TESIS FINAL DE GRADO

INGENIERÍA INFORMÁTICA

Modelos de Generación de Texto en Bases de Datos Orientadas a Grafos

Autor

Alberto López Povedano

Directores

Juan Francisco Huete Guadix

Ciencias de la computación e Inteligencia Artificial

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, noviembre de 2022

Modelos de Generación de Texto en Bases de Datos Orientadas a Grafos

Alberto López Povedano

Palabras clave: lenguaje natural, generación de texto, grafos, modelo del lenguaje, n-gramas

Resumen:
Desarrollo de un sistema que tiene como objetivo sugerir palabras, pequeñas frases y autocompletar palabras a un usuario escribiendo texto en castellano.
Para cumplir este objetivo se ha desarrollado un sistema que implementa modelado del lenguaje con n-gramas, nombrado GPG, construido sobre una base de datos orientada a grafos.
Para poder interactuar con el mismo se han habilitado una interfaz por terminal y una pequeño demo gráfica que ofrece sugerencias al usuario que escribe a tiempo real.

Text Generation Models in Graph-Oriented Databases

Alberto López Povedano

Keywords: natural language, text generation, graphs, language model, n-grams

Abstract:
Development of a system that aims to suggest words, short sentences and autocomplete words to a user typing text in Spanish.
To meet this objective, a system has been developed that implements language modeling with n-grams, named GPG, built on a graph oriented database.
In order to be able to interact with it, a terminal interface and a small graphical demo that offers suggestions to the user writing in real time have been enabled.

Yo, **Alberto López Povedano**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77448870G, autorizo la ubicación de la siguiente copia de mi Tesis Final de Grado en la biblioteca del centro para que pueda ser consultado por las personas que lo deseen.

Fdo: Alberto López Povedano

Granada a 08 de Noviembre de 2022

D. **Juan Francisco Huete Guadix**, Catedrático en el departamento de Ciencias de la Computación e Inteligencia Artificial en la Universidad de Granada.

Informan:

Que el presente trabajo, titulado **Modelos de generación de texto basados en bases de datos orientados a grafos**, ha sido realizado bajo su supervisión por **Alberto López Povedano**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 8 de noviembre de 2022.

Los directores:

Juan Francisco Huete Guadix

Agradecimientos

A Juan Francisco Huete Guadix por la supervisión en la realización de la tesis de final de grado y su inestimable ayuda aportando recursos didácticos, sugerencias y nuevos enfoques.

A Ernesto Martínez del Prieto por dirigirme para sentar las bases del sistema.

Índice

Figuras

Tablas

Términos

Application Programming Interface Punto de acceso por el cual dos programas informáticos pueden interactuar entre ellos.

Chatbot Sistema software cuyo propósito es simular conversaciones con usuario humano.

Corpus Colección de texto o audio oficial perteneciente a un lenguaje organizado en conjuntos de datos.

Gated Recurrent Unit Arquitectura de redes neuronales que presenta menos parámetros que la arquitectura LSTM.

Graphical User Interface Interfaz de usuario gráfica.

Long-Short Term Memory Arquitectura de redes neuronales basada en técnicas de aprendizaje profundo, que se caracterizan por tener conexiones de retroalimentación.

Object Oriented Programming Modelo de programación que organiza el diseño del software en torno a la idea de objeto, una entidad que contiene datos y código en forma de métodos.

Object-Relational Mapping Método para convertir los datos almacenados en una base de datos a un modelo definido en un lenguaje orientado a objetos.

Perplexity Métrica utilizada comúnmente para realizar evaluaciones de modelos del lenguaje basados en n-gramas.

Pipeline Cadena de procesos alienados de tal forma que la salida del último proceso es la entrada del siguiente proceso.

Plugin Extensión de un programa software que añade nueva funcionalidad sin alterar la proporcionada por el programa original.

Smoothing En el terreno estadístico hace referencia al hecho de crear una función que otorgue los considerados como patrones importantes en los datos a la vez que deja lo considerado como ruido fuera de ellos.

Sprint Ciclo de tiempo acotado por un equipo en el desarrollo del software siguiendo los principios de las metodologías ágiles.

Token Unidad mínima que puede representar hechos, emociones, etc… En este documento es utilizado para designar la unidad atómica que consideramos en el análisis del texto.

Tokenización Acción de convertir datos complejos en una o una colección de unidades individuales conocidos como tokens.

Abreviaciones

API Application Programming Interface.

CRF Conditional Random Fields.

GPG Grafo Preentrando Generativo.

GRU Gated Recurrent Unit.

GUI Graphical User Interface.

HMM Hidden Markov Models.

JSON JavaScript Object Notation.

LSTM Long-Short Term Memory.

NLP Natural Language Processing.

OOP Object Oriented Programming.

ORM Object-Relational Mapping.

RNN Recurrent Neural Network.

SVM Support Vector Machines.

1. Introducción

Los primeros sistemas conversaciones, según (Jurafsky & Martin, Chapter 24 - Bibliographical and Historical Notes, 2022), datan de 1966 y 1971 con los *chatbots* ELIZA y PARRY respectivamente. Ambos sistemas tuvieron cierta relevancia y plantearon nuevas cuestiones éticas en cuanto a la privacidad de datos de los usuarios, sin embargo, fue con el desarrollo de la arquitectura del sistema GUS (1977) que se establecería el paradigma predominante en el desarrollo de sistemas conversaciones durante los 30 próximos años.

Las aplicaciones del procesamiento del lenguaje natural por una computadora son increíblemente extensas, por mencionar alguna de ellas:

- Traducción automática de textos
- Clasificación de información
- Detección de estados emocionales
- Generación de texto
- Etcétera

Por sus grandes aplicaciones y debido al gran interés que despertó el terreno de la inteligencia artificial, el procesamiento del lenguaje natural se ha convertido en un campo de la inteligencia artificial con décadas de estudio a sus espaldas. Varias son las técnicas que se han aplicado en su investigación, hablamos de sistemas basados en reglas, aprendizaje automático mediante métodos estadísticos y redes neuronales.

En particular la modelización del lenguaje mediante la implementación de modelos basados en n-gramas, modelos probabilísticos, es la técnica utilizada en esta ocasión.

1.1. Propósito

El propósito de este trabajo es el de crear un software que nos sugiera recomendaciones, en forma de palabras u oraciones, en función de las palabras introducidas por el usuario. Este proyecto surge de la idea de crear un sistema que facilite la escritura de textos en un idioma distinto al del usuario, en concreto este idioma sería el español.

El objetivo del proyecto no es el de producir grandes cantidades de texto a raíz de entrada dada, sino más bien el de ofrecer pequeñas sugerencias, como dar la siguiente palabra, autocompletar palabras o poder ofrecer pequeñas frases a la hora de escribir. Algo similar a las sugerencias que podrían ofrecerte herramientas como el teclado de un dispositivo móvil, o las barras de búsqueda de los navegadores más comunes.

Esta idea fue propuesta por la empresa 8Belts, para obtener un sistema que pudiera desplegarse e integrarse en sus herramientas de enseñanza. Al día de publicarse, y durante gran parte del desarrollo no ha existido ningún enlace entre este trabajo y dicha empresa, por lo que la integración en sus herramientas no es un objetivo que poder conseguir. Aun así la base de datos usada se encuentra en un contenedor virtual

para facilitar su despliegue.

1.2. Justificación

Las razones que me han llevado a seleccionar este tema para el desarrollo de mi tesis de final de grado son:

- El interés y deseo de introducirse en el estudio de las técnicas de aprendizaje automático.
- Mejorar mi habilidad con el lenguaje de programación Python y la tecnología de contenedores Docker.
- Conocer y asentar las bases en el campo del procesamiento del lenguaje natural.

1.3. Objetivos

Teniendo como objetivo general de este proyecto aprender, reforzar, y poner en práctica los conocimientos más fundamentales de los modelos del lenguaje por métodos estadísticos, mediante la propuesta y desarrollo de un sistema básico de generación de texto, se plantean los siguientes objetivos:

1.3.1. Objetivos Primarios

1. Obtener una **fuente de datos** para el entramiento del modelo.
2. Implementar una forma de leer y almacenar los datos en una **base de datos** adecuada.
3. Representar los datos en una **estructura de datos** que nos permita representar las conexiones entre los diferentes caracteres o palabras.
4. Construir **dos modelos del lenguaje**. El primero enfocando los n-gramas a nivel de caracteres y el segundo cambiando el enfoque a nivel de palabras.
    1. El modelo a nivel de caracteres **estará compuesto por 3 submodelos**, a nivel de unigramas, bigramas, trigramas y cuatrigramas.
    2. El modelo a nivel de palabras **estará compuesto por 3 submodelos**, a nivel de unigramas, bigramas y trigramas.
5. Implementar **funcionalidades que nos permitan interactuar y realizar consultas** a nuestro modelo.
    1. Obtener **lista de tokens, ordenada por frecuencia**, que aparezcan a continuación de un token dado.
    2. **Sugerir un token** a continuación de una entrada dada.
    3. Poder **autocompletar palabras**.

4. Poder **formar frases sencillas**.

6. Habilitar la interacción directa con nuestro modelo mediante **una interfaz sencilla por terminal**.

1.3.2. Objetivos Secundarios

1. Configurar un **contenedor** que permita un despliegue rápido y sencillo de la base de datos.

2. Realizar una **evaluación** que nos aporte información de como de bien funciona nuestro modelo.

3. Escribir un **fichero *readme.md*** que aporte instrucciones para el despliegue y uso del proyecto.

4. Montar una **demo** mediante una GUI que ofrezca al usuario sugerencias a medida que este va escribiendo.

1.3.3. Requisitos no funcionales

Durante el proceso de desarrollo al igual que se consiguen los objetivos mencionados se han de tener los siguientes principios en mente:

1. Como en todo proyecto software se ha de **mantener una estructura de ficheros** y directorios que sea **clara y organizada**.

2. Los datos utilizados se obtendrán de **fuentes que aporten una licencia libre** para su uso.

3. El código ha de ser reutilizable y sencillo de comprender. La longitud de **comentarios en el texto ha de ser breve** y **resaltar aquello que no se puede extraer con facilidad** al leer el código.

4. El sistema ha de ser poder **desplegado con relativa facilidad en diferentes equipos haciendo pocas instalaciones de dependencias**, para facilitar tanto el desarrollo como para posibilitar el uso o desarrollo por diferentes usuarios.

1.4. Metodología

La metodología utilizada en el desarrollo del proyecto no ha respetado estrictamente las normas y formas de proceder de una metodología ya conocida, pero si se ha inspirado y aproximado al terreno de las conocidas como metodologías ágiles pues el proyecto se ha ido desarrollando de forma iterativa.

En este grupo encontramos principios como la organización del trabajo en torno a *sprints,* identificar el trabajo en pequeñas tareas lo más atómicas posibles otorgándoles una estimación de tiempo o el uso de herramientas como los tableros Kanban.
De las prácticas mencionadas anteriormente se ha llevado a cabo principalmente la definición de pequeñas tareas a realizar siendo representadas en tarjetas de un tablero

Kanban proporcionado por la plataforma Trello. A continuación, en la Figura 1.1 se aporta un ejemplo de cómo se vería el tablero Kanban utilizado:

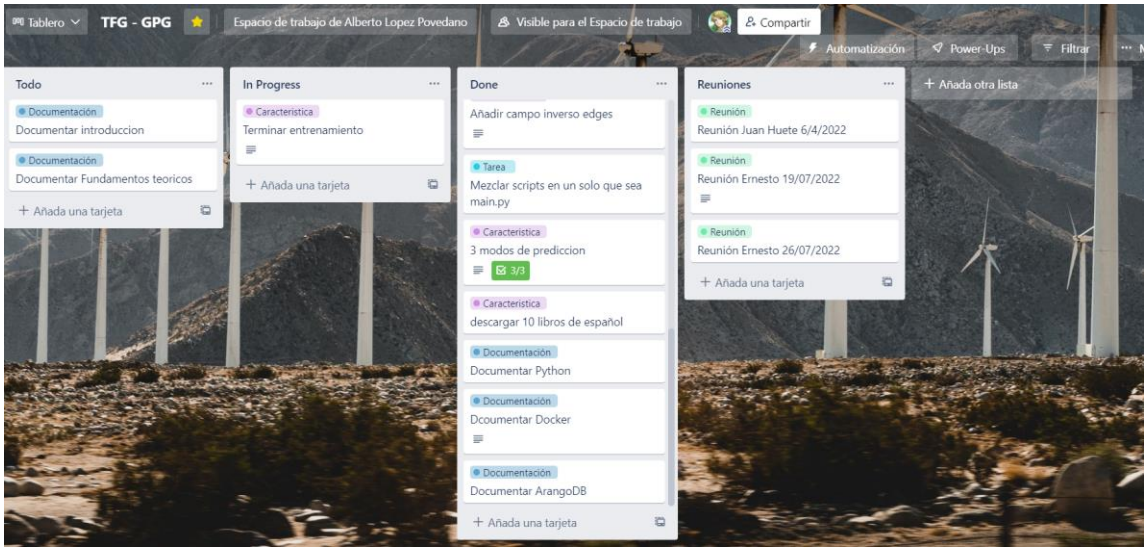

Figura 1.1: Tablero Kanban a día 14/08/2022

1.4.1. Terminología

Con el fin de evitar malentendidos y búsquedas confusas de términos usados en este documento se utilizarán términos en inglés, especialmente en los casos que la traducción de un término al castellano no produzca búsquedas satisfactorias o sea incluso confundido con acepciones relativas a otros campos del conocimiento. La terminología se utiliza, así, en castellano e inglés indistintamente. A la hora de utilizar términos en inglés, estos serán escritos utilizando cursiva.

En las primeras secciones no numeradas de este documento se han incluido un listado de Términos y otro de Abreviaciones, que exponen de forma breve los conceptos referenciados.

1.4.2. Licencia

Gracias a los conocimientos adquiridos en la asignatura: Dirección y gestión de proyectos. Se ha decidido lanzar el proyecto en un repositorio de GitHub público bajo la licencia GNU General Public License v3.0. Por lo tanto, se entiende que el código es *Free Software* según la definición otorgada por (Free Software Foundation, 2022).

Es importante resumir que esto implica la libre utilización del código, sin importar que sea un uso comercial, teniendo como condición el mantenimiento de la licencia utilizada en proyectos derivados a partir de este (*copyleft*).

1.5. Estado del arte

Según (Nadkarni, Ohno-Machado, & Chapman, 2011) el origen del procesamiento del lenguaje natural (NLP) data de los años 50 con el cruce entre la inteligencia artificial y la lingüística. Originalmente las aproximaciones a resolver problemas relacionados con

el lenguaje **eran muy simplistas** como diccionarios que traducían palabra a palabra entre diferentes idiomas.

Con el tiempo se fue entreviendo la dificultad que suponían los problemas relacionados con el NLP. Poco a poco se fue avanzando en este campo, desarrollo de la sintaxis de expresiones regulares (1956) y su implementación en UNIX o el uso en conjunto de analizadores léxicos y sintácticos incentivaron la creación (1970s) de los lenguajes de alto nivel que conocemos actualmente. A partir de los 80 métodos de aprendizaje automático basados en probabilidades se asentaron como los predominantes, dando lugar a un enfoque estadístico del NLP.

Este enfoque estadístico implicaba el nacimiento de modelos del lenguaje que necesitaban grandes conjuntos de datos para ser funcionales, en cambio el problema que esto presentaba es que dichos modelos son dependientes del contexto dado por el *corpus* de texto, haciendo que modelos entrenados con texto de periódicos no sean apropiados para problemas, por ejemplo, en campos médicos. Estos **modelos estadísticos** pueden ser construidos utilizando distintas aproximaciones y algoritmos, como son las máquinas de vectores de soporte (SVMs), los modelos escondidos de Markov (HMMs), los campos aleatorios condicionales (CRFs) y los conocidos como modelos de n-gramas.

En particular los modelos de n-gramas bien entrenados funcionan bien y son normalmente aplicados para tareas de:

- Sugerir como autocompletar palabras o frases.
- Corrección de fallos de escritura
- Identificación de texto

Estos últimos años, con las innovaciones en el campo de la inteligencia artificial, estamos observando una deriva a la utilización de **redes neuronales** para resolver problemas en este campo.

Como se menciona en (Casas, Mugellini, & Abou Khaled, 2020) las redes neuronales recurrentes (RNNs) han sido modelos muy usados, bajo las arquitecturas LSTM y GRU, durante finales de los 2000 y principios de 2010. En 2017 se presenta una nueva arquitectura para las redes neuronales conocida como *Transformer,* que ha tenido un impacto muy significativo en la modelización del lenguaje, con la creación de grandes modelos del lenguaje como el detallado en la Sección 1.5.1. Esta última arquitectura no se ve afectada por problemas a la hora de paralelizar múltiples procesos o de tratar con entradas de texto muy largas, en cambio LSTM y GRU si enfrentaban dichos problemas.

En nuestro proyecto para los objetivos que queremos completar haremos uso de los modelos de n-gramas. Por varias razones:

- Satisface los objetivos marcados
- Menos complejidad que otros modelos
- Aunque ya no existe relación con 8-Belts, inicialmente se descartó el uso de herramientas como GPT-3 pues implicarían un coste por su uso.

1.5.1. GPT-3

GPT-3, abreviación de *Generative Pre-trained Transformer 3,* es un modelo del

lenguaje perteneciente a la serie de modelos GPT desarrollados, bajo la arquitectura *transformer*, por la empresa OpenAI (About OpenAI, 2022).

Actualmente este modelo puede ser utilizado por el público, por un diferente rango de precios, desde su API habilitada. El modelo está compuesto por otros 4 submodelos *Davinci, Curie, Babbage* y *Ada,* siendo Davinci el más potente y caro, pero más lento y Ada el menos potente, pero el más rápido y barato. Cada modelo puede hacer una serie de tareas diferente, pero todo modelo más potente podrá realizar todas las funcionalidades del modelo por debajo suya, es decir, *Davinci* es capaz de realizar todas las tareas que los modelos *Curie, Babbage* y *Ada* pueden hacer.

GPT-3 es capaz de crear artículos de texto a raíz de una pequeña entrada de texto. Es capaz de redactar cartas, artículos, diálogos e incluso código dado una entrada que le otorgue ciertas instrucciones como el contexto o el tipo de texto a redactar.

Desde su misma API (Examples - OpenAI API, 2022) se ofrecen una serie de ejemplos de todas las posibles tareas o aplicaciones que este modelo podría realizar.

2. Fundamentos Teóricos

En este capítulo se exponen los conceptos teóricos necesarios para poder llevar a cabo el desarrollo del sistema pretendido. Para poder llevarlo a cabo necesitamos entender herramientas que nos permitan poder analizar y asignar buenas probabilidades a sentencias bien formadas del lenguaje natural, con la idea de poder generar texto, como es el caso del modelado del lenguaje. También se ha de conocer mediante que estructuras de datos podemos construir dicho modelo.

2.1. Modelado del lenguaje

El modelado del lenguaje es un término que engloba todos aquellos métodos que intentan solucionar el problema de conseguir un modelo del lenguaje natural. El modelado del lenguaje tiene como objetivo permitir el aprendizaje de la máquina, mediante el uso de técnicas estadísticas, sobre el lenguaje para poder producir aplicaciones que resuelvan problemas relacionados con el NLP.

Existen diferentes aproximaciones para construir un modelo del lenguaje, en concreto los modelos que funcionan mediante la asignación de altas probabilidades a frases bien estructuradas y formadas han de tener en cuenta la asunción de Márkov.

De primeras sin tener en cuenta la asunción de Márkov, para calcular la probabilidad de aparición de una frase $p(w_1,\cdots w_k)$ se desglosaría utilizando la regla de la cadena en:

$$p(w_1,\cdots,w_k) = p(w_1) * p(w_2|w_1) * p(w_3|w_1,w_2) * \cdots * P(w_k|w_1,\cdots,w_{k-1})$$

Como se puede observar este cálculo es bastante laborioso, pero según (Gudivada, Rao Vijay, & Raghavan, 2015) la asunción de Márkov nos permite, aplicado en el contexto de modelos del lenguaje, presumir que la probabilidad de la siguiente palabra en una secuencia depende únicamente de la palabra previa o secuencia n de palabras previas, siendo n un número pequeño de palabras. El tamaño de n indica el n-orden del proceso de Márkov. Simplificando la ecuación anterior:

$$p(w_1,\cdots,w_k) \approx \prod_{i=1}^{k} p(w_i \mid w_{i-(n-1)} \cdots w_{i-1}) \; 1 \leq n < i$$

Entendiendo que $p(w_i \mid w_{i-(n-1)} \cdots w_{i-1})$ hace referencia a la probabilidad de ver w_i tras la secuencia de palabras $w_{i-(n-1)} \cdots w_{i-1}$

Dentro del modelado del lenguaje se han desarrollado varios modelos diferentes con sus propias ventajas y desventajas. Encontramos modelos exponenciales, modelos de n-gramas y modelos del lenguaje neuronales. Como se puede asumir tras la explicación del proceso de Márkov, los modelos de n-gramas son en los que se va a centrar el proyecto.

2.1.1. Modelos de N-gramas

Los modelos de n-gramas puede ser definidos como una cadena de Márkov de orden

n-1. En ellos se trabaja con una distribución de probabilidad sobre las secuencias de palabras o caracteres. Estos modelos se caracterizan por trabajar a base de n-gramas. Los n-gramas son conjuntos de datos formados por n palabras, caracteres o sílabas, esto depende en la *tokenización* elegida a la hora de analizar el *corpus* de texto.

Dependiendo de la *tokenización* elegida se pueden clasificar diferentes tipos de modelos. Lo más conocidos son los que dividen y analizan el *corpus* a nivel de palabras haciendo uso de espacios y signos de puntuación como separadores. Diferenciamos entre unigramas, bigramas, trigramas, … en función del número de *tokens* que formen el n-grama. Un ejemplo de lo explicado, considerando como *token* una palabra, se puede visualizar en la Figura 2.1:

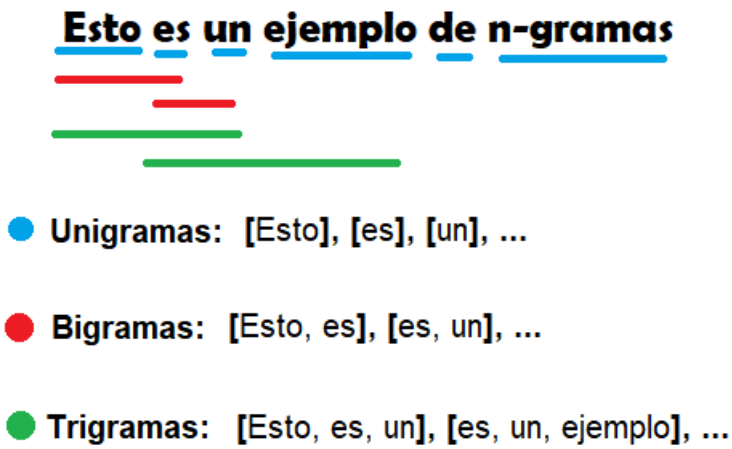

Figura 2.1: Formación de n-gramas usando como token las palabras

Para el desarrollo de este proyecto se ha construido dos modelos del lenguaje a nivel de palabras y a nivel de caracteres. Cada uno de estos cuenta con otros 3 submodelos que analizan los tokens a nivel de unigramas, bigramas y trigramas.

2.1.1.1. Modelos de N-gramas a nivel de caracteres

Los modelos basados en n-caracteres presentan pocas aplicaciones como se expone en (Russell & Norvig, N-gram character models, 2009), principalmente destacan en la identificación del lenguaje de un texto escrito, aunque también son usados para detectar fallos ortográficos. En este caso se está utilizando un modelo de caracteres para poder ofrecer sugerencias que completen palabras a medio escribir.

Es importante tener en cuenta que el conjunto de unigramas, bigramas y trigramas posibles a nivel de caracteres en el lenguaje español es mucho menor a la explosión combinatoria resultante al analizar n-gramas a nivel de palabras. Esto se debe a que los vocabularios que analizamos en ambos casos son en tamaño muy diferentes. Para visualizar esto pensemos en que la cota superior de posibles n-gramas por nivel puede ser calculada como x^n, siendo x el vocabulario del lenguaje a analizar y n el nivel de n-grama. Estos números quedan reducidos por la imposibilidad de algunas combinaciones de letras o palabras en el lenguaje.

2.1.1.2. Modelos de N-gramas a nivel de palabras

Los modelos de n-gramas de palabras afrontan el problema conocido como *data sparsity* al presentar vocabularios muchos más extensos. Este problema aparece dada la explosión combinatoria que se produce al querer tener un mayor tamaño en los n-gramas construidos, pues al presentar mayores niveles más contexto podremos tener en cuenta y por lo tanto nuestras respuestas tendrán mayor parecido con la realidad.

El problema de intentar hacer los n-gramas de gran tamaño es que cuanto mayor es el tamaño más posibilidad hay de que aparezcan n-gramas que no hayan sido observados en el conjunto de datos utilizado para el entrenamiento, también se considera que a partir de los trigramas, es decir, haciendo uso de tetragramas nuestro modelo memoriza en vez de generar texto. Para enfrentar este problema se hace uso comúnmente de técnicas de *smoothing*, y no se suele analizar n-gramas con un orden mayor a $n = 3$

(Russell & Norvig, N-gram word models, 2009)

También se pueden encontrar los modelos de lenguaje neuronales como una solución al problema mencionado anteriormente. Pues con el modelo expuesto en (Vincent, 2003) se introdujo la idea de que las redes neuronales son capaces de aprender representaciones similares de palabras dentro de un contexto, otorgando así respuestas a n-gramas que no se hayan observado en el *corpus* de entrenamiento usado.

2.2. Grafos

Los n-gramas y las conexiones entre ellos pueden ser almacenados y visualizados como un grafo. Un grafo es una estructura constituida por nodos y arcos entre esos nodos. El ámbito de estudio de los grafos es conocido como teoría de Grafos, la cual es una rama de la matemáticas y ciencias de la computación.

En este caso las palabras serían representadas por los nodos y los arcos establecerían relaciones de sucesión entre ellas, es decir, imaginemos el nodo que simboliza a la palabra “por” dicho nodo podría presentar arcos que lo relacionaría con las palabras “lo” y “ejemplo” tras haber analizado las frases: “… por lo tanto esta comida está fría” y “por ejemplo, aquí tienes mi estuche”, pues como se puede ver la palabra “lo” y “ejemplo” suceden a la palabra “por”. La Figura 2.2 ilustra el ejemplo descrito, además nótese como los arcos reflejan conexiones unidireccionales, si estudiáramos una sentencia donde “lo” precediera a “por” entonces el arco entre ellas pasaría a reflejar una conexión bidireccional.

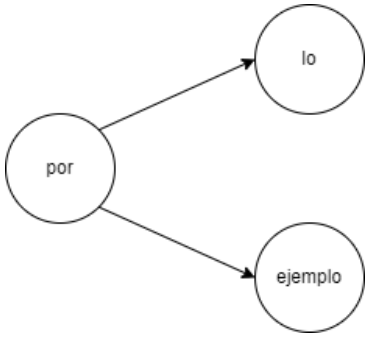

Figura 2.2: Ejemplo de grafo de unigramas a nivel de palabras

La representación de las conexiones entre n-gramas puede ser realizada por varias estructuras de datos diferentes, matrices de vectores, vectores de instancias de objetos, etcétera. La razón por la cual se ha escogido el grafo como estructura para representar dichas conexiones es dado a la facilidad que presenta para recorrerlo, sobre todo teniendo en cuenta que la base de datos elegida para el proyecto es una base de datos orientada a grafos.

3. Conjunto de datos

En este capítulo se explica el conjunto de datos utilizados tanto para el entramiento como para la evaluación realizada. Se expone donde se ha obtenido los datos y que consideraciones se realizan a la hora de analizarlos.

3.1. Fuente de los datos

Actualmente existen muy diversas fuentes de datos que pueden ser utilizadas para entrenar este tipo de modelos. En este caso por facilidad de lectura de formato se ha decidido elegir libros electrónicos escritos en la extensión de archivo TXT, es decir, los libros se encuentran en texto plano, además, presentan codificación de caracteres ANSI.

Los libros, todos en castellano, se han obtenido de la plataforma web del Proyecto Gutenberg (The Project Gutenberg Literary Archive Foundation, s.f.), disponen de aproximadamente 60000 libros en diferentes idiomas de los cuales la mayoría pertenecen a dominio público, por lo tanto, las licencias de uso no suponen un problema.

Se ha divido el conjunto de datos, en un conjunto de entrenamiento que supone aproximadamente un 80% del *corpus* total y un conjunto de prueba o test que ocupa el 20% restante.

A continuación, se enumeran los libros utilizados en el **conjunto de entramiento**:

1. Al primer vuelo (Pereda, Al primer vuelo, 1986)
2. Algo de todo (Valera, 1883)
3. Amar es vencer (Caro, 2008)
4. Amistad funesta: Novela (Martí, 1885)
5. Amparo (Memorias de un loco) (Fernández y González, 1858)
6. Ariel (Rodó, 1900)
7. Arroz y tartana (Blasco Ibánez, 1894)
8. Cádiz (Pérez Galdós, 1874)
9. Cartas de mi molino (Daudet & Cabañas, 1869)
10. Don Quijote de la Mancha (de Cervantes Saavedra, 1605)
11. Novelas Ejemplares (Cervantes Saavedra, 1613)
12. Viajes por España (de Alarcón y Ariza, Viajes por españa, 1883)
13. El niño de la bola: Novela (de Alarcón y Ariza, El niño de la bola, 1880)
14. La Regenta (Leopoldo Alas, La Regenta, 1884)

Lista de libros utilizados para el **conjunto de prueba:**

1. La vida de Lazarillo de Tormes y de sus fortunas y adversidades (Anónimo, 1554)
2. Su único hijo (Leopoldo Alas, Su único hijo, 1890)
3. Crónicas de Marianela (Palma Román, 1917)
4. La quinta de Palmyra (Gómez de la Serna, 1944)
5. La Montálvez (Pereda, La Montálvez, 1888)

Para saber la importancia del corpus utilizado a continuación se ofrecen una serie de tablas sobre el conjunto de entrenamiento y de test que ofrecen el número de palabras y caracteres por libro usado en cada uno.

Libro	N.º caracteres (sin espacios)	N.º palabras
1	463807	102184
2	266908	56741
3	254249	54965
4	279164	61413
5	131144	28590
6	130052	25708
7	445047	91430
8	366500	78286
9	169742	34966
10	1718347	381104
11	838985	189412
12	337952	68386
13	359302	75073
14	1496535	310692
Total	7257734	1558827

Tabla 3.1: Número de palabras y caracteres del conjunto de entrenamiento

Libro	N.º caracteres (sin espacios)	N.º palabras
1	85920	20063
2	425275	89836
3	251517	51827
4	212632	45551
5	494428	106976
Total	1469772	314253

Tabla 3.2: Número de palabras y caracteres del conjunto de prueba

3.2. Limpieza de los datos

Se han tenido que tomar varias consideraciones a la hora de tratar los datos para proporcionar la información más relevante y menos conflictiva al modelo de lenguaje. A continuación, se listan todas las decisiones tomadas:

- En todos los libros mencionados se puede encontrar en las primeras líneas y últimas líneas unos párrafos en inglés conteniendo metadatos del libro en sí y

además indicando los términos y condiciones de la licencia de uso del libro. Se decidide eliminar, directamente del archivo, dichas líneas pues contienen vocabulario en inglés, además de presentarse los metadatos en un formato no natural y poco parecido al lenguaje humano.

- Se ha optado por transformar todas las palabras presentes en los libros a minúscula, es decir, ‘Hola’ y ‘hola’ serían consideradas la misma palabra.

- Los signos de exclamación en interrogación son ignorados y no se tienen en cuenta a la hora de formar una palabra e identificar las palabras que le suceden, es decir, en la oración ‘¿Qué tal estás?’ se detectarán correctamente las palabras [qué], [tal], [estás] y sus correspondientes conexiones.

- Se respetan acentuaciones de las palabras, por lo tanto, ‘que’ y ‘qué’ son palabras distintas. Por esta misma razón se puede diferenciar entre ‘qué’ interrogativo o exclamativo y ‘que’ como conjunción o pronombre relativo.

- En cuanto a signos ortográficos menos comunes, tales como ‘<<’, ‘>>’, ‘-‘, ’_’, etcétera, son eliminados y no interfieren en la sucesión o detección de palabras.

- El final de una frase se indica por un punto ‘.’, por lo cual el texto de un libro es divido en sentencias. De esta forma no se contabilizan relaciones de continuidad entre la última palabra de una frase y la primera de la siguiente.

- En varios de los libros se ha observado el uso de la preposición “a”, “o”, “u” pero acentuándose, es decir: “á”, “ó”, “ú”. Esto se debe a un uso anticuado, se ha corregido en todo texto usado.

Se pueden consultar los detalles de implementación a código que satisfacen estos requisitos en la Sección 4.2.1

4. Diseño del sistema

Se recuerda que el nombre del sistema GPG, abreviación de grafo preentrenado generativo y en este capítulo se exponen los conceptos de diseño por los cuales se le ha decidido otorgar dicho nombre.

Una idea fundamental para entender GPG es porque está formado. Como se detalla en la Figura 4.1, GPG está formado por dos modelos de n-gramas diferentes. En uno de ellos se *tokeniza* y se trabaja a nivel de palabras, en el otro el *token* se considera a nivel de carácter. Cada modelo principal a su vez está compuesto por 3 submodelos diferentes, cada submodelo opera a nivel de unigramas, bigramas o trigramas.

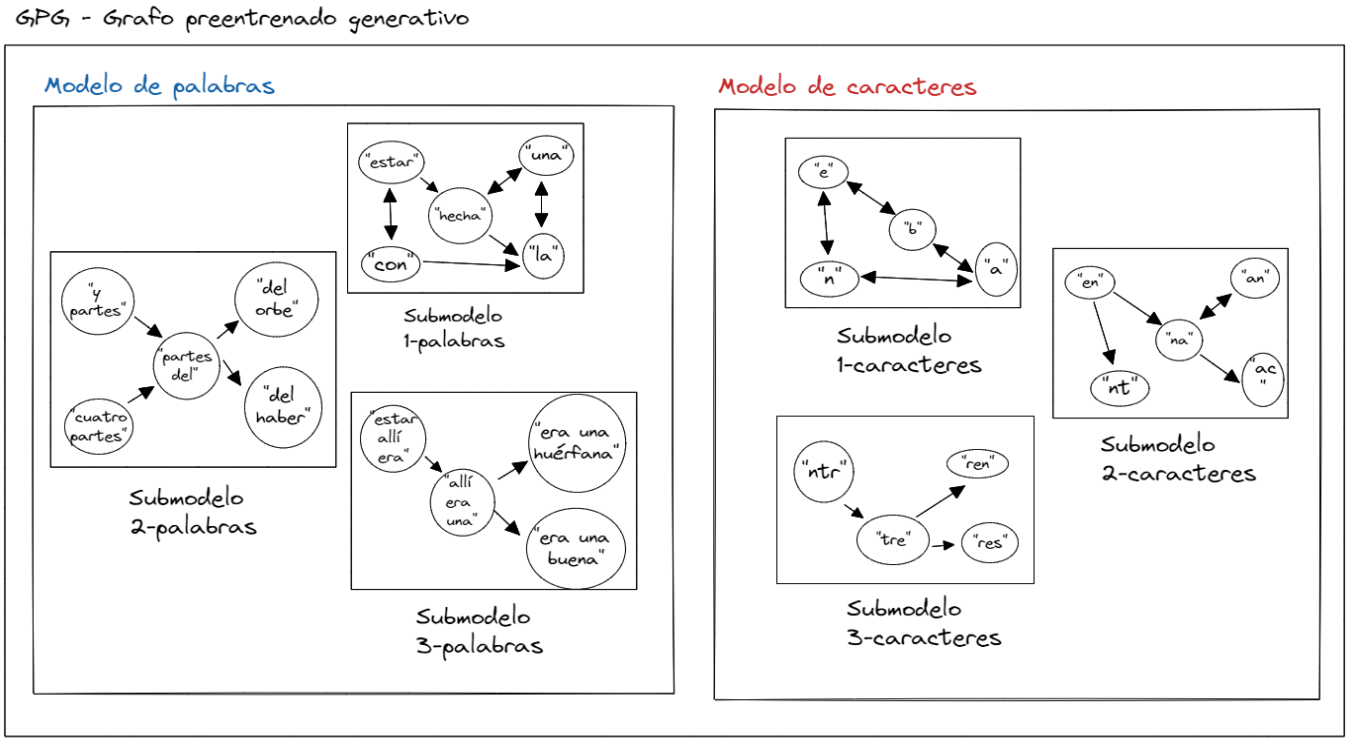

Figura 4.1: Composición de GPG

Antes de introducirse a implementar el sistema ha de asentarse conceptos de diseño previos. Saber identificar el ciclo de ejecución deseado para nuestro sistema y como el usuario podrá interactuar con el mismo, además de visualizar el proceso de extracción, transformación y almacenamiento de los datos.

Como se expone en siguientes apartados se pueden identificar pequeños procedimientos que componen el ciclo de ejecución total del programa, procedimientos que generalmente se repiten y que pueden ser encapsulados en trozos de código conocidos como funciones. Teniendo en cuenta las razones anteriores para implementar se decide alejarse del paradigma orientado a objetos y centrarse en un paradigma de programación funcional.

4.1. Flujo de control

El ciclo de vida del sistema es sencillo, este puede ser visto de una forma jerárquica empezando por una vista global como la expuesta en Figura *4.2*.

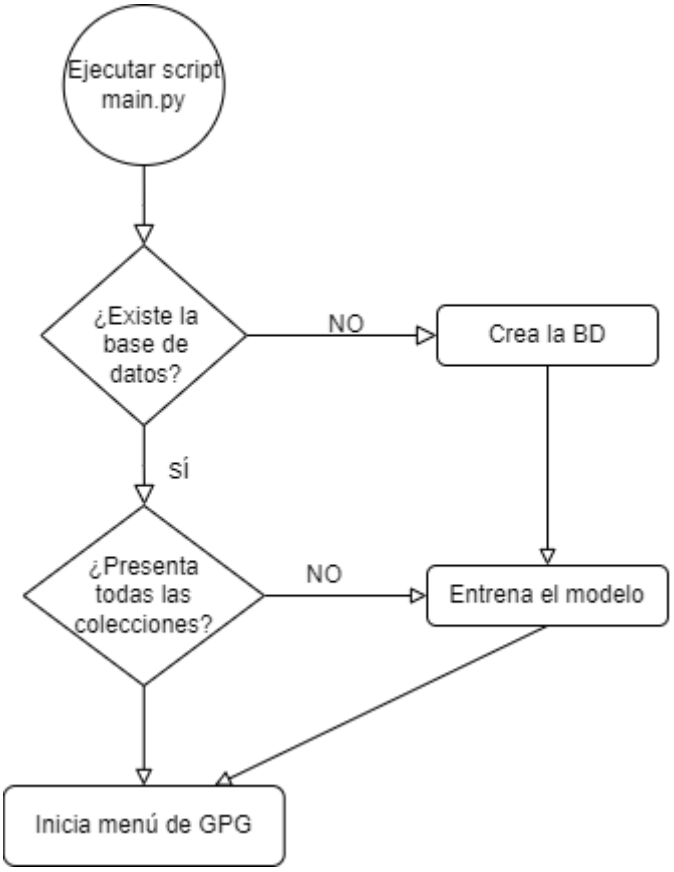

Figura 4.2: Flujo de control global de GPG

Como se puede observar en el mayor nivel de abstracción el flujo de control de GPG es bastante simple. A continuación, se exponen otros diagramas donde se desgranan los procesos complejos que en esta primera vista han sido simplificados.

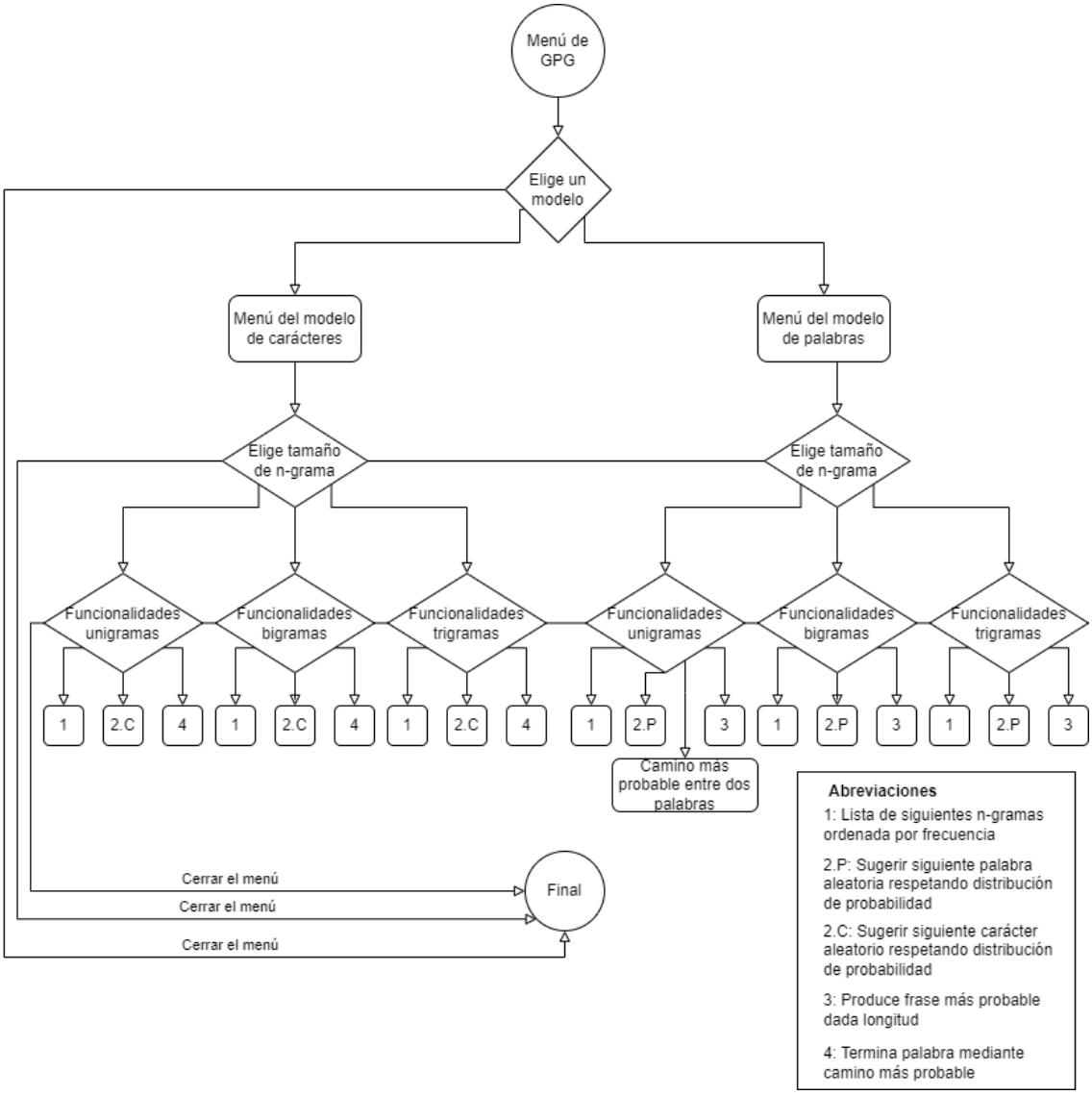

Figura 4.3: Flujo de control del menú de GPG

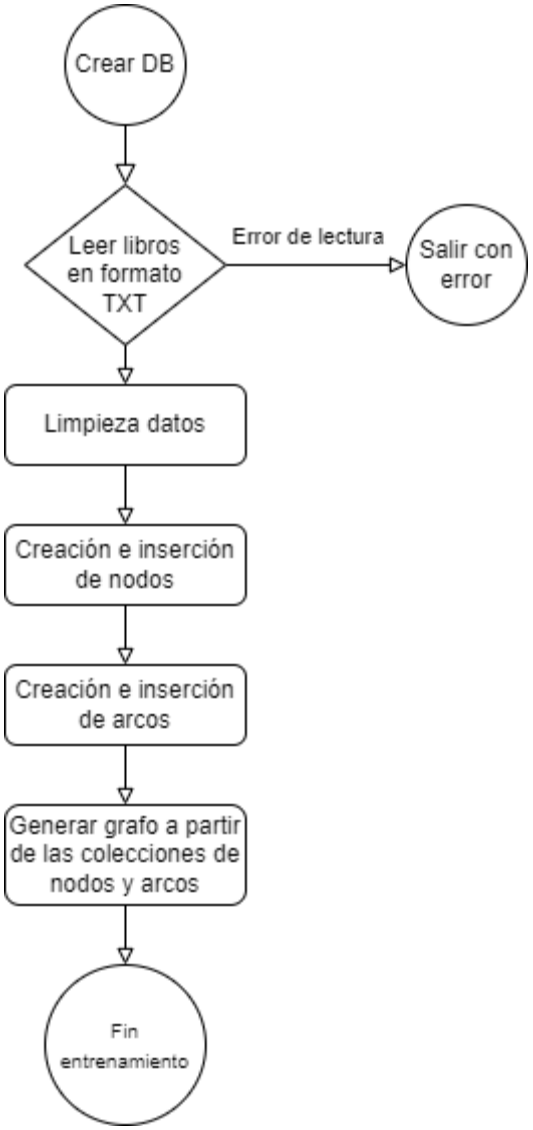

Figura 4.4: Flujo de control del entrenamiento del modelo

4.2. Flujo de datos

Los datos que encontramos en los libros mencionados en la Sección 3.1 han de pasar por procesos de extracción y transformación hasta que finalmente puedan ser insertados en la base de datos ArangoDB.

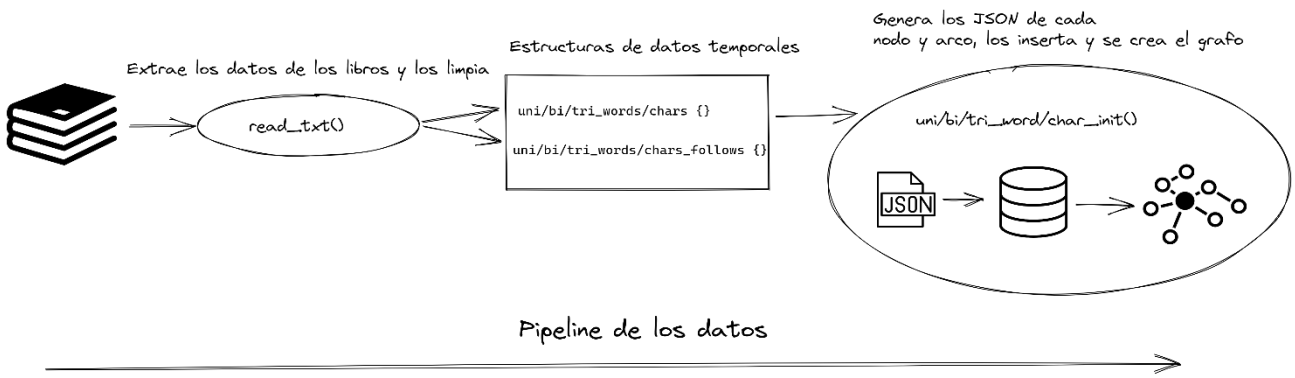

Figura 4.5: Diagrama del flujo de datos

Como se deja expuesto en la Figura 4.5 el conjunto de datos sigue una lista de pasos, normalmente conocida como *pipeline,* hasta finalmente poder construir los grafos que utilizaran cada uno de los submodelos de GPG. Seguidamente se entra en más detalle de las estructuras de datos producidas y los procesos involucrados.

4.2.1. Estructuras de datos temporales

La función *read_txt(array_libros: [])* espera como entrada una lista compuesta por los caminos absolutos de cada uno de los libros usados para el entrenamiento. En este proceso se realizan los detalles de limpieza especificados en la Sección 3.2.

- El texto de un libro es separado en frases mediante los puntos con la función *split*.
- Mediante la utilización de la expresión regular **'\b\w+\b'** se detectan las palabras en una cadena de texto, esta ignora la mayoría de los signos ortográficos a excepción de '_' que es remplazado mediante la función *replace*.
- Toda letra mayúscula es convertida a minúscula mediante la función *lower.*
- Cuando se detecta la palabra 'á', 'ó', 'ú' (antiguo uso del castellano) estas son remplazadas por sus homónimas sin acentuación.

La lectura del texto genera como salida estructuras de datos temporales. Dichas estructuras son dos diccionarios por cada submodelo perteneciente a GPG, en total 6 submodelos. Los dos tipos de diccionarios generados para cada submodelo contienen información sobre que n-gramas aparecen, cuántas veces aparecen y las relaciones de sucesión entre los diferentes n-gramas. En la Figura 4.6 se refleja la estructura de los diccionarios producidos.

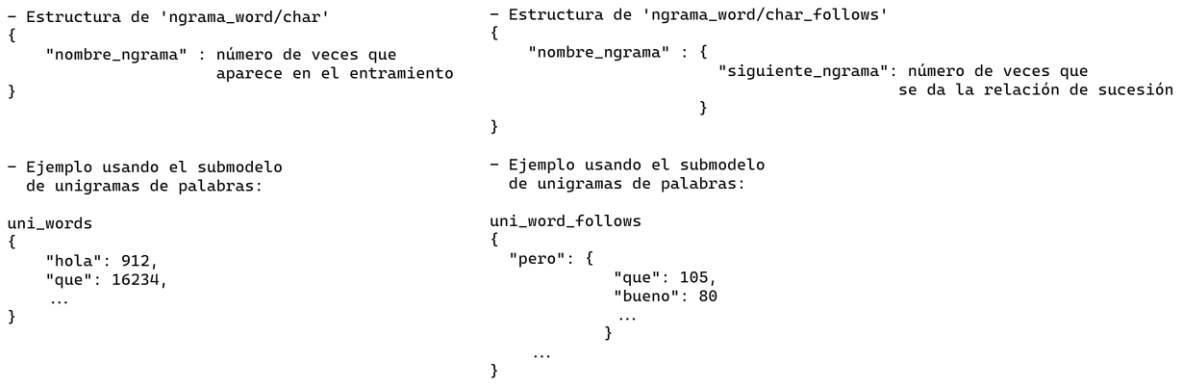

Figura 4.6: Diccionarios temporales tras lectura del conjunto de entramiento

4.2.2. Archivos JSON para nodos y arcos

Las estructuras temporales mencionadas en el apartado anterior son utilizadas en cada uno de métodos para poblar la base de datos. Cada submodelo tiene su función 'ngram_word/char_init' que utiliza la correspondiente estructura temporal generada anteriormente.

Estos métodos comprueban previamente que no existen las colecciones en la base de datos, con la intención de que si el proceso se para en alguna insertando ficheros en alguna colección no se tenga que repetir desde el principio.

ArangoDB solo permite la inserción de datos en formato JSON. Cada nodo y cada arco de un grafo está representado por un documento JSON que es guardado en su respectiva colección. Las colecciones son un conjunto de nodos o arcos, habrá una colección de nodos y otra de arcos por cada submodelo de GPG. Los nodos tienen un campo obligatorio llamado "_key" que sirve como clave principal, en cambio los arcos contienen tres campos obligatorios "_key", "_from" y "_to" estos dos últimos identifican inequívocamente a los dos nodos entre los que existe ese arco.

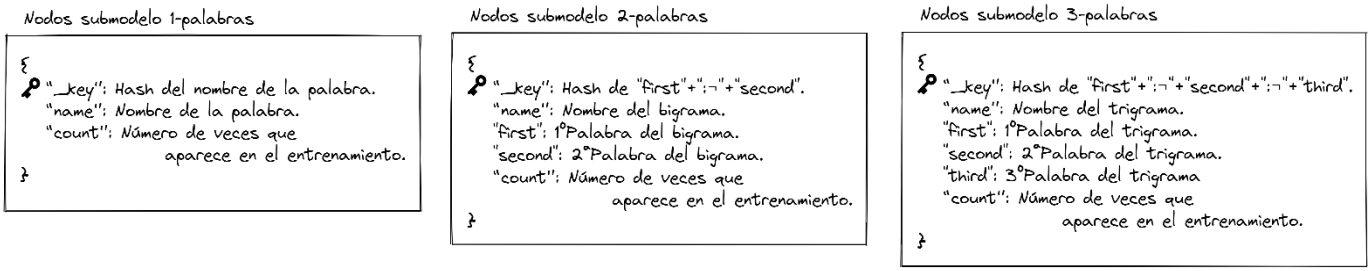

Figura 4.7: Estructura de los documentos relativos a los nodos del modelo de palabras

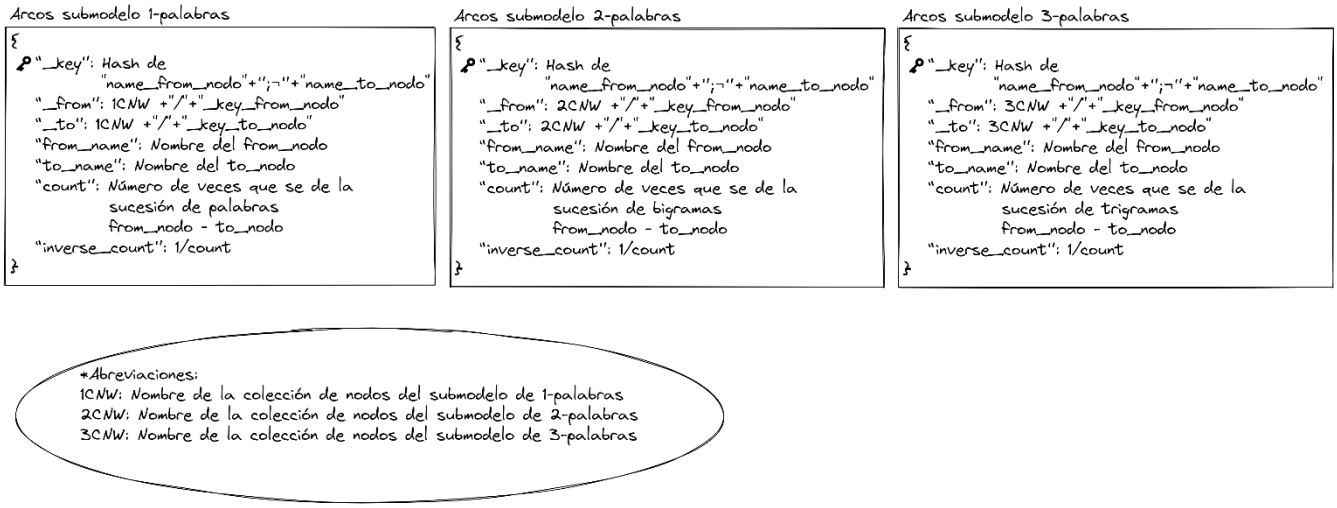

Figura 4.8: Estructura de los documentos relativos a los arcos del modelo de palabras

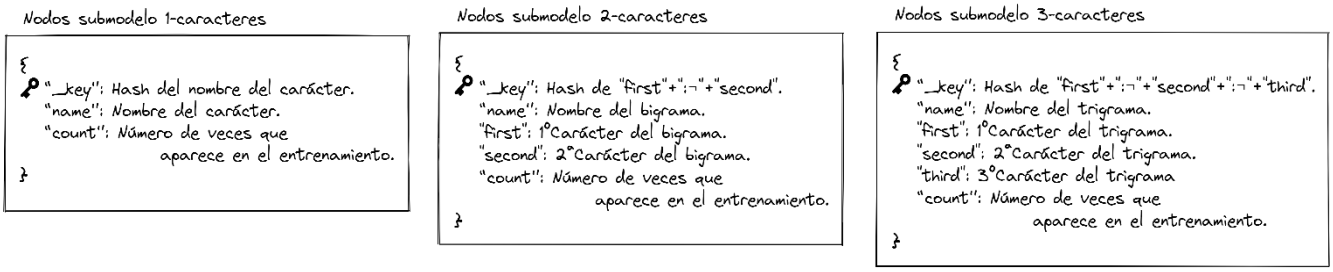

Figura 4.9: Estructura de los documentos relativos a los nodos del modelo de caracteres

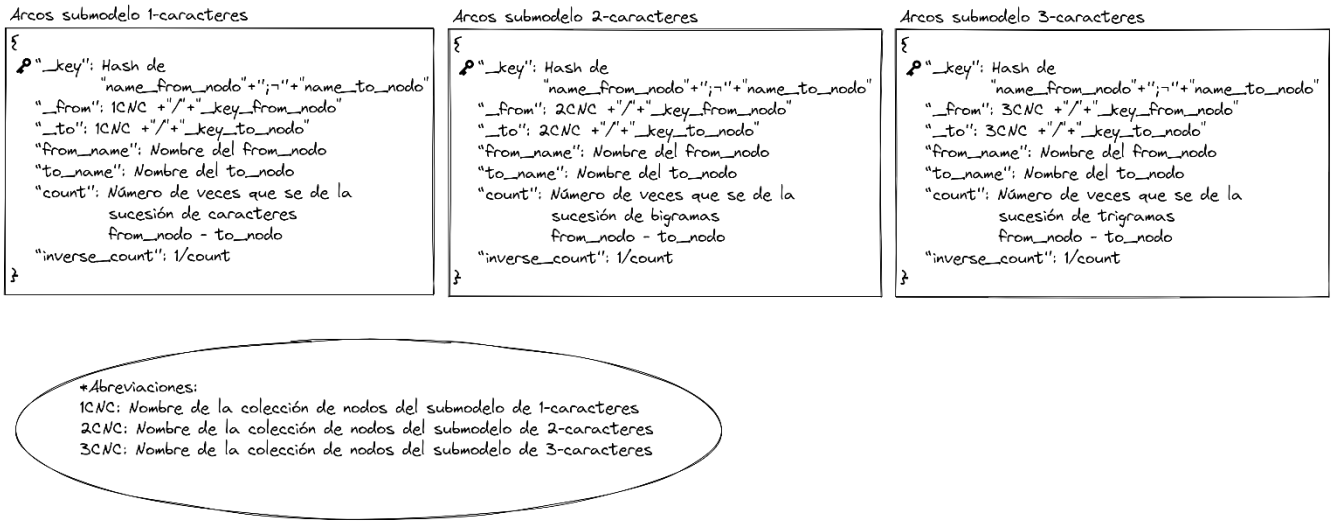

Figura 4.10: Estructura de los documentos relativos a los arcos del modelo de caracteres

5. Implementación del sistema

En este capítulo se exponen los detalles relacionados con la implementación de GPG, como la estructura de directorios, las dependencias entre módulos, los grafos obtenidos y las funcionalidades que presenta cada submodelo.

5.1. Estructura de directorios

El proyecto está organizado siguiendo la estructura de directorios siguiente.

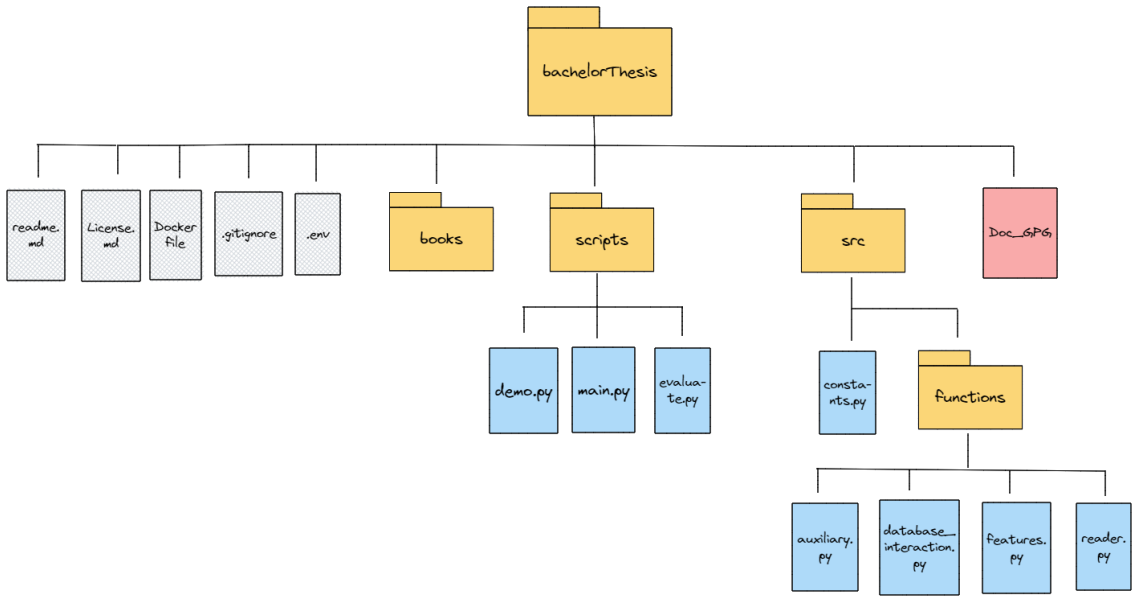

Figura 5.1: Estructura de directorios del proyecto

Para tener en mente la modularización del código y su posible reusabilidad se ha diseñado un diagrama de paquetes, Figura 5.2, que muestra las dependencias entre los distintos módulos que constituyen al proyecto.

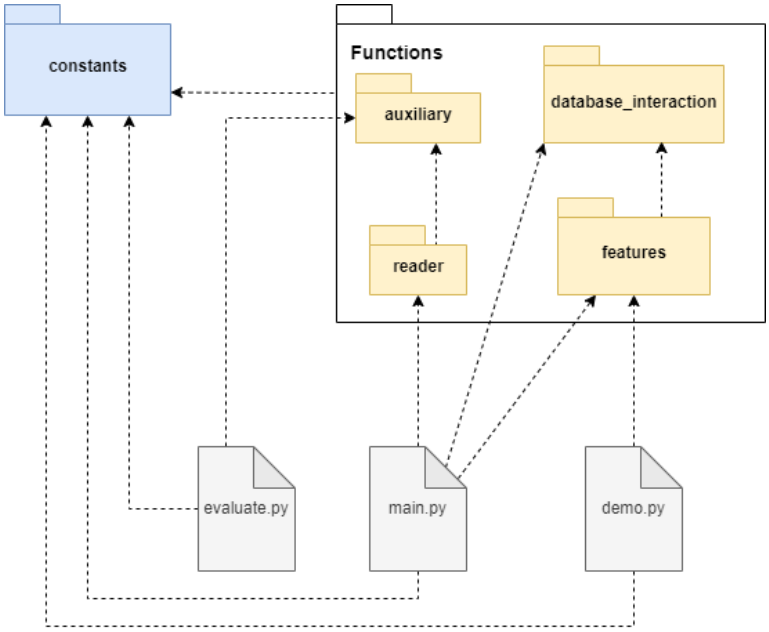

Figura 5.2: Diagrama de paquetes del proyecto

5.2. Grafos de los submodelos

Cada submodelo funciona encima de un grafo. Cada grafo se construye utilizando una colección de nodos y una colección de arcos. En este apartado se aportan estadísticas del número de vértices y aristas que presenta cada grafo.

Grafo	N.º nodos	N.º arcos
Submodelo 1-palabras	65248	532269
Submodelo 2-palabras	532269	1087686
Submodelo 3-palabras	1087686	1300274
Submodelo 1-caracteres	62	988
Submodelo 2-caracteres	988	7768
Submodelo 3-caracteres	7768	32542

Tabla 5.1: Número de nodos y arcos presentes en cada grafo

Se puede apreciar como el número de arcos de cada submodelo coincide con el número de nodos del modelo siguiente. Esto tiene sentido pues la relación existente entre dos unigramas, por ejemplo, se puede visualizar como un bigrama.

Relativo al submodelo de 1-caracteres el hecho de que su vocabulario esté compuesto por 62 caracteres puede parecer excesivo. Ha de tenerse en cuenta que el abecedario español está formado por 27 caracteres, más las vocales acentuadas y la “ü” con diéresis, aparte también se cuentan los dígitos numéricos. En total acabamos con 43 caracteres, a los cuales se añaden dos tokens especiales “<>” y “</>” que sirven para indicar cuando comienza o termina una palabra, esto caracteres nos permiten identificar caminos más probables para saber cuándo terminar una palabra. Por último, los restantes son caracteres pertenecientes a otros lenguajes, especialmente el francés, ya que al analizar novelas en varías de estas aparecen nombres propios o palabras en otro idioma, pero esto no supone un gran inconveniente pues su

frecuencia de aparición es tan mínima que no afectan a las sugerencias.

5.3. Funcionalidades

Se habilitan funcionalidades que permiten recorrer y utilizar los grafos creados pertenecientes a cada submodelo. Estas funcionalidades han sido desarrolladas teniendo en cuenta poder satisfacer los objetivos principales que se marcaron para la implementación del sistema. Todas estas funcionalidades pueden ser probadas mediante la interacción por terminal con el sistema.

5.3.1. Modelo de palabras

Este modelo está compuesto por 3 submodelos unigrama, bigrama y trigrama de palabras. Cada submodelo contiene estas funcionalidades comunes:

- **Lista de n-gramas:** Ante un n-grama y grafo especificado se busca la existencia de ese n-grama en el grafo. Identificado el n-grama se obtienen todos aquellos nodos adyacentes y son devueltos en una lista ordenada según el número de veces que se dé la sucesión entre el nodo original y el nodo adyacente. Si no se encuentra el n-grama en el grafo se devuelve una lista vacía.

- **Sugerir siguiente palabra:** Ante un n-grama y grafo especificado se busca la existencia de ese n-grama en el grafo. Identificado el n-grama se analiza el arco con todo nodo adyacente, se genera una distribución de probabilidad discreta respetando la frecuencia de sucesión entre el nodo original y cada nodo adyacente. Finalmente usando esta distribución a modo de bolsa de palabras se selecciona “aleatoriamente” la siguiente palabra a sugerir. Si no se encuentra el n-grama en el grafo se devuelve una cadena vacía “”.

- **Genera una frase:** Dado un n-grama que comience la frase, un grafo especificado y el tamaño máximo de la frase a generar, se busca la existencia de ese n-grama en el grafo. Identificado el n-grama se recorre el grafo mediante el camino más probable, usando como peso de los arcos la probabilidad de sucesión entre un nodo y otro, generando la frase. Para evitar ciclos si en algún momento se llega a un nodo ya recorrido se descarta y se pasa al segundo camino más probable.

En el caso del submodelo de 1-palabras se encuentra una funcionalidad adicional:

- **Devuelve el camino más probable entre dos palabras:** Dadas dos palabras se comprueba su existencia en el grafo. Identificadas las dos palabras se devuelve el camino más probable entre los dos nodos originalmente especificados, usando como peso de los arcos la probabilidad de sucesión.

Esta última funcionalidad no se ha aplicado en el caso de 2-palabras y 3-palabras, pues la idea de encontrar el camino más corto entre bigramas “eres un” “que tal”, por ejemplo, parece poco práctica en comparación al camino más corto entre dos palabras.

5.3.2. Modelo de caracteres

Este modelo también está integrado por 3 submodelos, unigrama, bigrama y trigrama. Estos submodelos también presentan la funcionalidad de **lista de n-gramas** y **sugerir siguiente carácter** que operan de forma similar a las mencionadas anteriormente. La nueva funcionalidad propia de estos submodelos es:

- **Autocompletar palabra:** Dado a que se han añadido al vocabulario como caracteres “<>”, “</>” para indicar inicio y final de palabra, se ha podido implementar esta funcionalidad. Dado n-caracteres se encuentra el camino más probable, entre n-caracteres introducido y un n-caracteres que termine en n con el carácter “</>”.

5.4. Ejemplos de uso

En este apartado se visualizan ejemplos de uso de las funcionalidades explicadas en la Sección 5.3, para cada submodelo presente en el sistema. Los entradas y salidas aquí reflejadas han sido introducidas y obtenidas, respectivamente, mediante el uso de la interfaz por terminal proporcionada al ejecutar *main.py*

5.4.1. Lista de N-gramas

La Tabla 5.2 ofrece una lista de entradas que producirán siempre estas mismas salidas si el sistema es entrenado con el mismo conjunto de entrenamiento ofrecido en la Sección 3.1.

	Entrada		**Salida**	
Submodelos	N-grama	Tamaño máximo lista devolver	Array de siguientes n-gramas	Número de veces sucesión n-grama entrada-salida
1-palabras	‘soy’	5	[‘yo’, ‘el’, ‘un’, ‘de’, ‘tan’]	[69,51,48,42,23]
2-palabras	‘soy un’	5	[‘un pobre’, ‘un miserable’, ‘un hombre’, ‘un caballero’, ‘un buen’]	[5,4,3,3,2]
3-palabras	‘soy un hombre’	5	[‘un hombre que’, ‘un hombre ni’]	[2,1]
1-caracteres	‘a’	5	[‘</>’, ‘s’, ‘n’, ’r’, ’l’]	[319443, 90237, 83689, 78237, 60961]
2-caracteres	‘as’	5	[‘s</>’, ‘sa’, ‘st’, ‘se’, ‘si’]	[64468, 5504, 5257, 4145, 2946]
3-caracteres	‘<>as’	5	[‘así’, ‘asi’, ‘asu’, ‘aso’, ‘ase’]	[2625, 596, 467, 376, 359]

Tabla 5.2: Ejemplo de uso de la funcionalidad lista de n-gramas

Como se puede observar de los resultados a mayor orden de n-grama, menor es la frecuencia de sucesiones, debido a la explosión combinatoria que se produce con los

distintos vocabularios de cada submodelo. Podemos deducir que con el tamaño del conjunto de entrenamiento si podemos producir un decente modelo de caracteres, pero el modelo de palabras necesitaría de un mayor entrenamiento. También es curioso fijarse en como el carácter de final de palabra '</>' es observado muy frecuentemente tras una vocal, en el caso de 'a'.

5.4.2. Sugerir siguiente palabra o carácter

En este caso la tabla de resultados obtenida no es determinista, es decir, dada una misma entrada no siempre obtendremos la misma salida. Pues en este caso la salida es otorgada según probabilidades. Esta vez se proporcionarán varias salidas con la misma entrada, pues si nos quedáramos únicamente con la primera no se podría concluir demasiado de los ejemplos.

Submodelos	Entrada	1º Salida	2º Salida	3º Salida
1-palabras	'soy'	'hija'	'turco'	'algún'
2-palabras	'soy un'	'tal'	'hombre'	'aldeano'
3-palabras	'soy un hombre'	'ni'	'ni'	'que'
1-caracteres	'a'	'b'	'</>'	'</>'
2-caracteres	'as'	'p'	'</>'	'</>'
3-caracteres	'<>as'	'í'	'u'	'í'

Tabla 5.3: Ejemplo de uso de la funcionalidad sugerir siguiente palabra/carácter

Viendo la Tabla 5.2 y la Tabla 5.3 se puede concluir que la mayoría de los resultados en esta última funcionalidad se encuentran entre las 5 primeras sugerencias más comunes. Esto solamente cambiaría con la entrada 'soy' donde se puede ver como ninguna de las salidas otorgadas es una de las 5 primeras sugerencias. Esto se debe a que en el caso de 'soy' encontramos que es sucedido por una gran variedad de palabras, y este gran conjunto de otras palabras que no se encuentra entre las 5 primeras puede significar una mayor frecuencia de aparición en conjunto mayor que la de las 5 primeras sugerencias.

5.4.3. Generar una frase

A la hora de generar frases solamente los submodelos de palabras pueden hacerlo.
En este caso se ofrecerán dos entradas distintas por submodelo de palabras y de esta forma tener más ejemplos sobre los que poder discutir. Esta función también es determinista.

	Entrada		Salida
Submodelos	N-grama inicial	Tamaño máximo frase devolver	Frase producida
1-palabras	'soy'	7	"soy yo no se le había de"
1-palabras	'hola'	7	"hola dijo el que no se le"
2-palabras	'soy un'	7	"soy un pobre diablo escaso de hipérboles"
2-palabras	'eres un'	7	"eres un niño que movido de caridad"
3-palabras	'soy un	7	"soy un hombre que se esmeraba en"

	hombre’		
3-palabras	‘llueve mucho en’	7	“” – No se encuentra el trigrama “llueve mucho en” en la base de datos

Tabla 5.4: Ejemplo de uso de la funcionalidad generar una frase

Vista la Tabla 5.4 se puede discernir que el submodelo 1-palabras no es muy bueno para generar frases, esto es lógico pues los unigramas no son capaces de mantener nada de contexto. Ofrecen únicamente resultados dependiendo del unigrama anterior únicamente, por esto mismo las primeras palabras sugeridas si son correctas, pero en cuanto la frase generada supera cierta longitud acaba por no tener sentido. En el caso de los bigramas y trigramas se observa una mejora de los resultados, pero si se aumenta el tamaño máximo de la frase a devolver acabarían generando frases que perderían el sentido. Se ha de mencionar que en el caso de los trigramas es posible que introduzcamos entradas que no se encuentren en la base de datos, como es el caso de “llueve mucho en”, por ello podemos intuir que el modelo está poco entrenado.

5.4.4. Camino más probable entre dos palabras

Se recuerda que esta funcionalidad **únicamente** la implementa el **submodelo 1-palabras.** Se han realizado varias ejecuciones de esta función con diferentes entradas. La función es determinista.

Entrada		**Salida**
1º Palabra	2º Palabra	Frase más probable
‘soy’	‘hombre’	‘**soy** yo no lo que en un **hombre’**
‘eres’	‘tal’	‘**eres** un poco de **tal**’
‘cielo’	‘hola’	‘**cielo** y en ella **hola**’
‘hablar’	‘pequeño’	‘**hablar** de un **pequeño**’

Tabla 5.5: Ejemplo de uso de la funcionalidad camino más probable entre dos palabras

Las conclusiones que se pueden obtener de estos resultados son similares a las obtenidas en la funcionalidad anterior. Las frases formadas por unigramas suelen no ser correctas, al fijarse únicamente en la palabra anterior.

5.4.5. Autocompletar palabra

A la hora de autocompletar palabras solamente los submodelos de caracteres pueden hacerlo. En este caso se ofrecerán dos entradas distintas por submodelo de caracteres y de esta forma tener más ejemplos sobre los que poder discutir. Esta función también es determinista y sobre todo resulta útil en la demo planteada en la Sección 7.

	Entrada	**Salida**
Submodelos	Trozo de palabra	Sugerencia para completar
1-caracteres	‘a’	No se te recomienda hacer nada más
1-caracteres	‘d’	“d-e”
2-caracteres	‘as’	No se te recomienda hacer nada más
2-caracteres	‘po’	“po-r”
3-caracteres	‘<>as’	“as-í”

3-caracteres	‘enton’	“enton-ces”

Tabla 5.6: Ejemplo de uso de la funcionalidad de autocompletar palabra

Como se puede observar esta funcionalidad no resulta tan útil su uso por terminal, ya que varias de las secuencias introducidas pueden terminar con un “No se te recomienda hacer nada más”, especialmente si operamos desde el submodelo de 1-caracteres. Esto se debe a que el camino más probable en el submodelo de 1-caracteres, para terminar una palabra, suele ser termínala directamente. Muchos caracteres tienen como su 1º sugerencia más frecuente el carácter de final de frase, esto hace que el submodelo de 1-caracteres no rinda nada bien en el apartado de autocompletar palabras, aunque de vez en cuando puede tener alguna sugerencia como con el caso ‘d’.

6. Evaluación

Para saber la utilidad o como de bien puede desempeñar nuestro sistema se necesita encontrar algún tipo de métrica que nos sirva. El problema al que nos enfrentamos en este caso es que el sistema desarrollado no se puede considerar una aplicación de cara al usuario final, sino un modelo del lenguaje simple que podría ser integrado en un servicio más complejo.

Teniendo esto en cuenta podemos diferenciar entre **evaluación extrínseca** y una **evaluación intrínseca.** Las primeras son evaluaciones que toman muchas medidas de una aplicación, donde se valoran el rendimiento (tiempo, memoria), tiempos de respuesta, número de peticiones a servicios, etc… Su intención es que sean realizadas sobre aplicaciones finales. Como se puede ver la evaluación extrínseca no parece una opción en este caso, por lo tanto se realizará una evaluación intrínseca.

La evaluación intrínseca no nos ofrecerá datos del rendimiento de nuestra aplicación, pero nos aportará información sobre nuestro modelo del lenguaje. Una métrica comúnmente utilizada es *perplexity.*

6.1. Perplexity

La *perplexity,* a menudo abreviada como PP, es una medida que valora como de perplejo se encuentra nuestro modelo del lenguaje ante los datos, nunca vistos por el modelo, del conjunto de pruebas.

La *perplexity* de un modelo del lenguaje sobre un conjunto de prueba es la inversa de la probabilidad de aparición, que nuestro modelo asignaría a las sentencias presentes en el test, normalizada por el número de palabras. Para un conjunto de prueba $W = w_1, \dots, w_n$

$$PP(W) = p(w_1, \dots, w_n)^{-\frac{1}{n}} = \sqrt[n]{\frac{1}{p(w_1, \dots, w_n)}}$$

Como se puede observar, al ser la inversa, a menor *PP* mayor probabilidad de aparición otorga nuestro modelo a las sentencias presentes en el conjunto de prueba.

(Jurafsky & Martin, Evaluating Language Models, 2022)

Se recuerda que $p(w_1, \dots, w_n)$ puede ser expandido utilizando la regla de la cadena, o simplificado aplicando la asunción de Márkov de orden N dependiendo del modelo de N-grama que estemos analizando. Esto se explicó en más detalle en la Sección 2.1.

Un problema que encontramos con la *perplexity* es que esta no puede ser medida si el conjunto de entramiento no presenta todo el vocabulario que puede estar presente en el conjunto de pruebas. Para solucionar este problema se ha optado por la **aproximación de Laplace** a la hora de calcular la probabilidad de aparición.

Mediante este enfoque toda aquella palabra no reconocida en el conjunto de prueba será considerada como el token <UNK>. Esto implica un cambio en el cálculo de probabilidades condicionales al hacer uso de Laplace.

En el caso de una palabra (unigramas) w_i donde c_i significa el número de veces que aparece y k el número de palabras analizadas, antes de aplicar Laplace el cálculo de su probabilidad condicional sería:

$$p(w_i) = \frac{c_i}{k}$$

Tras aplicar Laplace y sabiendo que V hace referencia al tamaño del vocabulario del entrenamiento más todas las palabras no reconocidas en el conjunto de prueba por primera vez:

$$p(w_i) = \frac{1 + c_i}{k + V}$$

Gracias a esto evitamos que el cálculo de probabilidades con n-gramas no conocidos en el conjunto de pruebas otorgue 0 y por lo tanto $PP(W) = inf$.

6.2. Resultados

Todas estas consideraciones hacen que el proceso se observe como algo complejo, afortunadamente la biblioteca NLTK nos aporta herramientas para el cálculo automático de esta medida dados un conjunto de entramiento y un conjunto de prueba.

El código de evaluación del modelo se encuentra en *scripts/evaluate.py* y se ha realizado mediante la documentación de NLTK en (NLTK :: ntlk.lm package, 2022). Los resultados que la ejecución del *script* ha arrojado frente a nuestro conjunto de entramiento formado por 1.558.827 palabras y el conjunto de prueba formado por 314.253 palabras son los siguientes:

Perplexity	Unigramas	Bigramas	Trigramas
Modelo de palabras	7327,03	2838,71	3775,03
Modelo de caracteres	23,75	14,67	11,22

Tabla 6.1: Resultados perplexity sobre corpus de GPG

Para poner estos resultados en contraste también se ofrecen la *perplexity* calculada sobre un *corpus* de datos, formado por 38.000.000 palabras para el entramiento y 1.500.000 palabras para el test y usando un vocabulario cerrado de 19.979 palabras, obtenido del Wall Street Journal (WSJ) y que nos proporcionan (Jurafsky & Martin, Evaluating Language Models, 2022).

Perplexity	Unigramas	Bigramas	Trigramas
Modelo de palabras	962	170	109

Tabla 6.2: Resultados perplexity sobre corpus del Wall Street Journal

Viendo estos resultados podemos concluir:

- Los modelos de caracteres al presentar vocabularios mucho más pequeños que los modelos de palabras siempre obtendrán una mejor puntuación.

- A pesar de que no es una comparación justa, pues el modelo de WSJ cuenta con un vocabulario cerrado de 19.979 palabras frente a nuestro modelo que tras entrenamiento cuenta con 65.000 palabras aproximadamente, se puede

intuir que nuestro modelo está poco entrenado.

- Los bigramas y trigramas presentan menores valores pues al tener contexto de los últimos *tokens* se hace más sencillo predecir el siguiente. Por lo tanto ver que el resultado del modelo de trigramas en palabras es peor que el de bigramas indica que no tenemos un conjunto de entrenamiento lo suficientemente grande.

Se ha de tener en cuenta que debido a los recursos con los que se cuenta al desarrollar el proyecto, no se ha podido aumentar el corpus de datos significativamente, pues el procesamiento de estos mismos se completaría en varios días.

7. Demo técnica

Con el objetivo de mostrar la utilidad de GPG, se ha decidido desarrollar una pequeña demo técnica en la que se satisfarían los objetivos por los que inicialmente se planteó este proyecto, ayudar a un usuario conforme escribe un texto mediante la sugerencia de palabras o autocompletando las que este escribe.

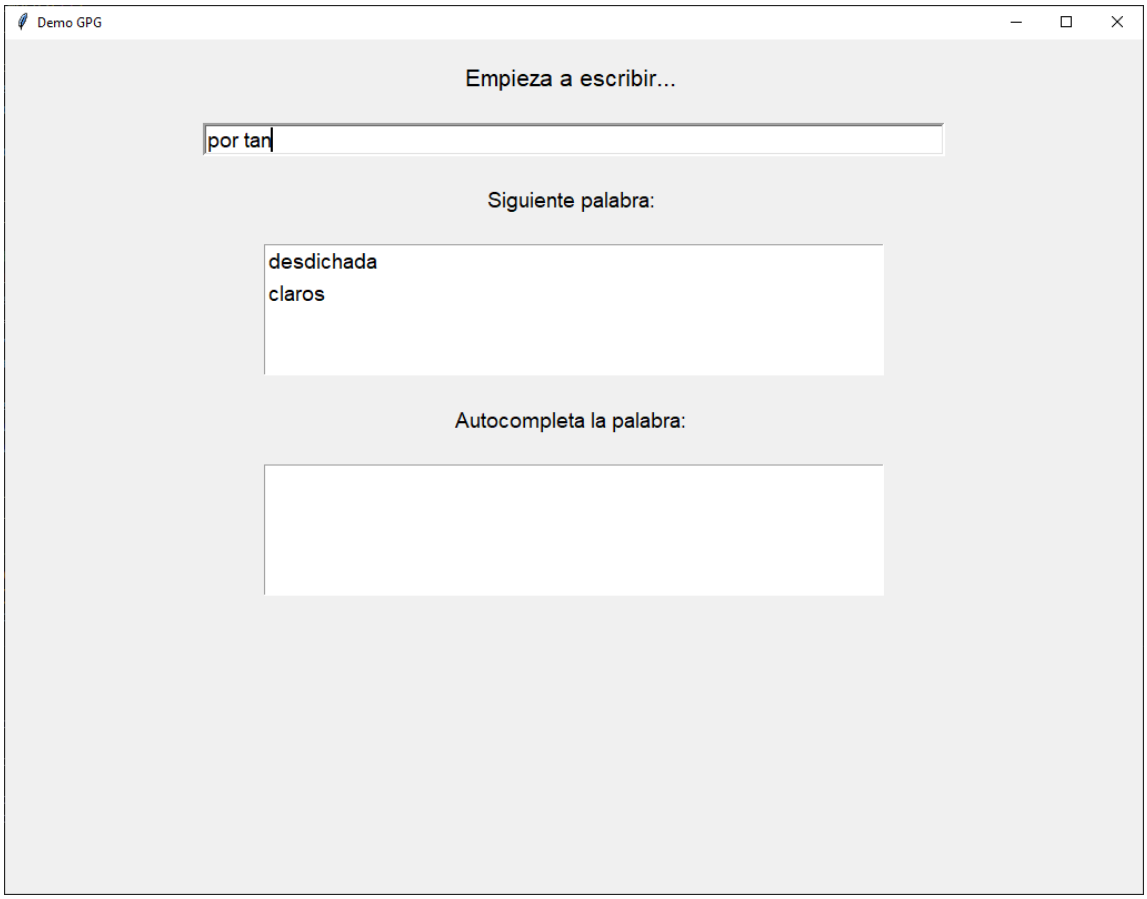

Figura 7.1: Demo técnica de GPG

El desarrollo de esta demo implica una simple GUI, desarrollada con la biblioteca Tkinter (tkinter - Interface de Python para Tcl/Tk, 2022), en la que el usuario es capaz de escribir texto y se le ofrecen **dos tipos de sugerencias** distintas.

- Siguiente palabra: Se analiza el texto escrito por el usuario, y se ofrecen en una caja de opciones diferentes sugerencias al input introducido. Se ofrece como máximo una sugerencia por cada modelo de N-Palabra, es decir 3.

- Autocompletar palabra: Dado un texto introducido por el usuario si la última entrada es una palabra a medio escribir, se ofrecen en una caja de opciones diferentes sugerencias. Se ofrece como máximo una sugerencia por cada modelo de N-Carácter, es decir 3.

Téngase en cuenta que para que modelos de bigramas o trigramas requieren de un mínimo de *tokens* escritos, de ahí que estos no ofrecerán sugerencias si el *input* no es

suficientemente largo.

Se exponen los diagramas de secuencia de cada tipo de sugerencia. En ellos se da por hecho que se realiza una secuencia de ejecución correcta, es decir, no se reflejan casos como que la entrada no sea suficientemente larga o que el *token* introducido no se encuentre en la base de datos. Si ocurriera alguna de estas situaciones la sugerencia afectada será nula y no se mostrará en pantalla.

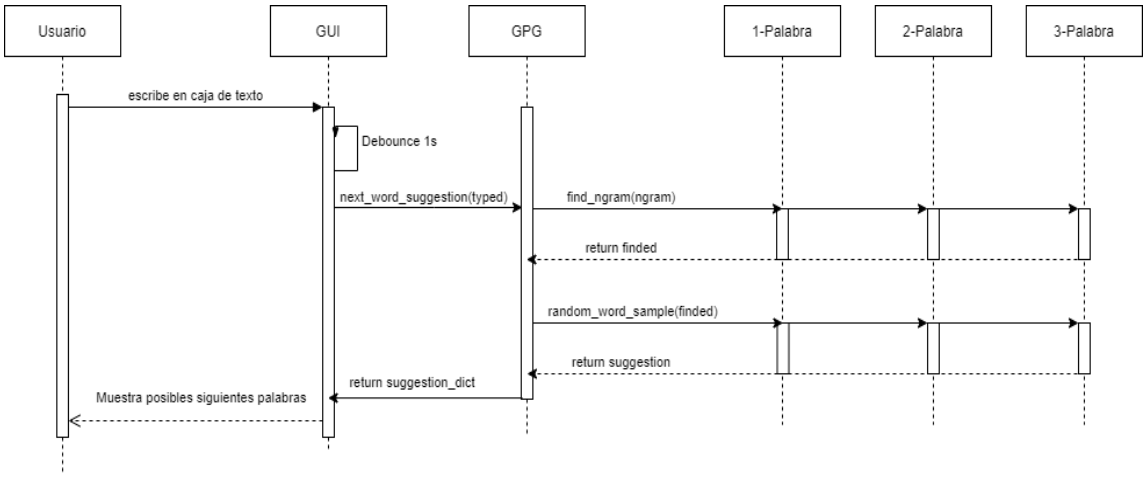

Figura 7.2: Diagrama de secuencia sugerencias de siguientes palabras en la demo

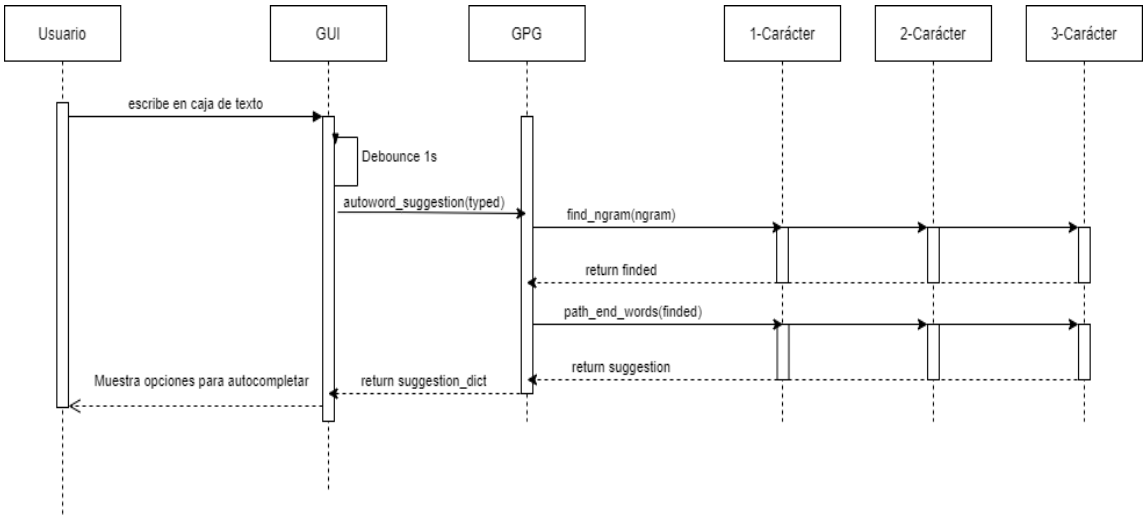

Figura 7.3: Diagrama de secuencia sugerencias para autocompletar palabras en la demo

Tanto en la Figura 7.2 como en la Figura 7.3 se puede apreciar una función llamada *debounce*, esta se encarga de realizar únicamente la última llamada de las funciones realizadas a GPG tras haber pasado 1 segundo desde que el usuario soltó una tecla. Esto se hace para que no se haga una petición a ArangoDB cada vez que el usuario tecle un nuevo carácter, mejorando así el rendimiento considerablemente.

8. Conclusiones

8.1. Conclusiones sobre objetivos

En este punto se revisarán los objetivos mencionados en la Sección 1.3 y como se les ha dado solución en el desarrollo del sistema, empezando por los objetivos primarios y continuando por los secundarios.

Objetivos **primarios:**

1. Se ha obtenido una **fuente de datos** como se detalla en la Sección 3.1

2. Los datos son transformados y almacenados siguiendo una *pipeline* en la **base de datos** ArangoDB, este flujo de datos se puede consultar en la Sección 4.2 y más detalles sobre la base de datos puede ser vista en la Sección 9.3.

3. Los datos son **representados en diferentes grafos**, los detalles de la estructura de los documentos que forman los nodos y arcos del grafo se localizan en la Sección 4.2.2, aparte en la Sección 5.2 se aportan datos del número de nodos y arcos que forman cada grafo construido.

4. GPG se compone de **dos modelos del lenguaje** a nivel de caracteres y a nivel de palabras. Cada modelo está **compuesto por 3 submodelos,** unigrama, bigrama, trigrama. Esta división se observa al principio de la Sección 4.

5. Se han implementado las **funcionalidades** propuestas y su correspondiente explicación se encuentra en la Sección 5.3. También se ofrecen ejemplos de uso de dichas funcionalidades en la Sección 5.4.

6. Se ha habilitado una interacción directa con nuestro modelo mediante **una interfaz sencilla por terminal,** el flujo de control de dicho menú puede ser visto en la Sección 4.1.

Objetivos **secundarios:**

1. Se ha configurado correctamente el despliegue y uso de un **contenedor** mediante el fichero *Dockerfile* presente en (Repositorio GPG, 2022).

2. Se ha realizado una **evaluación** de nuestro modelo como se detalla en la Sección 6

3. Se ha escrito un **fichero *readme.md*** con las instrucciones necesarias para hacer uso del proyecto, este se puede encontrar en (Repositorio GPG, 2022).

4. Se ha montado una **demo** mediante una GUI realizada con tkinter como se expone en la Sección 7.

En cuanto a los requisitos no funcionales hay que mencionar que con el objetivo de mantener el código reutilizable y organizado este se encuentra modularizado y esto queda reflejado en la Sección 5.1.

Los resultados del GPG cumplen los objetivos propuestos de forma simple. Mencionar, también, que el submodelo de unigramas de caracteres es más bien anecdótico pues su forma de rendir en las funcionalidades implementadas no es muy significativa.

8.2. Trabajo Futuro

A continuación, se listan una serie de aspectos que de aplicarse podrían mejorar el producto final con el que se ha concluido este proyecto.

1. Integración del sistema en herramientas de escritura normalmente utilizadas en forma de *plugin*.

2. Desarrollo y despliegue del sistema en una API, sobre todo si se quiere ofrecer un servicio web que utilice el modelo.

3. Reducir el impacto del problema de *data sparsity* que se originaría en el modelo de palabras al hacer uso de bigramas o principalmente en los trigramas. Esta solución pasaría parcialmente por conseguir un corpus de datos mucho mayor al actual. Una manera de hacer esto se expone en (Suzuki, Itoh, Nagano, Kurata, & Thomas, 2019) aumentando el conjunto de datos con los ejemplos generados por una red neuronal, o bien con aplicación de técnicas de *smoothing.*

8.3. Valoración personal

El trabajo realizado aun consiguiendo resultado humildes y sencillos no ha sido fácil de desarrollar. El proyecto se ha desarrollado desde cero y ha atravesado numerosas complicaciones, principalmente derivadas por el final de la relación con la empresa 8-Belts.

El hecho de haber desarrollado GPG hasta su estado actual me produce satisfacción y orgullo, pues considero que el esfuerzo depositado en él ha sido considerable. Realizar este proyecto me ha aportado nuevos conocimientos en tecnologías y ámbitos desconocidos para mí, como las bases de datos orientadas a grafos o principios del procesamiento del lenguaje natural.

Me alegro de haber tenido como mi tutor a Juan Francisco Huete Guadix por su orientación a la hora de como poder continuar el proyecto, otorgándome así la voluntad para poder terminar el mismo.

9. Tecnologías

En este capítulo se introducirán las distintas tecnologías utilizadas en el desarrollo del proyecto.

La elección del lenguaje es una decisión fundamental. En este caso los lenguajes más populares cuentan con las librerías más completas y diversas. Enfocándonos en el ámbito de análisis de datos C++, Python o Java presentan muy buenas opciones. En este caso se ha decidido optar por Python debido a su facilidad de aprendizaje y de uso.

El uso de la base de datos ArangoDB se debe a que nos aporta la posibilidad de producir grafos de los datos almacenados en la misma, quitando dicha carga de programación en la parte del lenguaje. Relacionado con el uso de esta se ha usado Docker para poder aislar dicha tecnología en un contenedor y permitir su despliegue en distinto equipos de trabajo.

9.1. Python

Python es un lenguaje de programación interpretado creado usando C y C++ por Guido van Rossum. Python es un lenguaje multiparadigma, es decir, podemos programar desde diferentes paradigmas de programación, como OOP o programación funcional.

Se caracteriza por ser uno de los lenguajes más sencillos de utilizar. El código escrito en Python es legible y se asemeja al lenguaje humano. También presenta reglas de sintaxis más laxas que otros lenguajes populares como C++.
Python también es uno de los lenguajes más utilizados por programadores en 2022, como nos indica la entrevista anual realizada por Stack Overflow (Stack Overflow, 2022).

La versión de Python utilizada es 3.10.2. Para realizar una sencilla, instalación y gestión de paquetes se ha usado el famoso gestor de paquetes Pip en su versión 21.2.4.

Las librerías instaladas utilizadas para el desarrollo del sistema han sido:

- **Python-Arango**: Implementa un *ORM* proporcionándonos una interfaz para interactuar con la base de datos desde el lenguaje. Versión 7.3.1

- **Python-Dotenv**: Habilita el uso de variables de entorno. Usado para poder desarrollar en diferentes equipos de trabajo. En este caso contiene las direcciones absolutas de los libros a leer, ya que estás rutas cambian según el dispositivo en el que trabajemos. Versión 0.20.0

- **SciPy**: Implementa una gran cantidad de algoritmos y modelos matemáticos. En concreto estamos utilizando esta librería por el módulo de estadística de la misma para poder generar una distribución discreta. Versión 1.9.1

- **Nltk:** Es la librería más conocida de Python relativa al procesamiento del lenguaje natura. En concreto se usa esta biblioteca para calcular la métrica *perplexity.* Versión 3.7

- **Tkinter:** Permite el desarrollo de sencillas interfaces gráficas de usuario mediante el uso de la biblioteca gráfica Tcl/Tk. Versión 8.6

9.2. Docker

Docker es un software de código abierto, en cuyo desarrollo encontramos que han participado grandes titanes de la industria, hablamos de Microsoft, Cisco Systems, Google, IBM, o Red Hat.

Docker intenta resolver el famoso problema en el de desarrollo de aplicaciones software resumido en la frase “en mi ordenador si funciona”. Esta tecnología nos permite aislar conjuntos de dependencias en los conocidos como “contenedores Docker”. En estos contenedores podemos desarrollar y ejecutar nuestros proyectos, evitando posibles conflictos con otras librerías o programas que pudiéramos haber instalado para previos desarrollos.

Pero sobre todo una de las características clave que ha hecho que sea tan popular es que permite hacer esto de manera automática, es decir, una vez realizada la configuración de dicho contendor el mismo puede ser utilizado en cualquier otro dispositivo que presente el mismo sistema operativo y el proyecto Docker instalado.

En este proyecto se ha utilizado Docker para poder realizar un despliegue automático de la base de datos Arango en diferentes equipos.

9.3. ArangoDB

La base de datos Arango es una base de datos multimodelo lo cual permite su utilización como una base de datos orientada a grafos, una base de datos orientada a documentos y una base de datos orientada a parejas clave-valor.

Aparte Arango cuenta con su propio lenguaje de *queries* (consultas), común a los tres modelos mencionados anteriormente. Dicho lenguaje aspira por parecerse a los lenguajes de alto nivel multipropósito, utilizando palabras reservadas similares como, por ejemplo, *for* para iterar entre los documentos de una colección en la base de datos. Gracias a esto la familiarización con sus consultas resulta más fácil.

Debido a dichas características ArangoDB es una base de datos realmente flexible, que permite una gran eficiencia y escalabilidad horizontal en las aplicaciones donde sea utilizado.

En este caso su uso ha resultado muy conveniente en el desarrollo del modelo, la base de datos nos proporciona las herramientas para el almacenamiento y tratamiento de los datos en un grafo, incorporando herramientas recorrer al mismo.

9.4. Git

Git es un sistema de control de versiones ampliamente conocido por los desarrolladores. Este es de código abierto y libre.

Normalmente un sistema de control de versiones suele ser utilizado junto a un servicio de almacenamiento en la nube de esas mismas versiones. Entre los más conocidos encontramos servicios como Github, Gitlab o BitBucket. Para facilitar el desarrollo entre diferentes equipos se ha creado un repositorio público en la plataforma Github al que se puede acceder buscando “alberto-lopov/bachelor-thesis”, mediante el siguiente hiperenlace: https://github.com/alberto-lopov/bachelor-thesis o con la referencia en la bibliografía (Repositorio GPG, 2022).

10. Bibliografía

About OpenAI. (2022). Obtenido de OpenAI Web site: https://openai.com/about/

Casas, J., Mugellini, E., & Abou Khaled, O. (2020). Overview of the Transformer-based Models for NLP Tasks. *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, (págs. 179-183).

Examples - OpenAI API. (2022). Obtenido de OpenAI API: https://beta.openai.com/examples

Free Software Foundation. (2022). *What is Free Software? - GNU Project.* Obtenido de https://www.gnu.org/philosophy/free-sw.en.html

Gudivada, V. N., Rao Vijay, D., & Raghavan, V. (2015). Handbook of Statistics. En *Chapter 9 - Big Data Driven Natural Language Processing Research and Applications* (págs. 203-238). Elsevier.

Jurafsky, D., & Martin, J. H. (2022). Chapter 24 - Bibliographical and Historical Notes. En *Speech and Language Processing* (pág. 552). Obtenido de https://web.stanford.edu/~jurafsky/slp3/ed3book_jan122022.pdf

Jurafsky, D., & Martin, J. H. (2022). Evaluating Language Models. En *Speech and Language Processing* (págs. 36-39). Obtenido de https://web.stanford.edu/~jurafsky/slp3/ed3book_jan122022.pdf

Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: an introduction. *Journal of the American Medical Informatics Association, Volume 18*, 544-551.

NLTK :: ntlk.lm package. (2022). Obtenido de NLTK :: Natural Language Toolkit: https://www.nltk.org/api/nltk.lm.html

Repositorio GPG. (2022). Obtenido de Github: https://github.com/alberto-lopov/bachelor-thesis

Russell, S., & Norvig, P. (2009). N-gram character models. En *Artificial Intelligence: A Modern Approach, 3rd edition* (págs. 861-863). Pearson.

Russell, S., & Norvig, P. (2009). N-gram word models. En *Artificial Intelligence: A Modern Approach, 3rd edition* (págs. 864-865).

Stack Overflow. (2022). *Stack Overflow Developer Survey.* Obtenido de https://survey.stackoverflow.co/2022/

Suzuki, M., Itoh, N., Nagano, T., Kurata, G., & Thomas, S. (2019). Improvements to N-gram Language Model Using Text Generated from Neural Language Model. En *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (págs. 7245-7249).

The Project Gutenberg Literary Archive Foundation. (s.f.). *About | Project Gutenberg*.

Obtenido de Project Gutenberg Web Site: https://www.gutenberg.org/

tkinter - Interface de Python para Tcl/Tk. (2022). Obtenido de Documentación de Python: https://docs.python.org/es/3/library/tkinter.html

Vincent, J. (2003). Journal of Machine Learning Research. En D. R. Bengio Y, *A neural probabilistic language model* (págs. 1137-1155).

Libros para el conjunto de datos usado:

Anónimo. (1554). *La vida de Lazarillo de Tormes y de sus fortunas y adversidades.* Obtenido de https://www.gutenberg.org/ebooks/320

Blasco Ibánez, V. (1894). *Arroz y tartana.* Obtenido de https://www.gutenberg.org/ebooks/16413

Caro, M. P. (2008). *Amar es vencer.* Obtenido de https://www.gutenberg.org/ebooks/24925

Cervantes Saavedra, M. D. (1613). *Novelas Ejemplares.* Obtenido de https://www.gutenberg.org/ebooks/61202

Daudet, A., & Cabañas, F. (1869). *Cartas de mi molino.* Obtenido de https://www.gutenberg.org/ebooks/29706

de Alarcón y Ariza, P. A. (1880). *El niño de la bola.* Obtenido de https://www.gutenberg.org/ebooks/59154

de Alarcón y Ariza, P. A. (1883). *Viajes por españa.* Obtenido de https://www.gutenberg.org/ebooks/26314

de Cervantes Saavedra, M. (1605). *Don Quijote de la Mancha.* Obtenido de https://www.gutenberg.org/ebooks/2000

Fernández y González, M. (1858). *Amparo (Memorias de un loco).* Obtenido de https://www.gutenberg.org/ebooks/27295

Gómez de la Serna, R. (1944). *La quinta de Palmyra.* Obtenido de https://www.gutenberg.org/ebooks/63228

Leopoldo Alas, C. (1884). *La Regenta.* Obtenido de https://www.gutenberg.org/ebooks/17073

Leopoldo Alas, C. (1890). *Su único hijo.* Obtenido de https://www.gutenberg.org/ebooks/17341

Martí, J. (1885). *Amistad funesta: Novela.* Obtenido de https://www.gutenberg.org/ebooks/18166

Palma Román, M. A. (1917). *Crónicas de Marianela.* (P. Balza, Ed.) Obtenido de https://www.gutenberg.org/ebooks/34565

Pereda, J. M. (1888). *La Montálvez.* Obtenido de https://www.gutenberg.org/ebooks/25812

Pereda, J. M. (1986). *Al primer vuelo.* Obtenido de https://www.gutenberg.org/ebooks/23957

Pérez Galdós, B. (1874). *Cádiz.* Obtenido de https://www.gutenberg.org/ebooks/21906

Rodó, J. E. (1900). *Ariel.* Obtenido de https://www.gutenberg.org/ebooks/22899

Valera, J. (1883). *Algo de todo.* Obtenido de https://www.gutenberg.org/ebooks/30213