultados de los experimentos llevados a cabo con objeto de determinar el comportamiento ante ellos y determinar su influencia en la solución que se obtenga. En concreto, los aspectos clave analizados son:

- **Número mínimo de direcciones por *uuid_idt***: realizar ajustes en este valor puede indicar si la tasa de acierto aumenta al usar *uuid_idt* que tienen más direcciones asociadas.
- **Aumento de datos**: es necesario comprobar si crear direcciones "artificiales" tiene efectos positivos o negativos.
- **Tecnología**: como se comentó anteriormente, es interesante probar si el rendimiento de los métodos propios creados con Spark son comparables con usar Chroma DB y sus herramientas.
- **Modelos**: se realizarán pruebas con los modelos especificados anteriormente. Word2vec (W2v), text-embedding-3-small(TE3S) y all-MiniLM-L6-v2 (MiniLM).

En una primera ronda de pruebas y tras obtener unos resultados no tan acertados, surgió la intriga de conocer qué tan lejos estaban los N mejores resultados de coincidir con la dirección a evaluar, y cómo se tenían las coordenadas de las direcciones, se podía calcular esta distancia.

La distancia de Haversine es una medida que permite conocer la distancia entre dos puntos en una esfera y por tanto, la distancia entre las coordenadas geográficas de dos puntos. En las siguientes tablas de resultados, se muestra la distancia de Haversine media, en kilómetros, de los N mejores resultados obtenidos para cada una de las direcciones, de modo que a menos valor, más cerca están entre sí. La fórmula es la siguiente:

$$d = 2r \cdot arcsin\left(\sqrt{sin^2\left(\frac{\Delta\varphi}{2}\right) + cos(\varphi 1) \cdot cos(\varphi 2) \cdot sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

Donde:

- r es el radio de la esfera (para la Tierra, aproximadamente 6371 km).
- $\varphi 1$ y $\varphi 2$ son las latitudes del primer y segundo punto, respectivamente, en radianes.

- $\lambda 1$ y $\lambda 2$ son las longitudes del primer y segundo punto, respectivamente, en radianes.
- $\Delta\varphi = \varphi 2 - \varphi 1$ es la diferencia de latitudes.
- $\Delta\lambda = \lambda 2 - \lambda 1$ es la diferencia de longitudes. (Greyrat, 2022)

Cabe destacar que el tiempo de ejecución mostrado en las siguientes tablas tiene añadido el tiempo requerido para realizar el preprocesado de los datos, que es de 492 segundos.

Número de direcciones por *uuid_idt*: 50. **Aumento de datos**: No.						
Modelo	**Tecnología**	**Tamaño dataframe entrena.**	**Tamaño dataframe validación**	**Tasa de acierto**	**Distancia de Haversine media (kilómetros)**	**Tiempo de ejecución (segundos)**
W2v	Spark	20.635	4.986	0.95	0.56	3.426
W2v	Chroma	20.635	4.986	0.7	1.62	669
TE3S	Spark	20.635	4.986	0.95	0.52	8.158
TE3S	Chroma	20.635	4.986	0.71	1.83	3.510
MiniLM	Spark	20.635	4.986	0.95	0.5	3.930
MiniLM	Chroma	20.635	4.986	0.72	1.71	853

Tabla 5.1: Comparación entre modelos y tecnologías utilizadas

En la tabla 5.1, al usar un número mínimo de direcciones por *uuid_idt* igual a cincuenta, se refleja una clara diferencia en el desempeño dependiendo de la tecnología utilizada. Al emplear los mismos dataframes y modelos, Spark generalmente obtiene tasas de acierto superiores a las de Chroma DB en más del veinte por ciento. No obstante, el tiempo de ejecución para Chroma DB es, en promedio, un sesenta y ocho por ciento inferior al de Spark.

La distancia de Haversine media ofrece una perspectiva interesante sobre la calidad de las asignaciones realizadas por cada modelo. Se observa que, en promedio, las distancias de Haversine son más altas en los modelos implementados en Chroma DB en comparación con los de Spark. Esto sugiere que las asignaciones realizadas por los modelos de Chroma DB tienden a estar más distantes de la dirección de referencia en comparación con los modelos de Spark.

Además, cabe destacar que en pruebas posteriores no se utilizó el modelo text-embedding-3-small (TE3S) debido a su rendimiento inferior en comparación con otros modelos y a que su tiempo de ejecución fue más del doble.

Número mínimo de direcciones por *uuid_idt*: 20 **Aumento de datos**: No						
Modelo	**Tecnología**	**Tamaño dataframe entrena.**	**Tamaño dataframe validación**	**Tasa de acierto**	**Distancia de Haversine media (kilómetros)**	**Tiempo de ejecución (segundos)**
W2v	Spark	42.096	10.227	0.92	0.73	11.378
W2v	Chroma	29.435	7.145	0.53	1.97	1.331
MiniLM	Spark	29.435	7.145	0.83	0.92	6.692
MiniLM	Chroma	29.435	7.145	0.6	1.84	1.510

Tabla 5.2: Resultados sin aumento de datos y mínimo de 20 direcciones por uuid_idt

En la tabla 5.2, tras utilizar un número mínimo de direcciones por *uuid_idt* igual a 20, se observa que la tecnología utilizada influye significativamente en el rendimiento de los modelos. En general, se observa que los modelos implementados en Spark superan en rendimiento a los implementados en Chroma DB en términos de tasa de acierto. Por ejemplo, los modelos W2v y MiniLM implementados en Spark lograron tasas de acierto de 0.92 y 0.83 respectivamente, mientras que sus contrapartes en Chroma DB obtuvieron tasas de 0.53 y 0.6 respectivamente.

Aunque los modelos implementados en Spark tienden a ofrecer un mejor rendimiento en términos de tasa de acierto, el tiempo de ejecución es significativamente más alto en comparación con los modelos de Chroma DB. Por ejemplo, el tiempo de ejecución para los modelos implementados en Spark oscila entre 6.692 y 11.378 segundos, mientras que para los modelos de Chroma DB es considerablemente menor, entre 1.331 y 1.84 segundos.

Número mínimo de direcciones por *uuid_idt*: 50							
Modelo	**Tecnología**	**Aumento de datos**	**Tamaño dataframe entrena.**	**Tamaño dataframe validación**	**Tasa de acierto**	**Distancia de Haversine media (kilómetros)**	**Tiempo de ejecución (segundos)**
MiniLM	Chroma	No	20.635	4.986	0.72	1.71	853
MiniLM	Chroma	Si	33.987	19.360	0.72	2.19	3.386

Tabla 5.3: Comparación de resultados con y sin aumento de datos

Al observar la tabla 5.3 y comparar los resultados del modelo MiniLM implementado en Chroma DB sin aumento de datos con los resultados obtenidos con aumento de datos, se

observa que no hay una mejora significativa en la tasa de acierto. Ambos modelos logran una tasa de acierto del 72%. Sin embargo, el aumento de datos resulta en un aumento significativo en el tamaño del dataframe de entrenamiento y validación, lo que afecta la eficiencia computacional.Sin embargo, incluso con el aumento de datos, el tiempo de ejecución para el modelo MiniLM en Chroma DB sigue siendo relativamente bajo en comparación con otros modelos y tecnologías.

Número mínimo de direcciones por *uuid_idt*: 20							
Modelo	**Tecnología**	**Aumento de datos**	**Tamaño dataframe entrena.**	**Tamaño dataframe validación**	**Tasa de acierto**	**Tiempo de ejecución sin evaluación(segundos)**	**Tiempo de ejecución (segundos)**
W2v	Chroma	Si	43.503	24.676	0.54	686	10.116
MiniLM	Chroma	Si	43.503	24.676	0.58	1.989	11.750

Tabla 5.4: Mejores modelos con aumento de datos y mínimo de 20 direcciones por uuid_idt

Al comparar los resultados de la tabla 5.4, donde se hace uso de los modelos W2v y MiniLM implementados en Chroma DB con aumento de datos, se observa que el modelo MiniLM tiene un rendimiento ligeramente superior en términos de tasa de acierto, con un 58% frente al 54% del modelo W2v. Esto sugiere que el modelo MiniLM puede ser más efectivo para la tarea de asociación de direcciones postales en este contexto específico.

Ambos modelos muestran un aumento en el tamaño del dataframe de entrenamiento y validación, con respecto a la tabla 5.2, debido al aumento de datos. Sin embargo, este aumento no se traduce en una mejora en la tasa de acierto. Aunque el modelo MiniLM tiene una tasa de acierto ligeramente superior, el aumento de datos tiene un impacto limitado en la mejora del rendimiento del modelo.

La comparación entre Chroma DB y Spark revela diferencias significativas en varios aspectos clave. En términos de precisión, Spark supera consistentemente a Chroma DB en más del veinte por ciento, lo que subraya su capacidad para asociar direcciones postales con mayor exactitud. Sin embargo, esta mejora en la precisión viene acompañada de un tiempo de ejecución más prolongado en Spark. Es esencial tener en cuenta que los tiempos de ejecución reportados para Spark incluyen la fase de evaluación, donde pueden emplearse operaciones no distribuidas como el *count*, lo que implica traer todos los datos al driver, aumentando así considerablemente el tiempo de procesamiento.

Además, al evaluar el costo computacional, Spark ofrece la ventaja de manejar eficazmente grandes volúmenes de datos y su inherente escalabilidad, lo que lo convierte

en una opción ideal para aplicaciones que requieren o desean un procesamiento distribuido. Por otro lado, Chroma DB puede ser más conveniente en situaciones más simples donde se necesita un procesamiento rápido y eficiente, especialmente al trabajar con conjuntos de datos más pequeños.

Capítulo 6 Conclusiones y líneas futuras

El presente trabajo demuestra la viabilidad de usar técnicas avanzadas de procesamiento de lenguaje natural y aprendizaje automático en la tarea de la asociación de direcciones postales. Gracias a la implementación de Apache Spark y su API, PySpark, se ha logrado manejar y procesar eficientemente una gran base de datos proporcionada por el Instituto Canario de Estadística (ISTAC), que contiene más de dos millones de registros.

El proyecto se ha centrado en varias etapas clave, incluyendo la limpieza y el preprocesamiento de los datos, la optimización del entorno de Apache Spark para el procesamiento de distribuido, y la aplicación de modelos preentrenados para calcular representaciones vectoriales de las direcciones. La integración de Chroma DB ha permitido mejorar la eficiencia del almacenamiento y consulta de embeddings, aunque no ha conseguido obtener mejores resultados en las asignaciones.

A lo largo del proyecto, se han experimentado con múltiples configuraciones y enfoques para optimizar el rendimiento del sistema. Esto ha incluido ajustes en la asignación de memoria y el procesamiento de grandes volúmenes de datos.

Una de las principales limitaciones de este trabajo fue la capacidad de hardware de la máquina virtual utilizada. La gran cantidad de datos requeridos para este proyecto superó los recursos disponibles, lo que obligó a adaptar y reducir el problema a un volumen más manejable. Sería de gran interés evaluar la eficacia de los modelos aplicando la asociación al dataframe completo sin restricciones y realizar la ejecución en un clúster con múltiples sistemas para mejorar el rendimiento.

Apache Spark dispone de la biblioteca Spark MLlib, que permite crear modelos de clasificación propios aprovechando la escalabilidad y el procesamiento distribuido que ofrece. Sin embargo, el costo computacional de entrenar un modelo con todos los datos disponibles superó los recursos actuales. Queda pendiente la creación efectiva de un modelo propio y el ajuste o fine-tuning de los modelos utilizados, así como la exploración de otros modelos potencialmente más eficaces.

Capítulo 7 Summary and Conclusions

This work demonstrates the feasibility of using advanced natural language processing and machine learning techniques for the task of postal address matching. Thanks to the implementation of Apache Spark and its API, PySpark, it has been possible to efficiently manage and process a large database provided by the Canary Islands Institute of Statistics (ISTAC), containing more than two million records.

The project has focused on several key stages, including data cleaning and preprocessing, optimizing the Apache Spark environment for distributed processing, and applying pre-trained models to calculate vector representations of the addresses. The integration of Chroma DB has improved the efficiency of embedding storage and querying, although it has not achieved better results in assignments.

Throughout the project, multiple configurations and approaches have been experimented with to optimize system performance. This has included adjustments in memory allocation and the processing of large volumes of data.

One of the main limitations of this work was the hardware capacity of the virtual machine used. The large amount of data required for this project exceeded the available resources, necessitating adapting and reducing the problem to a more manageable volume. It would be of great interest to evaluate the effectiveness of the models by applying the association to the complete dataframe without restrictions and executing it in a cluster with multiple systems to improve performance.

Apache Spark has the Spark MLlib library, which allows creating custom classification models by leveraging the scalability and distributed processing it offers. However, the computational cost of training a model with all available data exceeded the current resources. The creation of an effective custom model and the adjustment or fine-tuning of the models used, as well as the exploration of other potentially more effective models, remain pending tasks.

Capítulo 8 Presupuesto

8.1 Personal

Puesto de trabajo	Costo
Científico de datos junior (Jobted, s.f.)	10,2 €/h (brutos)

Tabla 8.1: Salario de científico de datos junior

8.2 Desglose de tiempos

Trabajo realizado	Tiempo requerido (horas)
Estudio inicial	20
Análisis exploratorio de datos	12
Configuración de Spark	20
Preprocesado	17
Pruebas con la librería MLib	30
Uso de modelos de AA	33
Cálculo de similitudes	31
Implementación de Chroma DB	26
Análisis de resultados	5
TOTAL	194

Tabla 8.2 : Desglose de tiempo requerido por trabajo realizado

8.3 Coste computacional

El uso del modelo Text-embedding-3-small tiene un costo de 1 dólar por 50 millones de tokens, y como en este proyecto se usaron aproximadamente 29 millones de tokens, se traduce en un costo total de 0,5 euros.

Uso	Costo
API de OpenAI	0,5 €
Servidor de características similares (OVH Cloud, s.f.)	54 €/mes
TOTAL	54,5 €

Tabla 8.3: Coste computacional

8.4 Costo total

Costo de personal	Costo de cómputo	TOTAL
1978,8 €	54,5 €	2.033,3 €

Tabla 8.4: Costo total

Capítulo 9 Funciones destacables

9.1 Método del cálculo de similitudes con Spark

```
def get_top_matches(
    test_df: DataFrame,
    train_df: DataFrame,
    embedding_col: str,
    address_col: str,
    top_n: int = 3,
) -> DataFrame:
    """Return the top N results of the cosine similarity computation.

    Args:
        test_df: DataFrame containing the addresses to evaluate.
        train_df: DataFrame containing the addresses used for training.
        embedding_col: Name of the column containing the precomputed embeddings.
        address_col: Name of the column containing the addresses.
        top_n: Number of top results to return for each address in test_df.

    Returns:
          DataFrame containing the addresses from test_df and their top N matches
from train_df based on cosine
    """
    test_df = test_df.alias("test_df")
    train_df = train_df.alias("train_df")

    cross_df = test_df.crossJoin(train_df)
    cross_df = cross_df.withColumn(
        "cosine_similarity",
        cosine_similarity_udf(
```

```
            col(f"test_df.{embedding_col}"), col(f"train_df.{embedding_col}")
        ),
    )

    window = Window.partitionBy(col(f"test_df.{address_col}")).orderBy(
        col("cosine_similarity").desc()
    )

    cross_df = cross_df.withColumn("rank", row_number().over(window))
    top_n_df = cross_df.filter(col("rank") <= top_n)

    return top_n_df
```

9.2 Método de evaluación en Croma DB

```
def evaluation_collections(
    collection_test: chromadb.api.models.Collection.Collection,
    collection_train: chromadb.api.models.Collection.Collection,
):
    """
    Args:
        collection_test: Collection with the data to search for similar.
        collection_train: Collection to query the n best embeddings.

    Returns:
        match: Number of hits.
        no_match: Numbers of no hits.
        evaluated_dirs: Dataframe with the made comparations.
    """
    evaluated_dirs = spark.createDataFrame([], schema)

    counter = 0
    match = 0
    no_match = 0
    while counter < collection_test.count():
```

```
        data_to_evaluate = collection_test.get(
            ids=[str(counter)], include=["embeddings", "metadatas", "documents"]
        )

        embedding_to_evaluate = data_to_evaluate["embeddings"][0]
        uuid_to_evaluate = data_to_evaluate["metadatas"][0]["uuid_idt"]

        best_similarities = collection_train.query(
            query_embeddings=[embedding_to_evaluate], n_results=3
        )

        best_uuids = [u["uuid_idt"] for i in best_similarities["metadatas"] for u
in i]

        if uuid_to_evaluate in best_uuids:
            match += 1
        else:
            no_match += 1
            evaluated_dirs = extract_data(data_to_evaluate, best_similarities,
evaluated_dirs)
        counter += 1
    return match, no_match, evaluated_dirs
```

Bibliografía

Apache Spark. (2024, 24 de febrero). *PySpark Overview – Pyspark master documentation*. https://spark.apache.org/docs/3.5.1/api/python/index.html

Apache Spark. (s.f.). *Cluster Mode Overview - Spark 3.5.1 Documentation*. Recuperado el 23 de abril, 2024, de https://spark.apache.org/docs/3.5.1/cluster-overview.html

Apache Spark. (s.f.). *Configuration - Spark 3.5.1 Documentation*. Recuperado el 24 de abril, 2024, de https://spark.apache.org/docs/3.5.1/configuration.html

https://www.aprendemachinelearning.com/como-funcionan-los-transformers-espanol-nlp-gpt-bert/

Apache Spark. (s.f.) *RandomForest - Pyspark master documentation*. Recuperado el 20 de mayo, 2024, de https://spark.apache.org/docs/3.5.1/api/python/reference/api/pyspark.mllib.tree.RandomForest.html?highlight=randomforest#pyspark.mllib.tree.RandomForest

Baeldung. (2024, 18 de marzo). *Semantic Similarity of Two Phrases*. https://www.baeldung.com/cs/semantic-similarity-of-two-phrases

Bagnato,J. I., (2022). *¿Cómo funcionan los Transformers? en Español*. Aprende Machine Learning en Español. https://www.aprendemachinelearning.com/como-funcionan-los-transformers-espanol-nlp-gpt-bert/

Bao, H. (s.f.). *Visualize cosine similarity of word pairs*. Recuperado el 17 de mayo, 2024, de https://psychbruce.github.io/PsychWordVec/reference/plot_similarity.html

Cabrera, J. (2023). *Procesamiento de Lenguaje Natural en direcciones postales.* (Trabajo de Fin de Grado). Universidad de La Laguna. http://riull.ull.es/xmlui/handle/915/33992

Çano, E. (2018). *Text-based Sentiment Analysis and Music Emotion Recognitio*n. (Tesis doctoral) . Universidad Politécnica de Torino. https://www.researchgate.net/publication/328160770_Text-based_Sentiment_Analysis_and_Music_Emotion_Recognition

Chase, R. (2013). *How to Configure the Linux Out-of-Memory Killer*. Oracle. https://www.oracle.com/technical-resources/articles/it-infrastructure/dev-oom-killer.html

Chatterjee, A. (s.f.). *Minkowski distance*. OpenGenus. Recuperado el 17 de mayo, 2024, de https://iq.opengenus.org/minkowski-distance/

Chroma DB. (s.f.). *Chroma*. Recuperado el 22 de abril, 2024, de https://docs.trychroma.com/

Comber, S., & Arribas-Bel, D. (2019). Machine learning innovations in address matching: A

practical comparison of word2vec and CRFs. Transactions in GIS, 23(2), 334–348. https://doi.org/10.1111/tgis.12522

Datacamp. (s.f.) *Introduction to Natural Language Processing in R*. Recuperado el 15 de mayo, 2024, de https://campus.datacamp.com/courses/introduction-to-natural-language-processing-in-r/representations-of-text?ex=12

DeepLearning.AI. (2023, 11 de enero). *Natural Language Processing*. https://www.deeplearning.ai/resources/natural-language-processing/

Espejel, O. (2022, 23 de junio). Getting Started With Embeddings. *Hugging Face.* https://huggingface.co/blog/getting-started-with-embeddings

Facebook Research. (s. f.). *Home / facebookresearch/faiss Wiki*. Recuperado el 27 de abril, 2024, de https://github.com/facebookresearch/faiss/wiki

González, J. A., Betancor, R., & Hernández, M. S. (2021). *Sistema de georreferenciación para fines estadísticos*. Instituto Canario de Estadística (ISTAC). https://jecas.es/wp-content/uploads/2021/11/21.4.ISTAC_Sistema-georreferenciacion.pdf

Greyrat, R. (2022). *Fórmula Haversine para encontrar la distancia entre dos puntos en una esfera.* Barcelona Geeks. https://barcelonageeks.com/formula-de-haversine-para-encontrar-la-distancia-entre-dos-puntos-en-una-esfera/

Guermazi Y., Sellami S, Boucelma O. (2023). *GeoRoBERTa: A Transformer-based Approach for Semantic Address Matching*. CEUR Workshop Proceedings. https://ceur-ws.org/Vol-3379/DARLI-AP_2023_1.pdf

Gupta, S. (2019). Top 5 Distance Similarity Measures implementation in Machine Learning. Top 5 Distance Similarity Measures implementation in Machine. Medium. https://medium.com/@gshriya195/top-5-distance-similarity-measures-implementation-in-machine-learning-1f68b9ecb0a3

Hoffman, H. (2023). Embeddings and Vector Database with ChromaDB. *Real Python.* https://realpython.com/chromadb-vector-database/#word-embeddings

Jobted. (s.f.) *Sueldo del Data Scientist en España*. Recuperado el 24 de mayo, 2024, de https://www.jobted.es/salario/data-scientist

Miesle, P. (2023, 29 de septiembre). What is Cosine Similarity: A Comprehensive Guide. *Datastax*. https://www.datastax.com/guides/what-is-cosine-similarity

OpenAI. (2024, 25 de enero). *New embedding models and API updates*. https://openai.com/blog/new-embedding-models-and-api-updates#OpenAI

OVH Cloud. (s.f.). *Alquiler de servidores para empresas*. Recuperado el 23 de mayo, 2024, de https://www.ovhcloud.com/es-es/bare-metal/advance/

Pandaral·lel. (s.f.). *Pandaral·lel documentation*. Recuperado el 21 de mayo, 2024, de https://nalepae.github.io/pandarallel/

Pandas Development Team. (2008). *Pandas: powerful Python data analysis toolkit*. GitHub. https://github.com/pandas-dev/pandas

Sentence-Transformers. (s.f.). *SentenceTransformers Documentation*. Recuperado el 22 de mayo, 2024, de https://www.sbert.net/index.html

Syne, L. (2022). *Machine learning and NLP approaches in address matching*. Trabajo Fin de Máster. Universidad de La Laguna. https://riull.ull.es/xmlui/handle/915/31641

Tutorialkart. (s.f.). *What are the cluster manager supported in Apache Spark*. Recuperado el 23 de abril, 2024, de https://www.tutorialkart.com/apache-spark/cluster-managers-supported-in-apache-spark/#gsc.tab=0

Wikipedia. (2023). *Geometría del taxista*. Geometría del taxista - Wikipedia, la enciclopedia libre https://en.wikipedia.org/wiki/Taxicab_geometry

Xu, L., Mao, R., Zhang, C., Wang, Y., Zheng, X., Xue, X., & Xia, F. (2022). *Deep transfer learning model for semantic address matching*. Applied Sciences, 12(19), 10110. https://doi.org/10.3390/app121910110