eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Extracción de conocimiento a partir de descripciones en lenguaje natural

Autora

Ane Arburua Sagastibeltza

Director

Manuel Graña Romay

Resumen

Este proyecto fin de grado se enmarca en un proyecto de investigación que trata de obtener conocimiento sobre procesos de fabricación realizados por personas, a partir de las descripciones en lenguaje natural que estas personas hacen de los procesos. Debido a la finalización efectiva del proyecto Elkartek en el que se enmarca este proyecto y a la situación creada por la pandemia, se ha tenido que restringir el dominio de aplicación para que fuera abordable en estas condiciones. Los procesos de fabricación se han convertido en recetas de cocina descritas oralmente.

Por un lado, se capturan estas descripciones orales de los procesos creando una base de datos de audio. Después, se realizan las transcripciones de los registros a documentos de texto. Por último, se hace el análisis del texto, para extraer el conocimiento sobre el proceso a partir de la codificación del discurso. Este conocimiento se hace accesible al usuario, mediante un mecanismo de consultas interactivas, para que se consiga extraer información de interés.

La alumna partía sin tener conocimientos previos sobre el procesamiento del lenguaje natural. Para la realización del proyecto, se ha tenido que investigar sobre las diferentes capacidades que ofrece la plataforma de Google Cloud, y se ha centrado en los servicios de transcripción, análisis de voz y lenguaje natural que proporciona. Además, se ha buscado información de diferentes técnicas y aplicaciones que pueda tener. Asimismo, se han analizado varias herramientas abiertas para el lenguaje de programación Python, que soportan el desarrollo de proyectos de procesamiento del lenguaje natural.

En el aspecto formativo y personal, gracias a este proyecto se ha aprendido a trabajar de forma más autónoma, solucionando diferentes dudas y resolviendo problemas que han ocurrido durante su desarrollo. También se ha aprendido a gestionar mejor el trabajo, ya sea para investigar diferentes posibilidades o herramientas nuevas, como para la implementación y documentación del proyecto.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

Antes de comenzar a desarrollar el proyecto, se ha investigado sobre el procesamiento del lenguaje natural (o PLN). Comenzando desde la definición y los diferentes tipos de lenguaje que existen, investigando su historia, analizando las herramientas más comunes que se utilizan, y examinando las diferentes aplicaciones en las que se usa, se ha visto que el procesamiento del lenguaje natural es una herramienta muy utilizada, y que tiene un gran impacto sobre las tecnologías empleadas hoy en día.

1.1. Procesamiento del lenguaje natural

1.1.1. Definición del lenguaje

Hoy en día, el uso de los recursos naturales, industriales y humanos depende de la gestión eficiente de la información y el conocimiento. Además la cantidad de este conocimiento ha ido aumentando drásticamente durante los años, almacenándose sobre todo, en formato digital. Sin embargo, el conocimiento es diferente desde la perspectiva de los seres humanos y las computadoras, ya que en formato digital, se pueden copiar archivos, respaldarlos, transmitirlos, borrarlos, etc. Del mismo modo, las computadoras no pueden buscar las respuestas a preguntas formuladas, hacer inferencias lógicas sobre el contenido, generalizar la información y resumirla, cosa que las personas sí pueden hacer, ya que comprenden el lenguaje natural [Vásquez et al., 2009].

Se pueden distinguir dos tipos de lenguajes: los lenguajes naturales (español, inglés, francés, etc.) y lenguajes formales (matemático, lógico, programable, etc.). El lenguaje natural es el medio que utilizan los seres humanos de forma cotidiana, para establecer comunicaciones entre ellos. Este lenguaje se ha ido perfeccionando durante los años a partir de la experiencia, y tiene un gran poder expresivo y de razonamiento. Aun así, el lenguaje natural puede ser modulado por un lenguaje formal.

Los lenguajes formales, son aquellos que el ser humano ha creado para expresar situaciones que ocurren en diferentes áreas de conocimiento, como la mecánica, física, matemática, ingeniería o la naturaleza. La principal diferencia, es que en estos tipos de lenguajes no hay ambigüedad, y el componente semántico es mínimo.

Desde la perspectiva de la Inteligencia Artificial (o IA), el estudio del lenguaje natural tiene dos objetivos principales: por un lado, facilitar la comunicación entre el usuario y la computadora, para que los no especializados puedan acceder a ella. Por otro lado, pretende diseñar sistemas que puedan hacer tareas lingüísticas más complejas, como traducir textos, resumirlos, recuperarlos, etc.

Una de las tareas de la Inteligencia Artificial es la manipulación de lenguajes naturales usando herramientas de computación, como los lenguajes de programación. Por lo tanto, el procesamiento del lenguaje natural (o PLN) consiste en utilizar el lenguaje para poder comunicarse con la computadora. Así, se podrán desarrollar modelos que ayuden a entender los mecanismos humanos relacionados con el lenguaje, entendiendo las oraciones que proporcionan las personas, y facilitando programas que realicen tareas relacionadas con él. Aun así, el uso del lenguaje entre el ser humano y la maquina, presenta varias limitaciones, ya que las computadoras tienen una compresión limitada. Por ejemplo, el usuario no puede emplear sobrentendidos, ni introducir palabras nuevas en sus oraciones.

1.1.2. Historia del PLN

El modo en el que se entiende el procesamiento del lenguaje natural, ha ido cambiando durante los años [Keith, 2019].

A principios de la década de 1900, un profesor de lingüística suizo llamado Ferdinand de Saussure desarrolló un enfoque que describe los lenguajes como «sistemas». Saussure veía a la sociedad como un sistema de normas sociales compartidas que proporciona las condiciones para un pensamiento razonable y extendido, que da como resultado la toma de decisiones y acciones de los individuos. Cuando murió en 1913, dos de sus compañeros,

Albert Sechehaye y Charles Bally, reconocieron la importancia de sus conceptos. Gracias a ellos, escribieron el *Cours de Linguistique Générale*, publicado en 1916. El libro sentó las bases de la estructura de la lingüística y luego se expandió a otros campos, como la informática.

En 1950, Alan Turing escribió un artículo en el que describía una prueba para una máquina «pensante». Afirmó que si una máquina podía ser parte de una conversación mediante el uso de una teleimpresora, e imitaba a un humano sin haber diferencias notables, entonces la máquina podría pensar. Poco después, en 1952, el modelo de Hodgkin-Huxley mostró cómo el cerebro usa las neuronas para formar una red eléctrica. Estos eventos ayudaron a inspirar la idea de la inteligencia artificial, el procesamiento del lenguaje natural y la evolución de las computadoras.

Después, Noam Chomsky publicó su libro, *Estructuras sintácticas* en 1957. Con esto, Chomsky creó un estilo de gramática llamado *Phase-Structure Grammar*, que traducía metódicamente oraciones en lenguaje natural a un formato utilizable por computadoras. En 1958, John McCarthy lanzó el lenguaje de programación LISP, que todavía se utiliza en la actualidad. En 1964, se desarrolló ELIZA, un proceso de pregunta y respuesta mecanografiado, diseñado para imitar a un psiquiatra usando técnicas de reflexión. También en 1964, el Consejo Nacional de Investigación de EE. UU. creó ALPAC, un comité que se encargó de evaluar el progreso de la investigación sobre el procesamiento del lenguaje natural.

En 1966, la NRC y ALPAC interrumpieron el desarrollo de la IA y el PLN, al detener la financiación de la investigación sobre el procesamiento del lenguaje natural y la traducción automática, ya que muchos consideraban que la investigación sobre estos no era viable.

Hasta la década de 1980, la mayoría de los sistemas de PLN utilizaban reglas complejas escritas a mano. Pero a finales de la década de 1980, se produjo una revolución. Este fue el resultado tanto del aumento constante de la potencia computacional, como del cambio en los algoritmos de aprendizaje automático. A lo largo de esta década, IBM fue responsable del desarrollo de varios modelos estadísticos complicados y exitosos.

En la década de 1990, la popularidad de los modelos estadísticos para los análisis de procesos de lenguaje natural aumentó drásticamente. En 2001, Yoshio Bengio y su equipo propusieron el primer modelo de lenguaje neuronal, utilizando una red neuronal de retroalimentación. En el año 2011, Siri de Apple se hizo conocido como uno de los primeros asistentes exitosos del PLN y la IA del mundo para ser utilizado por los consumidores.

La combinación de un administrador de diálogo con el PLN hace posible desarrollar un sistema capaz de mantener una conversación y sonar como un humano, con preguntas, indicaciones y respuestas. Las IA modernas, sin embargo, todavía no pueden pasar la prueba de Alan Turing, y actualmente no suenan como seres humanos reales.

1.1.3. Herramientas lingüísticas

Hoy en día, existen varias herramientas lingüísticas [Jackson and Moulinier, 2002] que ayudan al procesamiento del lenguaje, ayudando a las máquinas a obtener información manejable basado en contenido creado por personas:

1. Delimitadores de oraciones y tokenizadores: para analizar las oraciones de un documento, se necesita determinar el alcance de estas oraciones e identificar sus constituyentes.

   a) Delimitadores de oraciones: detectar con precisión los límites de una oración no es una tarea fácil, ya que los signos de puntuación que marcan el final de una oración suelen ser ambiguos. Por ejemplo, el punto puede denotar un punto decimal, una abreviatura, el final de una oración o una abreviatura al final de una oración. Del mismo modo, las oraciones comienzan con una letra mayúscula, pero no todas las palabras en mayúscula comienzan una oración.

   Para eliminar la ambigüedad de los signos de puntuación, los delimitadores de oraciones a menudo se basan en expresiones regulares. Otra opción es usar técnicas empíricas que se entrenan en un corpus segmentado manualmente.

   b) Tokenizadores: los tokenizadores (también conocidos como analizadores léxicos o segmentadores de palabras), segmentan un flujo de caracteres en unidades significativas llamadas tokens. Por lo tanto, un segmento del texto puede entenderse como cualquier secuencia de caracteres separados por espacios en blanco. Aun así, un enfoque tan simple puede ser apropiado para algunas aplicaciones, pero puede dar lugar a inexactitudes, como por ejemplo si existen unidades compuestas por más de una palabra.

2. *Stemmers* y etiquetadores: para continuar, el análisis léxico es necesario, por lo que hay que identificar primero las formas fundamentales de las de palabras (la raíz) y determinar su parte del discurso (o *Part Of Speech*).

a) *Stemmers*: en el lenguaje lingüístico, los *stemmers* son analizadores morfológicos que asocian variantes del mismo término con una forma de raíz de la palabra. Se puede tomar la raíz como la forma que normalmente se encuentra como entrada en un diccionario. Por ejemplo, las palabras «biblioteca», «bibliotecas» y «bibliotecario» están asociadas a la raíz «bibliotec». Un lematizador heurístico intenta eliminar ciertas marcas superficiales de las palabras directamente para descubrir su forma más simple. En teoría, esto implica descartar tanto los afijos («in-», «a-», etc.) y los sufijos («-al», «-able», etc.), aunque la mayoría de los lematizadores utilizados solo eliminan los sufijos.

b) Etiquetadores (o *Part Of Speech taggers*): los etiquetadores se basan en etiquetar cada palabra de una oración con su etiqueta apropiada. Por ejemplo, se puede decidir si una palabra dada es un sustantivo, verbo, adjetivo, etc. Si a las palabras se les asignara una sola etiqueta POS, sería una tarea sencilla. Sin embargo, a algunas palabras se les pueden asignar múltiples etiquetas, y la función del etiquetador tendrá que ser elegir la correcta.

Existen dos enfoques para el etiquetado POS: el enfoque basado en reglas, y el estocástico. Por un lado, un etiquetador basado en reglas intenta aplicar algún conocimiento lingüístico para descartar secuencias de etiquetas que son sintácticamente incorrectas. Por otro lado, los etiquetadores estocásticos se basan en datos de entrenamiento y abarcan enfoques que se basan únicamente en información de frecuencia o probabilidades para eliminar la ambigüedad de las asignaciones de etiquetas. Los más simples eliminan esta ambigüedad basándose únicamente en la probabilidad de que una palabra aparezca con una etiqueta en particular. Esta probabilidad se calcula normalmente a partir de un conjunto de entrenamiento, en el que las palabras y las etiquetas ya se han emparejado manualmente.

3. Reconocedores de nombres y entidades: en algunos casos, se necesita ir más allá del etiquetado de parte del discurso, por ejemplo, si se quiere construir un sistema que extraiga noticias interesantes de varios documentos. Se necesitará identificar a las personas, los nombres de las empresas y sus relaciones, por lo que puede resultar útil saber que una palabra determinada es un nombre propio. Por lo tanto, se pueden clasificar estos nombres propios, sabiendo si designan personas, lugares, empresas, organizaciones y similares.

4. Analizadores gramaticales: el análisis o *parsing* se realiza respecto a una gramática, que presenta un conjunto de reglas que dicen qué combinaciones de qué partes del

discurso generan frases y estructuras de oraciones bien formadas. La ingeniería lingüística al escribir reglas gramaticales requiere mucha mano de obra. Aunque se han escrito grandes gramáticas de uso general (sobre todo en inglés), ninguna tiene una certeza del 100 % de todas las estructuras que se puedan encontrar en textos aleatorios. Por lo tanto, cualquier programa que se proponga analizar texto, tendrá que asumir que hay palabras no reconocidas y estructuras de frases inesperadas.

Estas herramientas muestran que hay recursos tanto teóricos como prácticos disponibles para ayudar en la construcción de sistemas de procesamiento del lenguaje natural.

1.1.4. Aplicaciones

Aun así, más allá de estas técnicas, existen varias aplicaciones en las que se ha empleado el procesamiento del lenguaje natural [Liddy, 2001].

1. Recuperación de información: aunque no siempre se haya utilizado el procesamiento del lenguaje natural, se trata de buscar documentos relacionados en base a la consulta que pueda tener un usuario.

2. Extracción de información: se centra en el reconocimiento, etiquetado y extracción de una representación estructurada de ciertos elementos clave de información, por ejemplo, personas, empresas, lugares, o organizaciones. Luego, estas extracciones se pueden utilizar para una variedad de aplicaciones, que incluyen sistemas de respuesta a preguntas, visualización y minería de datos.

3. Sistemas de respuesta a preguntas: a diferencia de la recuperación de información, que proporciona una lista de documentos potencialmente relevantes en respuesta a la consulta de un usuario, la respuesta a preguntas proporciona al usuario solo la parte del texto en la que se dan las respuestas.

4. Sistemas de resumen: los niveles más altos del PNL, particularmente el nivel de discurso, pueden potenciar una implementación que reduce un texto más grande a una representación del texto abreviada y más corta, pero bien construida.

5. Traducción automática: se han utilizado varios niveles del procesamiento del lenguaje en los sistemas de traducción automática, que empiezan desde el enfoque basado en palabras hasta aplicaciones que incluyen niveles más altos de análisis de texto.

6. Sistemas de diálogo: los sistemas de diálogo, que normalmente se centran en una aplicación concreta (por ejemplo, un frigorífico o sistema de sonido doméstico), utilizan actualmente los niveles fonético y léxico del lenguaje. Gracias a la utilización de todos los niveles de procesamiento del lenguaje explicados anteriormente, se pueden ofrecer sistemas de diálogo funcionales.

Por lo tanto, se puede ver que a lo largo de los años se ha utilizado el PLN en varias aplicaciones útiles. Sin embargo, hoy en día, el procesamiento del lenguaje natural es uno de los temas más populares en el campo de la ciencia de datos [Sharma, 2020], por lo que existen aplicaciones más recientes, que están muy presentes en el día a día:

1. Autocorrección y Autocompletar de Google Search: después de escribir varias letras en el motor de búsqueda de Google, se muestran los posibles términos de la búsqueda. Además, si se introduce algo con errores tipográficos, se corrigen y aun así consigue encontrar resultados relevantes. Por lo tanto, puede ayudar encontrar resultados precisos de manera muy eficiente.

2. Traductores de lenguaje: en el pasado, los sistemas de traducción automática se basaban en diccionarios y reglas, por lo que sus posibilidades eran limitadas. Sin embargo, debido a la evolución en el campo de las redes neuronales y la disponibilidad de datos y máquinas potentes, la traducción automática, se ha vuelto más precisa. Hoy en día, herramientas como Google Translate pueden convertir fácilmente texto de un idioma a otro.

3. Monitoreo de las redes sociales: analizar las opiniones en las redes sociales puede ayudar a generar información valiosa. Hoy en día, las empresas utilizan varias técnicas de PNL para analizar publicaciones en redes sociales y saber qué piensan los clientes sobre sus productos. También se emplea para comprender los problemas que puedan tener sus clientes al usar sus servicios.

4. *Chatbots*: muchas empresas utilizan *chatbots* en sus aplicaciones y sitios web, para ayudar a resolver consultas básicas que pueda tener un cliente. No solo facilita el proceso para las empresas, sino que también evita a los clientes la frustración de esperar para interactuar con el servicio de asistencia telefónica. Además, hoy en día se han convertido en un asistente personal, ya que pueden recomendar un producto y obtener comentarios de otros clientes.

5. Análisis de encuestas: cuando muchos clientes realizan una encuesta, se generan demasiados datos para que una persona pueda leerlos y sacar una conclusión de ellos. Ahí es donde las empresas utilizan el procesamiento del lenguaje natural, para analizar las encuestas y generar información a partir de ellas, como conocer los sentimientos de los usuarios sobre un evento a partir de los comentarios y analizar las reseñas de productos para comprender sus pros y contras.

6. Publicidad dirigida: es un tipo de publicidad online en la que se muestran anuncios al usuario en función de su actividad en la web. La publicidad dirigida funciona principalmente con la concordancia de palabras clave. Los anuncios están asociados con una palabra clave o frase, y se muestran solo a aquellos usuarios que buscan la palabra similar a la clave con la que se ha asociado el anuncio.

7. Contratación y reclutamiento: el reconocimiento de entidades se puede utilizar para extraer información como habilidades, nombre, ubicación y educación de un currículum. Luego, estas características se pueden usar para representar a los candidatos y poder clasificarlos en categorías de apto o no apto para un rol en particular.

8. Asistentes de voz: se trata de software que utiliza el reconocimiento de voz, la comprensión del lenguaje natural y el procesamiento del lenguaje para comprender los comandos verbales de un usuario y realizar acciones en consecuencia.

9. Correctores gramaticales: son capaces de corregir la gramática, la ortografía, sugerir sinónimos y ayudar a entregar contenido con mayor claridad. También ayudan a mejorar la legibilidad del contenido y, por lo tanto, permiten al usuario transmitir su mensaje de la mejor manera posible.

10. Filtrado en el correo electrónico: mediante la clasificación de texto, se puede filtrar el contenido de un mensaje en categorías predefinidas, como puede ser en el caso del correo electrónico.

En conclusión, se puede decir que el procesamiento del lenguaje natural es un campo muy amplio, que sirve de herramienta para muchas aplicaciones actuales que se van actualizando frecuentemente.

2. CAPÍTULO

Planificación

Antes de comenzar con el diseño y desarrollo del proyecto, se ha hecho la planificación en base a los objetivos establecidos. Se ha analizado el alcance del proyecto, junto a la descomposición de las diferentes tareas. También se ha estimado el tiempo de dedicación del proyecto, y se ha comparado con la desviación que ha tenido al final. Por último, se han identificado posibles peligros que pueden suceder, y se ha descrito el sistema de información que se ha empleado.

2.1. Alcance

El objetivo principal del proyecto es analizar procesos de fabricación realizados por personas a partir de sus descripciones transmitidas por voz. Para este análisis se usa el procesamiento del lenguaje natural, consiguiendo así, extraer información a partir de esos procesos descritos. Por lo tanto, por un lado se debe diseñar y desarrollar un sistema que permita cumplir con esos objetivos. Por otro lado, se deben investigar diferentes técnicas y herramientas que se ofrecen en Python para este desarrollo.

Además, se tendrá que planificar el proyecto desde el principio, identificando las diferentes fases a realizar, estimando el tiempo para dedicarle a cada fase, y analizando los peligros que puedan ocurrir durante el desarrollo. Por último, todo el trabajo creado debe proyectarse en una memoria que contenga todos los detalles del proceso realizado, los resultados obtenidos y las conclusiones que se puedan sacar. También se tendrá que crear una presentación para realizar la defensa del trabajo.

2.2. Tareas

Una parte importante de la planificación es identificar las diferentes tareas que hay en el proyecto. En la estructura de descomposición del trabajo (EDT) de la figura 2.1, se puede ver la estructura jerárquica de las tareas necesarias para cumplir con todos objetivos establecidos.

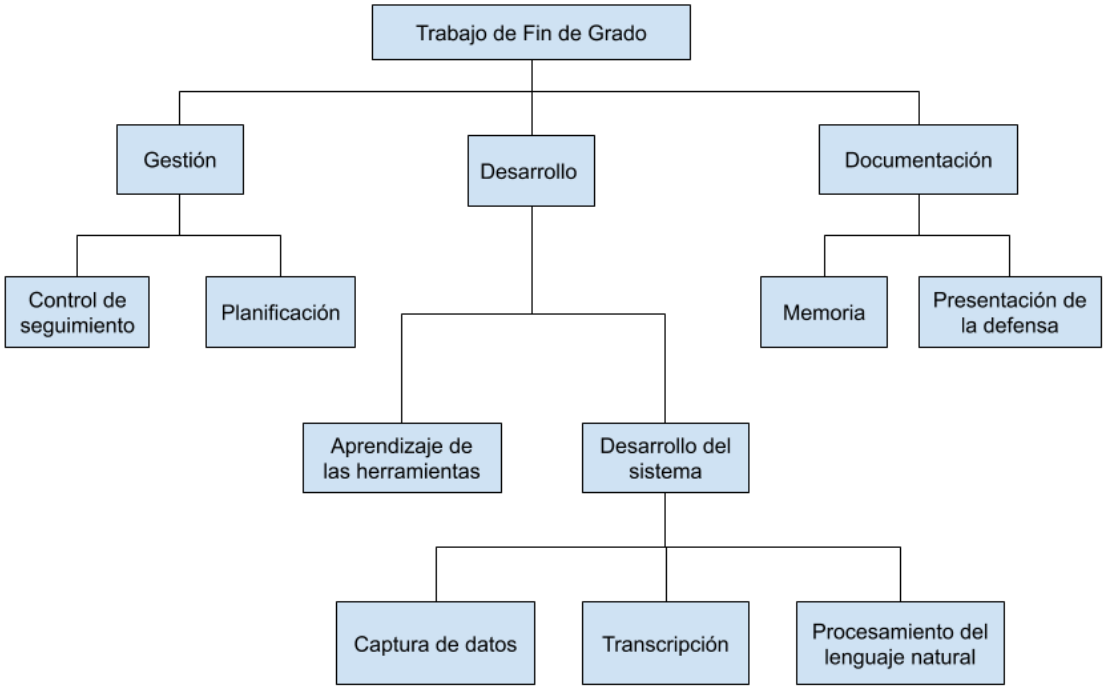


Figura 2.1: Estructura de descomposición del trabajo

Como se puede observar, el trabajo se divide en tres partes principales, la gestión, el desarrollo y la documentación.

1. Gestión: se trata de la fase que está presente durante todo el ciclo de vida del proyecto, pero toma más tiempo al principio. Se divide en dos partes diferentes, el control de seguimiento y la planificación.

    a) Control de seguimiento: es la fase que consiste en hacer el seguimiento del proyecto, ya sea del desarrollo, o de la documentación. Para ello, se controlan las horas que se hayan necesitado en cada parte, y al final, se compara con en

tiempo estimado en un principio. También se tienen en cuenta las reuniones realizadas con el tutor, para actualizar el estado del proyecto.

b) Planificación: en esta parte de la gestión, se identifican los objetivos y alcance del proyecto, y se estiman las horas necesarias para cada fase. Después, se compara esta estimación con el tiempo real que se haya necesitado. A continuación, se describe el sistema de información empleado, y se identifican los peligros posibles que puedan surgir durante el desarrollo del proyecto. Una buena planificación inicial, pueda ayudar a que el proyecto se desarrolle de la manera más ordenada y efectiva posible, por lo tanto, esta fase puede llevar más tiempo al principio.

2. Desarrollo: es la fase que contiene la mayor carga de trabajo, ya que se trata de investigar sobre diferentes herramientas para el proyecto, y de desarrollar o implementar el sistema según los objetivos.

   a) Aprendizaje de las herramientas: por un lado, antes de comenzar a implementar el trabajo, hay que investigar diferentes posibilidades que se ofrecen para cumplir con los objetivos previstos. Después de encontrar las herramientas posibles, hay que familiarizarse con ellas y aprender a utilizarlas adecuadamente. Por lo tanto, esta parte también puede llevar tiempo, ya que el uso de las herramientas adecuadas influye directamente con los resultados del proyecto.

   b) Desarrollo del sistema: se trata de la implementación del sistema, empleando las herramientas y recursos investigados en la fase anterior. Se divide en tres partes principales, la captura de datos, la transcripción y el procesamiento del lenguaje natural.

      1) Captura de datos: trata de conseguir la información obtenida a través de descripciones de procesos de fabricación. Una vez capturados los datos en formato de audio, se crea una base de datos que guarde todo el contenido.
      2) Transcripción: se transcriben los datos de audio obtenidos en la fase anterior, convirtiéndolos en archivos de texto.
      3) Procesamiento del lenguaje natural: trata de procesar los archivos de texto obtenidos en la fase anterior, utilizando técnicas del procesamiento del lenguaje natural. Una vez analizado y transformado el texto original, el objetivo es extraer información sobre los procesos descritos al principio.

3. Documentación: esta fase también esta presente durante todo el desarrollo del proyecto, ya que hay que plasmar toda la información analizada, las decisiones tomadas, y los resultados y conclusiones obtenidos en cada momento.

   a) Memoria: por un lado, en la memoria del proyecto se plasma toda la información mencionada anteriormente. Además, tiene que ser un documento detallado, para que otras personas que la lean obtengan toda la información posible sobre los procesos realizados.

   b) Presentación de la defensa: por último, en la presentación de la defensa, se tiene que resumir la memoria documentada a través de diapositivas. Con estas diapositivas, se hará una presentación oral de entre 15 a 20 minutos, con todos los detalles del proyecto creado.

2.3. Dedicación del proyecto

Una vez identificadas todas las fases del proyecto, se ha hecho una estimación de horas de dedicación para cada fase (teniendo en cuenta que en total, se deberían emplear un total de 300 horas aproximadamente). Aun así, hacer la previsión de las horas necesarias no es una tarea fácil, ya que pueden surgir problemas durante el desarrollo del proyecto, alargando su duración. La tabla 2.1 contiene las horas estimadas y reales calculadas, junto a la desviación entre ellas.

	Horas estimadas	Horas empleadas	Desviación
GESTIÓN	**30 h**	**22 h**	**-8 h**
Control de seguimiento	10 h	7 h	-3 h
Planificación	20 h	15 h	-5 h
DESARROLLO	**150 h**	**175 h**	**+25 h**
Aprendizaje de las herramientas	50 h	65 h	+15 h
Desarrollo del sistema	**100 h**	**110 h**	**+10h**
Captura de datos	10 h	5 h	-5 h
Trascripción	40 h	35 h	-5 h
Procesamiento del lenguaje natural	50 h	70 h	+20 h
DOCUMENTACIÓN	**120 h**	**130 h**	**+10 h**
Memoria	100 h	120 h	+20 h
Presentación de la defensa	20 h	10 h	-10 h
TOTAL	**300 h**	**327 h**	**+27 h**

Tabla 2.1: Horas estimadas, horas empleadas y desviación de la dedicación

Como se puede observar, en la fase de la gestión ha habido una desviación de 8 horas sobreestimadas. Sin embargo, en la fase del desarrollo se han estimado 25 horas menos de las que en realidad se han cumplido. Se trata de la mayor desviación entre todas las fases principales, y puede haberse dado por la falta de conocimiento inicial en área del proyecto.

Por un lado, se han empleado 10 horas de más buscando información sobre diferentes aplicaciones del procesamiento del lenguaje natural, y analizando las herramientas para su desarrollo. Aunque para la fase de la captura de datos y la transcripción se hayan sobreestimado las horas, en la fase del procesamiento del lenguaje ha habido un incremento de horas considerable (20 horas de más) ya que ha sido la parte más complicada de desarrollar.

En cuanto a la documentación, aunque para crear las diapositivas de la presentación se hayan necesitado menos horas de las estimadas, en la parte de la memoria también ha habido un incremento de horas importante, ya que se trata de un documento extenso. En conclusión, ha habido diferencias en la estimación con la realidad, y al final, la desviación ha sido de 27 horas de más, que no se aleja tanto de las 300 establecidas al inicio.

2.4. Identificación de peligros

Existen una serie de peligros que puede haber a medida que se desarrolle el proyecto. La identificación de estos posibles problemas puede ayudar a tomar precauciones para evitarlos, y tomar decisiones rápidas en caso de que ocurran. Por un lado, existe la opción de que haya problemas a la hora de programar. Para ello, se podría comentar el código y justificar las decisiones tomadas, para que a la hora de resolverlo sea más fácil detectar el error. Además, analizar diferentes bibliotecas o herramientas puede ser de gran ayuda para facilitar la implementación.

Por otro lado, también puede haber falta de tiempo a la hora de desarrollar el proyecto, o redactar la documentación. En este caso, se tendría que presentar el trabajo en la próxima convocatoria acordada. Aun así, creando una planificación exhaustiva y viable desde un principio, se podría evitar este problema.

Que se pierda el contenido creado puede ser otro riesgo de importancia. Para poder evitarlo, todo el contenido creado se guardará en una carpeta compartida de Google Drive, y se crearán copias locales tanto del código, como de la documentación y información

obtenida. De este modo, en caso de que se pierda algo, siempre habrá una versión anterior disponible.

2.5. Sistema de información

Mantener un sistema de información ordenado también es importante a la hora de ser eficientes con el trabajo. En este caso, se ha creado una carpeta compartida en Google Drive, para que tanto el alumno como el tutor tengan acceso a la documentación y el código en todo momento. También se han ido guardando copias locales de los resultados obtenidos a medida que avanzaba el proyecto. De este modo, en caso de que hubiera algún problema con la plataforma, siempre habría una copia de seguridad para no perder todo el trabajo realizado.

3. CAPÍTULO

Objetivos y recursos

A continuación, se analizan tanto los objetivos al principio del proyecto, como los objetivos definitivos y adaptados finalmente. También se han investigado posibles herramientas y recursos para el desarrollo del proyecto, entre ellos, varios servicios de Google Cloud, y diferentes bibliotecas para el procesamiento del lenguaje natural en Python.

3.1. Objetivos del proyecto

En un principio, el proyecto se enmarcaba en una colaboración con Vicomtech para un proyecto del programa Elkartek. Sin embargo, dicho proyecto finalizó a principios de 2020, de modo que la captura de las descripciones de procesos no se pudo hacer en colaboración con los *partners* de dicho proyecto. En consecuencia, se eligió otro tema que describiera procesos de fabricación realizados por humanos: las recetas de cocina.

A partir de la descripción oral de estas recetas, el objetivo principal es crear una base de datos de audio, hacer la transcripción del audio a texto, y mediante técnicas de procesamiento del lenguaje natural, extraer información del texto trascrito para resolver preguntas planteadas por el usuario. En este proyecto no se ha llegado a construir un sistema de interrogación. Para analizar el conocimiento recogido en las descripciones procesadas se ha realizado un análisis semántico utilizando herramientas de codificación (*word2vec*) y de visualización de datos de alta dimensión (t-SNE).

En consecuencia, los objetivos operativos son la familiarización con la plataforma de

Google Cloud, y la investigación sobre diferentes aspectos del procesamiento del lenguaje natural. El lenguaje de programación seleccionado es Python, ya que es compatible con la plataforma de Google, y existen muchos recursos para las tareas del procesamiento y extracción de la información accesibles desde Python.

3.2. Tecnologías utilizadas

Teniendo en cuenta los objetivos del proyecto, se han investigado varias herramientas para llevarlo a cabo. Por un lado, se ha obtenido información sobre la plataforma Google Cloud, de los servicios que ofrece, y cómo poner en marcha un proyecto. Por otro lado, se han analizado varias bibliotecas para Python, que permiten procesar el lenguaje natural, extraer información del texto, y visualizar los datos obtenidos de forma fácil.

3.2.1. Google Cloud

Es una plataforma que ofrece varios recursos físicos como computadoras, unidades de disco duro, y recursos virtuales. Todos estos recursos se ubican el diferentes regiones del mundo (en Asia, Australia, Europa, América del Norte, y América del Sur), consiguiendo así, estar lo más cerca posible de cada cliente [1].

Para acceder a todos estos recursos, Google Cloud ofrece diferentes tipos de servicio [2], ya sean de procesamiento, almacenamiento, bases de datos, herramientas de redes, operaciones, herramientas para desarrolladores, estadísticas de datos, IA y aprendizaje automático, administración de API, nubes híbridas y múltiples, migración, identidad y seguridad, computación sin servidores, contenedores, Internet de las cosas, herramientas de administración, salud y ciencias biológicas, o multimedia y videojuegos. Por lo tanto, cuando un cliente desea crear una aplicación, puede hacer uso de uno o varios servicios para implementar su proceso.

Los recursos que los clientes usen, deben pertenecer a un proyecto [3], que debe tener su propio nombre, una ID, y un número de proyecto (que proporciona Google Cloud). Cada proyecto tiene su propia configuración, con sus respectivos permisos. Además, un proyecto no podrá acceder a los recursos de otro, es decir, serán independientes, si así lo

[1] Documentación sobre recursos: https://cloud.google.com/docs/overview?gcp_resources
[2] Servicios de Cloud: https://cloud.google.com/docs/overview#accessing_resources_through_services
[3] Proyectos de Google Cloud: https://cloud.google.com/docs/overview#projects

desea el usuario. Cada proyecto también se asocia a una cuenta de facturación y a la vez, una cuenta podría facturar más de un proyecto.

Existen tres formas para que los clientes accedan a los servicios que se ofrecen [4]. Por un lado, está Google Cloud Console, que proporciona una interfaz gráfica de usuario para la administración de proyectos. En este apartado, se podrán crear proyectos, editar y ver los recursos empleados, visualizar los datos de trazas, visualizar las solicitudes de tráfico, errores y latencia, ver el estado de la plataforma, consultar la facturación del proyecto, reportar errores, acceder a la documentación de los servicios, explorar los tutoriales y leer las noticias relacionadas con cada proyecto.

Por otro lado, está la interfaz de línea de comandos, por si se prefiere trabajar desde aquí. Para ello, la herramienta que se ofrece es *gcloud*, con el que se puede ver el flujo de trabajo, de desarrollo, y los recursos empleados de Google Cloud en la propia terminal. Para ejecutar los comandos de la herramienta, se puede instalar el SDK de Cloud, trabajando en una terminal propia (localmente), o usar Cloud Shell, para trabajar en una ventana del navegador.

Por último, existe la opción de usar las bibliotecas cliente para la administración de recursos, que ofrecen las API con dos propósitos: las API de apps para acceder a los servicios de Google Cloud, y las API de Administrador para controlar los recursos.

Todos estos recursos tienen un precio, pero la plataforma ofrece un nivel gratuito para que el cliente pueda acceder a los tutoriales, leer la documentación y probar los servicios gratuitamente. Esta prueba gratuita consta de la obtención de un crédito de $300 durante 12 meses, con un acceso limitado a todos los recursos de Google Cloud.

El programa tiene varios requisitos [5], ya que solo se podrá acceder a él si el cliente nunca ha contratado los servicios de pago de Google Cloud, Google Maps Platform o Firebase anteriormente. Además, el cliente tampoco se podrá haber registrado con anterioridad para la prueba gratuita, y tendrá que habilitar la facturación en la consola (Google Cloud Console). Por lo tanto, se deberán introducir la tarjeta de crédito o los datos bancarios para verificar la identidad, comprobar que no haya accedido a la prueba anteriormente, y distinguir entre clientes reales y robots. Después de proporcionar una tarjeta de crédito, la plataforma se encargará de hacer un transacción de entre $0.00 y $1.00 dólares. No se trata de un cargo permanente, sino de una solicitud para validar la cuenta del cliente.

El cliente también deberá aceptar las Condiciones del Servicio de la prueba gratuita, y

[4] Interacción: https://cloud.google.com/docs/overview#ways_to_interact_with_the_services
[5] Requisitos del nivel gratuito: https://cloud.google.com/free/docs/gcp-free-tier#free-trial

tener en cuenta que algunas acciones están restringidas, como por ejemplo, extraer criptomoneda. La prueba finalizará cuando hayan pasado los 12 meses desde la facturación, o cuando el cliente haya gastado los $300 iniciales. Después de que finalice la prueba, el cliente tendrá que actualizar a una cuenta de pago para poder seguir usando Google Cloud. De no hacerlo, todos los recursos de la prueba gratuita se detendrán, y se perderán los datos de almacenamiento. Al final, el cliente recibirá un mensaje indicando la cancelación de la prueba, y no se sumará ningún cargo a no ser que el cliente desee contratar dicho servicio de pago.

Además, Google Cloud también ofrece el programa siempre gratuito [6]. Este, consiste en varios servicios y recursos gratuitos, controlados por intervalos mensuales. El programa también tiene varios requisitos, como no tener ningún contrato personalizado con Google, y tener una cuenta de facturación habilitada. Si la cuenta del cliente esta en regla, podrá comenzar a usar los servicios gratuitos teniendo en cuenta los límites de cada uno. Estos límites varían según el servicio que se quiera emplear, y usan medidas propias para el control del uso.

IA y aprendizaje automático

Google Cloud ofrece varias API para IA sin que el cliente tenga que crear y entrenar sus propios modelos. Para ello, la sección de aprendizaje automático ofrece varios servicios específicos como Video AI (análisis de vídeos), Speech-to-Text (conversión del audio en texto), Text-to-Speech (síntesis de voz), Vision AI (detectar emociones o texto), Cloud Natural Language (análisis de opinión, entidades, sintaxis y clasificación de contenido), AutoML (aprendizaje automático), Cloud Translation (traducir texto), Dialogflow (creación de interfaces de conversación), y AI Platform (alojar modelos de aprendizaje automático).

Speech-to-Text: El servicio Speech-to-Text, como el nombre indica, permite a los usuarios convertir el audio en texto a través de modelos de redes neuronales. Ofrece una API que es capaz de reconocer 120 idiomas (entre ellos el español) y variantes. Para ello, utiliza una red neuronal de *Deep Learning* (aprendizaje profundo) con el audio, para que la precisión de la transcripción sea la más precisa y correcta posible.

Para la transcripción, el audio se puede transmitir tanto de forma inmediata, a medida que el usuario habla, como a través de un archivo de audio almacenado de forma local o en

[6] Programa siempre gratuito: https://cloud.google.com/free/docs/gcp-free-tier#always-free

la nube [7]. Además, el formato del audio puede ser corto (menos de un minuto) o largo (un minuto o más). También es capaz de detectar y transcribir los sustantivos propios (de lugares, nombres, etc) y utilizar el formato adecuado para cada ocasión (fechas, números de teléfono, etc).

Existen diferentes tipos de modelos para cada caso [8], para que el reconocimiento de voz se haga de forma optimizada y sea la más adecuada posible. Entre los modelos, se encuentran *command_and_search* para consultas cortas, *phone_call*, adecuado para llamadas de teléfono o audio guardado de origen telefónico (ya que suelen rondar los 8 khz), *video* para audio que pertenezca a un vídeo o que participen varios interlocutores (con una tasa de muestreo de 16 khz o más, se trata de un modelo Premium) y *default* para audio que no se adapte a los casos anteriores (también para tasas de muestreo de 16 khz o más).

Gracias al programa siempre gratuito que ofrece la plataforma, el reconocimiento de voz es gratuito para un total de 60 minutos de procesamiento [9], tanto para modelos estándar como para modelos Premium, como se puede ver en la figura 3.1.

Función	Modelos estándar (todos excepto los modelos mejorados para vídeo y llamadas telefónicas)		Modelos mejorados (para vídeo y llamadas telefónicas)	
	De 0 a 60 minutos	De más de 60 minutos a 1 millón de minutos	De 0 a 60 minutos	De más de 60 minutos a 1 millón de minutos
Reconocimiento de voz (opción predeterminada, sin almacenamiento de registros de datos)	Gratis	0,006 USD por cada 15 segundos**	Gratis	0,009 USD por cada 15 segundos**
Reconocimiento de voz (con el almacenamiento de registros de datos habilitado)	Gratis	0,004 USD por cada 15 segundos**	Gratis	0,006 USD por cada 15 segundos**

Figura 3.1: Precio del Servicio Speech-to-Text

La plataforma ofrece tres herramientas para la transcripción de audio:

1. Uso de bibliotecas cliente: se envía a una solicitud de reconocimiento de voz a la API mediante las bibliotecas cliente de Google Cloud [10]. Para ello, se debe instalar la biblioteca *google-cloud-speech*, y hacer una solicitud mediante *recognize* a la API de Speech-to-Text.

[7] Solicitudes Speech-to-Text: https://cloud.google.com/speech-to-text/docs/basics#speech_requests
[8] Modelos de reconocimiento: https://cloud.google.com/speech-to-text/docs/basics#select-model
[9] Precio Speech-to-Text:https://cloud.google.com/speech-to-text/pricing#pricing_table
[10] Uso de bibliotecas cliente: https://cloud.google.com/speech-to-text/docs/quickstart-client-libraries

2. Uso de la herramienta *gcloud*: se envía una solicitud de reconocimiento de voz a la API, con la herramienta de *gcloud* desde la línea de comandos [11]. Para ello, como en el anterior caso, se envía una solicitud de *recognize* a la API de Speech-to-Text. Una vez ejecutado el comando e indicada la ubicación del archivo de audio, se transcribe el audio de un FLAC.

   Un FLAC, es tanto una codificación como un formato de archivo. En este caso, para la API de Speech-to-Text, se trata de una codificación gratuita, en la que se requiere que los datos de audio incluyan un encabezado. Si se quiere referir al archivo de formato FLAC, se deberá utilizar el formato «archivo .FLAC».

   La respuesta a la solicitud de transcripción se expresa en formato JSON en la misma línea de comandos, indicando la certeza, y la misma transcripción.

3. Uso de la línea de comandos: se envía una solicitud de reconocimiento a la API mediante la interfaz de REST y el comando *curl* [12]. Para ello, se debe crear un archivo de solicitud JSON, especificando la codificación, la tasa de muestreo en hercios, el lenguaje del audio, y su ubicación. El archivo se deberá guardar en un texto sin formato .json. Una vez creado el archivo, se usa el comando *curl* para hacer una solicitud *recognize*, indicando el nombre del archivo de la solicitud JSON. Si la transcripción se hace con éxito, se devolverá la respuesta en formato JSON en la misma línea de comandos, indicando la certeza, y la misma transcripción.

Una vez elegida la herramienta para hacer la transcripción, Speech-to-Text ofrece tres métodos para realizar el reconocimiento:

1. Reconocimiento síncrono [13]: se envían los datos de audio a la API, hace el reconocimiento de los datos, y da la respuesta después de procesar todo el archivo de audio. En este caso, la duración del audio debe ser igual o menor a un minuto. La solicitud síncrona implica un bloqueo, es decir, se muestra una respuesta antes de procesar la siguiente solicitud en Speech-to-Text, que normalmente se procesa más rápido que en tiempo real.

   Una solicitud síncrona consiste en dos partes, la configuración de voz y los datos de audio. En cuanto a la configuración, hay diferentes parámetros a elegir en el campo *config*, entre ellos la codificación (*encoding*), la tasa de muestreo en hercios

[11] Uso de gcloud: https://cloud.google.com/speech-to-text/docs/quickstart-gcloud
[12] Uso de la línea de comandos: https://cloud.google.com/speech-to-text/docs/quickstart-protocol
[13] Solicitudes síncronas: https://cloud.google.com/speech-to-text/docs/basics#synchronous-requests

(*sampleRateHertz*), y un código que representa el idioma junto a la región (*languageCode*). También hay parámetros opcionales, como la cantidad de transcripciones alternativas que se quieran obtener en la respuesta (*maxAlternatives*), si se quieren filtrar las palabras obscenas censurando la palabra con asteriscos (*profanityFilter*), y incluir información contextual adicional para el procesamiento (*speechContext*).

Por otro lado, en cuanto a los datos de audio, hay dos subcampos para elegir. El primero, contiene el audio que se quiera transcribir, proporcionado como datos binarios (*content*). El segundo, contiene un URI que dirige al contenido del audio no comprimido (*uri*), que deberá estar en Google Cloud Storage.

Del mismo modo, la respuesta a la solicitud se expresa mediante un campo, llamado *results*, que al mismo tiempo contiene el campo *alternatives*, con todas las alternativas que el usuario haya querido obtener (establecidas en el parámetro *maxAlternatives*). Cada alternativa también consta de dos diferentes campos, uno que contiene el texto transcrito (*transcript*), y otro que contiene un valor entre 0 y 1, que indica la confianza o certeza de la transcripción (*confidence*) calculada con los valores de probabilidad que se asignan a cada palabra [14].

2. Reconocimiento síncrono[15]: se envían los datos de audio a la API y se inicia una operación de larga duración, que permite sondear periódicamente los resultados de la transcripción. En este caso, la duración del audio debe ser igual o menor de 480 minutos. Esta solicitud, tiene el mismo formato que una solicitud síncrona, con la diferencia de que se creará una operación de larga duración, y no se mostrarán los resultados directamente.

   Por lo tanto, cuando el usuario haga la consulta, se mostrará una respuesta (en el campo *response*) sin resultados. A medida que se procese el audio se almacenarán los resultados y se conseguirá dicha respuesta en el campo *results*. Del mismo modo que en una solicitud síncrona, el resultado contiene el campo *alternatives*, con todas las alternativas que el usuario haya querido obtener. Cada alternativa también tiene otros dos campos, uno que contiene el texto transcrito (*transcript*), y otro que contiene un valor entre 0 y 1, que indica la confianza o certeza de la transcripción (*confidence*).

[14] Respuestas síncronas: https://cloud.google.com/speech-to-text/docs/basics#responses

[15] Solicitudes y respuestas asíncronas: https://cloud.google.com/speech-to-text/docs/basics#async-responses

3. Reconocimiento de transmisión [16]: se realiza el reconocimiento de los datos de audio mediante una solicitud de transmisión continua, por lo tanto esta dirigido al audio en tiempo real. Esto permite, que la respuesta de la solicitud aparezca mientras el usuario esté hablando, es decir, proporciona los resultados mientras se captura el audio. Estos resultados a tiempo real se corresponden a una sección concreta del audio, por lo que la respuesta del reconocimiento final puede mostrar mejoras, ya que se obtiene la interpretación del audio completo.

   Las llamadas de transmisión continua requieren varia solicitudes, a diferencia de los otros tipos de reconocimiento. Por un lado, la primera parte de la solicitud (*StreamingRecognizeRequest*) debe contener una configuración *StreamingRecognitionConfig* que consta de dos subcampos. Por un lado, está el campo obligatorio *config*, que contiene la información de la configuración del audio. Por otro lado, están los dos campos opcionales *single_utterance* y *interim_results*, que respectivamente, indican si la solicitud acabará cuando no se detecte el audio (en pausas o en silencios) y si los resultados temporales se mostrarán a medida de que se procesen, o después de que se procese todo el audio.

   Las respuestas de la solicitud se muestran en el campo *StreamingRecognitionResponse*, que al mismo tiempo, contiene dos campos [17]. El subcampo *speechEventType* consta de eventos que indican el momento en el que se completa una declaración, que se puede usar para determinar marcadores en la respuesta. Por último, el subcampo *results* incluye los resultados de la llamada, que al mismo tiempo indica la lista de transcripciones alternativas (*alternatives*), si los resultados son provisionales o finales (*isFinal*) y la estabilidad de la respuesta obtenida (*stability*). Este último, se mueve entre los valores 0.0 (inestabilidad) y 1.0 (estabilidad).

Cloud Natural Language: El servicio de Natural Language [18] permite al usuario obtener el significado y la estructura del texto, usando el aprendizaje automático. Así, los clientes podrán extraer información del texto, analizar las opiniones y las conversaciones entre personas. Dependiendo de las necesidades de cada caso particular, existen dos servicios diferentes: AutoML Natural Language y API de Natural Language.

AutoML Natural Language permite entrenar modelos personalizados de aprendizaje automático, para clasificar contenido, extraer entidades del texto y detectar opiniones. En

[16] Solicitudes de transmisión: https://cloud.google.com/speech-to-text/docs/basics#streaming-recognition

[17] Respuestas de transmisión: https://cloud.google.com/speech-to-text/docs/basics#streaming_responses

[18] Cloud Natural Language: https://cloud.google.com/natural-language

cambio, con la API de Natural Language, los modelos están previamente entrenados, y permite a los clientes hacer el análisis de opinión (conocimiento de la opinión general de una parte del texto), análisis de opiniones sobre entidades (conocimiento de opinión general de las entidades de un bloque de texto), análisis de entidades (identificación de entidades por tipo), análisis sintáctico (extracción de tokens y frases, identificación de categorías gramaticales y creación de árboles y análisis de dependencias), y la clasificación de contenido (identificación de las categorías del contenido).

Existe la posibilidad de trabajar independientemente con los dos productos, o combinar los beneficios de cada uno, dependiendo de las necesidades del usuario. Aun así, después de analizar todas las características del producto, no se ha decidido recurrir a Natural Language, eligiendo la opción de emplear otras bibliotecas para Python. Se ha visto que la mayor utilidad del servicio está relacionado con el análisis de sentimientos, cosa que no es necesaria para el caso concreto de las recetas.

Cloud Storage

Cloud Storage [19] es un recurso de Google Cloud para almacenar objetos, es decir, archivos de cualquier formato. Estos objetos se organizan y almacenan en contenedores llamados depósitos, que se asocian a un proyecto de la plataforma. Por lo tanto, cuando el usuario crea un proyecto, puede utilizar este recurso para crear uno o varios depósitos de Cloud Storage, guardar objetos, borrarlos, descargarlos cuando sea necesario, y compartirlos con diferentes miembros del proyecto.

Los depósitos [20] son contenedores básicos para guardar los datos del usuario, y todo lo que se almacene en Cloud Storage debe encontrarse en ellos. Por lo tanto, sirven para organizar los datos almacenados, pero también existen limites para su creación y eliminación. Cuando se crea un depósito, se le asigna un nombre global, una ubicación, su contenido, y una clase de almacenamiento predeterminada.

Sin embargo, los objetos [21] son ficheros o unidades individuales de datos, que se guardan en los depósitos. Los objetos pueden ser privados, accesibles a ciertos miembros del proyecto, o incluso públicos para todo Internet. Cloud Storage usa la encriptación del lado del servidor para encriptar los datos y protegerlos. Además, también gestiona el control de versiones de objetos, para evitar que los datos de reemplacen o se borren.

[19] Introducción a Storage: https://cloud.google.com/storage/docs/introduction
[20] Depósitos de Storage: https://cloud.google.com/storage/docs/key-terms#buckets
[21] Objetos de Cloud: https://cloud.google.com/storage/docs/key-terms#objects

El coste por el almacenamientos de datos es de $0.020 por GB al mes (para el almacenamiento estándar o Standard Storage) [22]. En cuanto al programa siempre gratuito, el límite del almacenamiento es de 5 GB por mes, por lo que el cobro comenzará si se sobrepasa este límite [23].

3.2.2. NLTK 3

NLTK se creó originalmente en 2001 como parte de un curso de lingüística computacional en el Departamento de Ciencias de la Información y la Computación de la Universidad de Pennsylvania. Desde entonces se ha desarrollado y ampliado con la ayuda de decenas de contribuyentes. Hoy en día, se usa en cursos en varias universidades y sirve como base para muchos proyectos de investigación [Bird et al., 2009].

Se trata de una plataforma para crear programas en Python, trabajando con datos de lenguaje natural. Proporciona interfaces fáciles de usar para más de 50 corpus y recursos léxicos como *WordNet*, junto a un conjunto de bibliotecas de procesamiento de texto para clasificación, tokenización, *stemming*, etiquetado, análisis y razonamiento semántico, y un foro de discusión [24] .

NLTK fue diseñado con cuatro objetivos principales: primero, por simplicidad, ya que proporciona un marco intuitivo, cosa que brinda a los usuarios un conocimiento práctico del procesamiento del lenguaje natural. Después, por coherencia, ya que proporciona un marco uniforme con interfaces y estructuras de datos coherentes, y nombres de métodos fáciles de adivinar. El siguiente objetivo fue la extensibilidad, para proporcionar una estructura en la que se puedan integrar fácilmente nuevos módulos de software. Por último, para la modularidad, proporcionado componentes que se pueden utilizar de forma independiente sin necesidad de comprender todas las librerías.

Los módulos que ofrece *NLTK* se pueden ver en la tabla 3.1.

Además, *NLTK* está disponible para Windows, Mac OS X y Linux, y es un proyecto gratuito, de código abierto e impulsado por la comunidad. La versión empleada en el proyecto es *NLTK* 3, versión actualizada para Python 3.5, 3.6, 3.7 o 3.8.

[22] Precio Storage: https://cloud.google.com/storage/pricing#storage-pricing
[23] Programa siempre gratuito: https://cloud.google.com/storage/pricing#cloud-storage-always-free
[24] Web nltk: https://www.nltk.org/

Tarea a realizar	Nombre del módulo NLTK
Acceder a los diferentes corpus	corpus
Procesar texto (tokenizar palabras y frases, y stemming)	tokenize, stem
Descubrimiento de la colocación	collocations
Etiquetado de Part-of-speech	tag
Tareas de machine learning	classify, cluster, tbl
Fragmentación o chunking	chunk
Analizar sintáxis	parse, ccg
Interpretación semántica	sem, inference
Métricas de evaluación	metrics
Probabilidad y estimación	probability
Aplicaciones	app, chat
Trabajo de campo lingüístico	toolbox

Tabla 3.1: Todos los módulos de la librería *NLTK*

3.2.3. Gensim

Gensim es un librería de trabajo para Python gratuita, diseñada para extraer automáticamente información semántica de documentos, de la manera más eficiente y fácil posible. Tiene como objetivo procesar textos digitales sin formato y no estructurados. Los algoritmos en *gensim*, identifican la estructura semántica de los documentos, examinando patrones de coocurrencia estadísticos de palabras dentro de un corpus de documentos de entrenamiento. Estos algoritmos no están supervisados, lo que significa que no es necesaria la participación humana, y que solo hace falta un corpus de documentos de texto sin formato [Rehurek, 2017].

Una vez que se encuentran estos patrones estadísticos, cualquier documento de texto sin formato se puede expresar como una nueva representación semántica y se podrá consultar la similitud con otros documentos. El paquete de *gensim* gira alrededor de tres conceptos básicos:

1. Corpus: se trata de una colección de documentos digitales. Esta colección se utiliza para inferir automáticamente la estructura de los documentos, sus temas, etc. Por esta razón, la colección también se denomina corpus de entrenamiento. La estructura latente inferida se puede utilizar posteriormente para asignar temas a nuevos documentos, que no aparecieron en el corpus de entrenamiento. Además, no se requiere intervención humana (como etiquetar los documentos a mano).

2. Vectores: en el modelo de espacio vectorial, cada documento está representado por

una serie de características. Por ejemplo, una sola característica puede considerarse como un par de pregunta y respuesta representada por una tupla. Por lo tanto, la representación de un documento se convierte en una serie de pares de tuplas, que contienen la pregunta-respuesta sobre una característica. Si las preguntas son las mismas para cada documento, mirando dos vectores (que representan dos documentos), se podrán sacar conclusiones como que parecido tienen varios documentos comparados.

3. Modelo: un modelo es una transformación de una representación de documento a otra (o, en otras palabras, de un espacio vectorial a otro). Tanto la representación inicial como la de destino siguen siendo vectores, y solo difieren en las preguntas y respuestas. Esta transformación se aprende automáticamente del corpus de entrenamiento, sin supervisión humana, y suponiendo que la representación del documento final sea más compacta y más útil que la inicial.

Gensim ofrece diferentes modelos para diferentes tareas, como *Word2Vec*, *Doc2Vec*, *FastText*, y algoritmos como *Latent Semantic Analysis* (LSA), *Latent Dirichlet Allocation* (LDA) o *Random Projections*.

Gensim, está disponible para Windows, Mac OS X y Linux, con las siguientes dependencias:

- Python $\geq$ 2.5
- NumPy $\geq$ 1.3
- SciPy $\geq$ 0.7

Word2vec

Word2vec fue creado y publicado en 2013 por un equipo de investigadores dirigido por Tomas Mikolov en Google. Se trata de un proceso no supervisado, en la que a partir de texto sin formato se crean *word embeddings*. *Word embedding* es un conjunto de técnicas de modelado de lenguaje para mapear palabras en un vector, de forma que cada vector numérico representa una palabra [Ling et al., 2015].

El objetivo es maximizar la probabilidad de que las palabras se predigan a partir de su contexto y viceversa. Para ello, se preentrena la matiz de proyección $W \in \Re^{d \times |V|}$, donde d es la dimensión del *embedding* con el vocabulario V. Se definieron dos modelos diferentes, el modelo de skip-gram y el modelo continuous bag-of-words (CBOW).

Por un lado, la función objetivo del modelo skip-gram a maximizar es la probabilidad de predicción de palabras contextuales dada la palabra central. Formalmente, dado un documento de palabras T, se quiere maximizar:

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

Donde c es un hiperparámetro que define la ventana (window) de palabras de contexto. Para obtener la probabilidad de salida $p\left(w_o|w_i\right)$, el modelo estima una matriz $O \in \Re^{|V|\times d_w}$ que mapea los *embeddings* r_{w_i} en un vector o_{w_i} de dimensión $|V|$. Entonces, la probabilidad de predecir la palabra w_o dada la palabra w_i se define como:

$$p\left(w_o|w_i\right) = \frac{e^{o_{w_i}(w_o)}}{\sum_{w\in V} e^{o_{w_i}(w)}}$$

Esto se conoce como el objetivo *Softmax*. Sin embargo, para vocabularios más grandes es ineficiente calcular o_{w_i} , ya que esto requiere el cálculo de una multiplicación de matrices $|V| \times d_w$.

Por otro lado, el modelo CBOW predice la palabra central w_o dada una representación de las palabras del contexto $w_{-c},...,w_{-1},w_1,w_c$. Así, el vector de salida $o_{w_{-c},...,w_{-1},w_1,w_c}$ es obtenido del producto de la matriz $O \in \Re^{|V|\times d_w}$ con la suma de *embeddings* de las palabras de contexto $\sum_{-c\leq j\leq c, j\neq 0} r_{w_j}$.

Como se puede observar, en ambos métodos, el orden de las palabras de contexto no influye en el resultado de la predicción. Aunque estos métodos pueden encontrar representaciones similares para palabras semánticamente similares, es menos probable que sean representaciones basadas en las propiedades sintácticas de las palabras. En la figura 3.2, se pueden ver las arquitecturas de los dos modelos de *word2vec*.

3.2.4. Scikit-learn

Scikit-learn es un módulo de Python para el aprendizaje automático construido sobre SciPy. El proyecto fue iniciado en 2007 por David Cournapeau como un proyecto de Google Summer of Code, y desde entonces han contribuido varios voluntarios [25]. El módulo integra una gama de algoritmos de aprendizaje automático para problemas supervisados y no supervisados de mediana escala. Este paquete se centra en llevar el aprendizaje automático a los no especialistas, haciendo hincapié en la facilidad de uso, el rendimiento, la documentación y la coherencia de API.

[25] Implementación de Scikit-learn: https://github.com/scikit-learn/scikit-learn

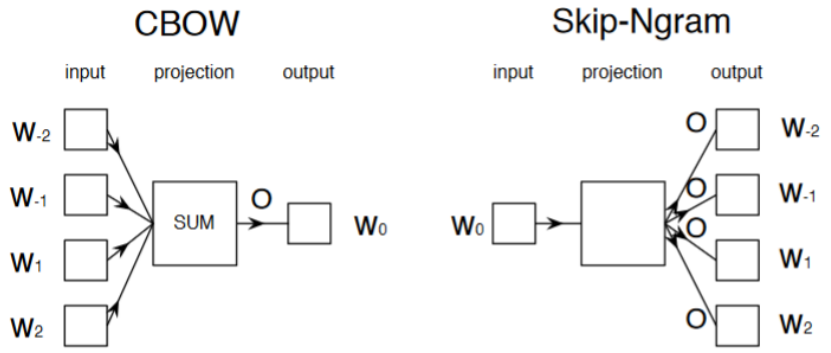


Figura 3.2: Arquitecturas del modelo CBOW y skip-Ngram de word2vec

Este módulo requiere las siguientes dependencias:

- Python $\geq$ 3.6
- NumPy $\geq$ 1.13.3
- SciPy $\geq$ 0.19.1
- joblib $\geq$ 0.11
- threadpoolctl $\geq$ 2.0.0

TSNE

TSNE trata de un módulo que ofrece la librería *Scikit-learn*, principalmente para poder visualizar vectores de grandes dimensiones, utilizando la técnica t-SNE (t-Distributed Stochastic Neighbor Embedding). Se trata de una técnica de reducción de dimensionalidad no lineal en la que los datos interrelacionados de alta dimensión se mapean en datos de baja dimensión, por ejemplo, de dos variables, mientras se conserva la estructura significativa (relación entre los puntos de datos en diferentes variables) de datos originales de alta dimensión [Bedre, 2020].

Los datos reducidos representan la estructura de datos de alta dimensión y son fáciles de visualizar en un diagrama. t-SNE se utiliza principalmente con fines de visualización y no para análisis cuantitativos detallados. Hay que tener en cuenta que es un método estocástico que produce representaciones ligeramente diferentes si se ejecuta varias veces. Estos resultados diferentes podrían afectar los valores numéricos, pero no afectan al *clustering* o agrupación de los puntos que se aprecia de forma cualitativa.

4. CAPÍTULO

Desarrollo del proyecto

El proyecto consta de tres fases principales: en la primera, se ha generado la base de datos de audio eligiendo el tema a tratar, los sujetos, la duración aproximada de cada archivo y el almacenamiento de estos. En la segunda fase, se ha hecho la transcripción de dicho audio a texto, utilizando los servicios que ofrece la plataforma de Google Cloud. Por último, en la tercera fase, se han empleado varias librerías de Python para el procesamiento del lenguaje natural, preparando el texto para la extracción de información, y obteniendo dos modelos diferentes que representan el vocabulario de las recetas.

Después de que se realicen las tres fases, se podrán hacer varios tipos de consultas para obtener información sobre diferentes aspectos de las recetas.

4.1. Diseño

Por lo tanto, mediante las tres fases principales (creación de la base de dados, transcripción y procesamiento del lenguaje natural) se podrá extraer el conocimiento adquirido a través de las descripciones de voz. El diseño general del proyecto se puede visualizar de la manera que muestra la figura 4.1.

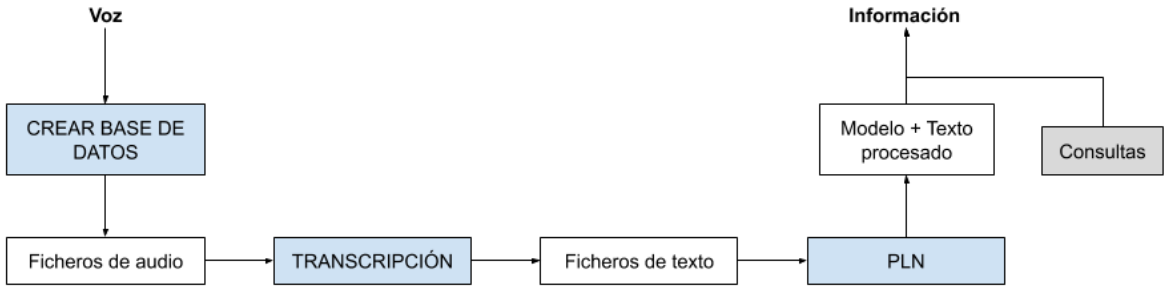


Figura 4.1: Diseño general del desarrollo del proyecto

4.2. Fases del proyecto

4.2.1. Generar la base de datos

El primer paso ha sido crear una base de datos de audio que describa varios procesos, para después extraer la información de ellos. El tema elegido ha sido la cocina, específicamente la recetas, un ámbito sencillo en el que las personas expresan su conocimiento, en este caso en formato de audio, describiendo los procesos de fabricación. El diseño de esta primera fase se puede ver en la figura 4.2.

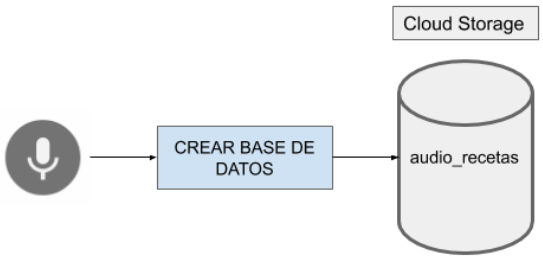


Figura 4.2: Diseño de la primera fase: creación de la base de datos

Se han empleado 39 recetas, y todas han sido seleccionadas del libro 1069 recetas de de cocina de Karlos Arguiñano [Arguiñano, 2011]. Son recetas cortas, que duran desde 40 a 95 segundos, y están divididas en dos partes: los ingredientes (donde se describen los alimentos necesarios y sus correspondientes cantidades), y la elaboración (donde se expresan los pasos a seguir con los ingredientes especificados). El tiempo total de audio es de 40 minutos y 35 segundos, y todos los archivos tienen el formato WAV (Waveform Audio Format).

En cuanto a la ubicación de los archivos, se ha empleado Cloud Storage, uno de los recursos de almacenamiento ofrecidos por Google Cloud. Para ello, se han tenido que seguir los pasos de iniciación de la plataforma.

Primero, se ha creado un proyecto llamado «TranscripionTexto» de Cloud Console, con ID transcripciontexto y un número de proyecto que Google Cloud ha asignado, tal y como muestra la figura 4.3.

Figura 4.3: Creación del proyecto «TranscripionTexto» en Google Coud

Después, se ha creado una cuenta de servicio para el proyecto, ya que Google Cloud necesita verificar que el cliente se trata de una persona. Estos tipos de cuentas usan una aplicación o una instancia de máquina virtual (VM) para hacer las llamadas a las APIs autorizadas, es decir, obtiene los permisos necesarios para que el cliente pueda acceder a los recursos que necesite.

Para crear una cuenta de servicio, se ha facilitado una dirección de correo electrónico, que es única a la cuenta. Además, cada cuenta de servicio esta asociada con dos conjuntos de pares de claves RSA públicas y privadas para autenticarse en Google Cloud. Existen dos tipos de cuentas de servicio, las que son administradas por el usuario, y las cuentas de servicio predeterminadas. En este caso, se ha recurrido a la primera opción, siendo así, el mismo usuario el responsable de administrar y proteger las diferentes cuentas que puedan acceder al proyecto [1].

En el proyecto «TranscripcionTexto» se ha creado una cuenta única, con el correo electrónico anearburua@transcripciontexto.iam.gserviceaccount.com. A esta cuenta, se le ha adjudicado una ID de clave única (creada por la plataforma), y el rol de propietario.

Una vez completados estos pasos, ya se ha podido crear el segmento «audio-recetas» en Cloud Storage, en el que se han guardado todos los archivos de audio, como muestra la figura 4.4.

[1] Creación de las cuentas de servicio: https://cloud.google.com/iam/docs/creating-managing-service-accounts#creating

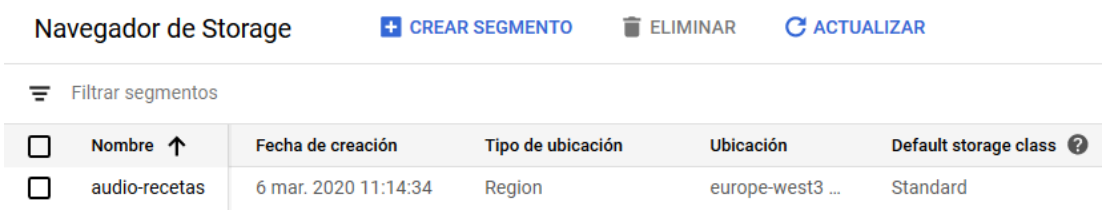


Figura 4.4: Creación del segmento audio-recetas de Cloud Storage

Para que posteriormente se pueda acceder a los archivos, se ha creado el miembro *allUsers* (figura 4.5), que hace referencia a todos los usuario de internet, para que puedan ver el contenido del depósito.

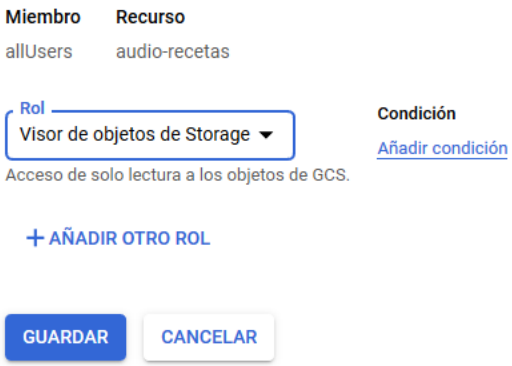


Figura 4.5: Permitir acceso al deposito de Google Cloud

En esta fase del proyecto, no se ha tenido que implementar ningún tipo de código.

4.2.2. Transcripción del audio

Una vez creada la base de datos, el siguiente paso ha sido la transcripción de audio a texto. Como se indicaba en el enunciado inicial, se han analizado las capacidades de análisis de voz y lenguaje natural que proporciona la nube de Google. La herramienta a utilizar ha sido, por lo tanto, Google Cloud, y se ha centrado en el uso de la API de aprendizaje automático, ya que ofrece el servicio de transcripción de audio a texto. El diseño de esta segunda fase se puede ver en la figura 4.6.

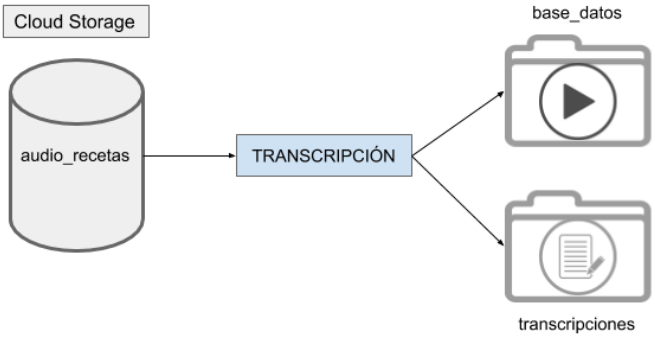


Figura 4.6: Diseño de la segunda fase: transcripción de los archivos de audio a texto

Primeros pasos

De las herramientas que se proporcionan en la API de aprendizaje automático, se ha utilizado el servicio Speech-to-Text, ya que posibilita la transcripción del audio a texto. Para el uso de este servicio, se ha acabado de configurar el proyecto previamente creado (TranscripionTexto), habilitado la API de Cloud Speech-to-Text, como la figura 4.7 indica.

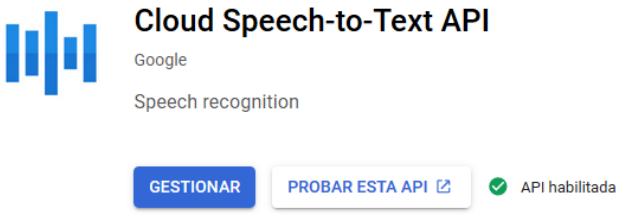


Figura 4.7: Habilitación de la API de Cloud Speech-to-Text

Después, se ha descargado la clave privada JSON vinculada a la cuenta creada, como se muestra en la figura 4.8. De este modo, se ha descargado un archivo para poder acceder a las API y los servicios habilitados de la cuenta, ya que contiene la clave de la cuenta de servicio.

A continuación, se ha configurado la variable de entorno *GOOGLE_APPLICATION_CREDENTIALS* en la ruta del archivo JSON que contiene la clave de la cuenta de servicio. Esta variable, solo se aplica a la sesión actual de terminal, por lo que se deberá configurar cada vez que se abra un sesión nueva.

Posteriormente, se ha instalado e inicializado el SDK de Google Cloud. Se trata de un

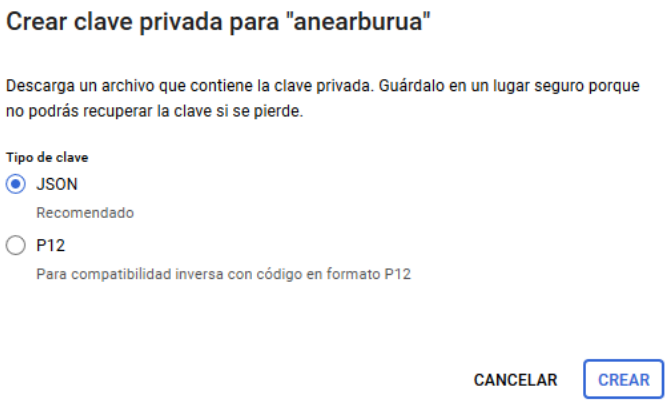


Figura 4.8: Descarga de la clave privada JSON de cuenta Cloud

conjunto de herramientas para usar en la administración de aplicaciones y recursos que se encuentren en Google Cloud Platform. En el caso de Linux, el SDK requiere Python, y las versiones compatible son las 3.5 a 3.7, y 2.7.9 o posteriores. Una vez instalado el paquete, se ha extraído el contenido del archivo, y se ha ejecutado para inicializar el SDK usando el comando *gcloud init* [2].

En este punto, los servicios de Speech-to-Text están habilitados para el proyecto, por lo que ya se podría comenzar a hacer las solicitudes de reconocimiento de voz.

Se ha decidido hacer la transcripción usando las bibliotecas cliente de Speech-to-Text, ya que es una forma sencilla de obtener la respuesta, tan solo instalando la biblioteca. Entre los tres tipos de reconocimientos que ofrece la plataforma (reconocimiento síncrono, asíncrono y en tiempo real), se ha elegido el reconocimiento síncrono, ya que es el método más simple para realizar la transcripción de los datos. Así, una vez que se procese y reconozca el audio completo, se muestra la respuesta directamente.

A través de la API, se puede procesar el audio hasta 1 minuto. En la base de datos, existen archivos de audio de entre 40 a 95 segundos, por lo que no es posible hacer este tipo de llamadas de reconocimiento con todos. Por lo tanto, para el audio que tenga la duración de más de un minuto, se utilizará el reconocimiento asíncrono.

Para realizar la llamada, el audio se puede encontrar en un archivo local o remoto. En este caso, la base de datos se encuentra en Google Cloud Storage, por lo que se ha podido realizar el reconocimiento sin necesidad de enviar el contenido del archivo de audio en el

[2]Inicialización del SDK en Linux: https://cloud.google.com/sdk/docs/quickstart-linux#initialize_the_sdk

cuerpo de la solicitud.

Implementación

La fase de la transcripción de audio consta de cinco ficheros incluidos en la carpeta «transcripcion» del proyecto: transcripciontexto.json, dependencias.sh, init_transcripcion.py, main_transcripcion.py, y transcribir_audio.py.

transcripciontexto.json: Es el archivo que contiene la clave privada JSON vinculada a la cuenta de Google Cloud. Gracias a él, se puede acceder a las API y los servicios habilitados de dicha cuenta. En cada nueva sesión de terminal, será necesario exportar la ubicación de las credenciales, ya que por el contrario no se podrá acceder al proyecto de la plataforma.

dependencias.sh: Es el fichero que instala las bibliotecas necesarias para la ejecución de los programas relacionados con la transcripción. En este caso, las dos bibliotecas necesarias son *google-cloud-speech* y *google-cloud-storage*.

init_transcripcion.py: Importa las bibliotecas necesarias para la ejecución de los otros dos ficheros .py, incluyendo las librerías de Google Cloud Speech-to-Text y Cloud Storage.

main_transcripcion.py: Contiene el programa principal que accede a la base de datos, consigue las transcripciones del audio, y guarda el texto correspondiente en cada fichero de formato .txt.

- transcripcion_principal()

  Primero, crea las carpetas «transcripciones» y «base_datos» que guardarán las transcripciones de audio en formato .txt, y los archivos de audio originales con el formato .wav, respectivamente. Después, se crea un cliente de Cloud Storage para poder acceder al depósito «audio-recetas» en el que se guardan los archivos de audio. Por cada archivo de audio, se consigue el nombre de la receta, y se descarga el audio a un fichero local con su respectivo nombre, que se guardará en la carpeta «base_datos».

A continuación, se calcula la duración del audio, ya que es significativa, porque las consultas que se realizan en la API dependerán de ella (llamada de reconocimiento síncrona si dura menos de un minuto, y por el contrario, asíncrona). Por último, se hace la llamada al método *reconocer_audio*, para que se encargue de hacer la transcripción de un archivo concreto con los datos obtenidos. Todas las transcripciones creadas por el método auxiliar se guardarán en la carpeta «transcripciones».

transcibir_audio.py: Contiene el programa *reconocer_audio*, el cual dadas la ubicación del audio, la duración y el nombre del archivo de destino, transcribe el audio a texto.

- reconocer_audio(p1,p2,p3)

  Primero, crea un cliente de Cloud Speech-to-Text para poder acceder al servicio de transcripción ofrecido por la plataforma. Después, se configuran los parámetros para la consulta, entre ellos la tasa de muestreo en hercios de 44100 (*sample_rate_hertz*) adecuada para archivos de audio .wav, el lenguaje del audio en español (*language_code*) y la codificación de los datos de audio enviados LINEAR16 (*encoding*).

  Posteriormente, se define la ubicación URI del archivo a transcribir, especificada en el parámetro p1. Luego, se analiza la duración del audio que indica el parámetro p2. En el caso de ser menor de un minuto, se hace la llamada de reconocimiento síncrona *recognize* con la configuración y el audio. En el caso contrario, se hace la llamada de reconocimiento asíncrona *long_running_recognize* con la misma configuración y audio.

  La respuesta que se obtiene después de hacer la consulta puede tener uno o varios resultados, ordenados dependiendo de la confianza o certeza de la transcripción. La respuesta más probable será la primera, por lo que se guardará la transcripción palabra por palabra, escribiéndola en un fichero .txt con la ubicación y nombres especificados en el tercer parámetro p3.

4.2.3. Procesamiento del lenguaje natural

Una vez obtenidas todas las transcripciones, la tercera y última fase consiste en procesar el texto obtenido. Para ello, se han investigado varios aspectos del procesamiento del lenguaje natural, y se han analizado diferentes herramientas para su desarrollo. El diseño de la fase se puede ver en la figura 4.9.

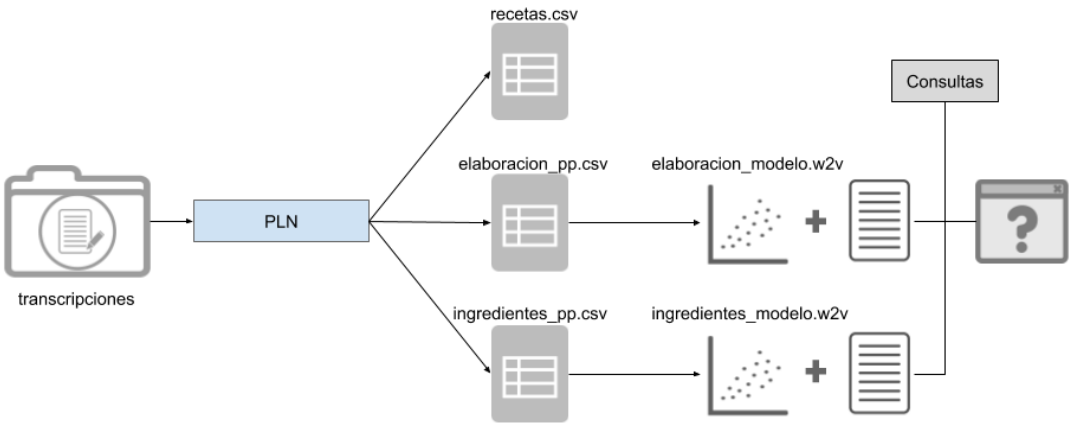


Figura 4.9: Diseño de la tercera fase: procesamiento del lenguaje natural

Las principales bibliotecas empleadas para esta parte del proyecto, han sido *NLTK*, *gensim* y *scikit-learn*. Además, se han empleado otras librerías de Python para ayudar a la implementación de los programas.

Primeros pasos

En este caso, no se ha requerido ningún paso previo en especial, ya que esta tercera fase comienza directamente analizando el texto de las transcripciones. Por lo tanto, con la nueva carpeta que se ha creado en la fase anterior es suficiente.

Implementación

La fase del procesamiento del lenguaje natural consta de siete ficheros incluídos en la carpeta «nlp» del proyecto: stanford-postagger-full-2017-06-09.zip, dependencias.sh, init_nlp.py, main_nlp.py, process.py, train.py, y consultations.py.

Stanford-postagger-full-2017-06-09.zip: Se trata de una carpeta comprimida que contiene los modelos necesarios para la sección de etiquetado POS o *Part-Of-Speech tagging*. Estos modelos se encargan de leer el texto que en este caso, está escrito en español, y asigna una etiqueta a cada palabra (o token) para obtener las características de ella.

Se trata de la versión 3.8.0, y incluye modelos en seis diferentes idiomas: árabe, chino, inglés, francés, alemán y español. En esta misma versión se incluyeron nuevos modelos de español y francés, por lo que es el más indicado para el caso de las recetas en español.

dependencias.sh: Es el fichero que instala las bibliotecas necesarias para la ejecución de los programas relacionados con el procesamiento del lenguaje.

init_nlp.py: Importa las bibliotecas necesarias para la ejecución de los otros ficheros .py. La mayoría de ellas pertenecen a distintos módulos de la librería *NLTK*, ya que es la librería principal con la que se desarrolla esta fase del proyecto.

También se importan las librerías *gensim* y *sklearn* (*scikit-learn*) para crear y visualizar los resultados de los corpus que se procesan. Aun así, también se emplean otras librerías que ayudan con la implementación del proyecto, como *itertools*, *zipfile*, *os*, *pandas*, *numpy*, *seaborn* y *matplotlib*.

main_nlp.py: Contiene el programa *cargar_ficheros* para poder crear una estructura de datos válida para el procesamiento de las recetas, y el programa principal llamado *nlp_principal*. Este último, se encargará de llamar a todas las funciones auxiliares para completar los pasos del procesamiento del lenguaje, y obtener información acerca de las recetas. Además, el fichero contiene el método *menu*, que crea un menú de opciones para extraer la información del texto y realizar las consultas que quiera hacer el usuario. Por último, los programas *elegir_opcion*, *elegir_receta*, *elegir_ingrediente* y *ver_recetas* son métodos auxiliares para el menú de opciones.

- nlp_principal()

  Es es programa principal en cuanto al procesamiento del lenguaje, ya que es el encargado de llamar a todas las funciones para esta tarea. Antes de comenzar, se crea una estructura de datos *DataFrame* con todas las transcripciones obtenidas en la fase anterior. El texto se analiza para dos de las tres partes principales de las recetas: los ingredientes y la elaboración. Por lo tanto, los pasos seguidos para este procesamiento se realizan independientemente para esas dos partes:

  Primero, se tokeniza el texto a través de *tokenizar*, después se etiquetan estos tokens para obtener la categoría da cada palabra mediante *etiquetado*. A continuación, se fragmenta el texto para que cada parte creada tenga un sentido, creando así conjuntos de tokens con *fragmentar*. Luego, se preprocesa el texto mediante *preprocesar*, preparándolo para crear el corpus definitivo con el que se creará un modelo de vectores de palabras (con los métodos *obtener_corpus* y *entrenar*).

Receta	Ingredientes	Elaboración
albóndigas con champiñones	2 huevos sal pimienta 15 gramos de migas de pan remojadas en leche ...	mezcla bien la carne con los huevos la sal la pimienta y la miga de pan forma las albóndigas ...

Tabla 4.1: *DataFrame* principal correspondiente a la fila «albóndigas con champiñones» con el nombre, ingredientes, y elaboración

Una vez seguidos los pasos tanto con los ingredientes como con la elaboración, se llama a la función de *menu*, dejando que el usuario consulte la información necesaria que se ha preparado gracias al procesamiento anterior.

- cargar_ficheros()

  Para comenzar, accede a la carpeta «transcripciones» previamente creada, que a la vez se encuentra el la carpeta «transcripcion» del proyecto. Ya que todos los archivos de texto que describen las recetas se encuentran aquí, abre los ficheros, y obtiene el contenido de cada una de ellas.

  Cada archivo de texto se compone de tres partes diferentes: por un lado, está el nombre de la receta, que se describe al principio. Después, al leer la palabra «ingredientes», se numerarán los ingredientes necesarios para dicha receta. Por último, la palabra «elaboración» definirá el final de los ingredientes, para pasar a la descripción de la receta, y los pasos que habrá que seguir para cocinarla.

  Así, se crea una única estructura de datos tipo *DataFrame*, que guardará toda esta información en tres columnas principales. La columna «Receta» guarda todos los nombres de las recetas de la base de datos, mientras que la columna «Ingredientes» contiene los ingredientes necesarios para cada receta, y «Elaboración» describe sus pasos a seguir. Por lo tanto, se obtiene y devuelve la estructura en la que cada receta pertenece a una fila, y sus características están divididas en las tres columnas mencionadas. Por ejemplo, la receta «albóndigas con champiñones» se representa de la manera que describe la tabla 4.1.

  Además, para que la información sea más accesible, se crea el archivo «recetas.csv», con la misma información de la estructura *DataFrame*, descrita en una tabla (ver Anexo B).

- menu(p1,p2,p3,p4,p5,p6,p7)

  Crea el menú de opciones para extraer la información necesaria después de procesar

las recetas. La función imprime las diferentes consultas que se pueden realizar. Por lo tanto, el usuario deberá elegir una (a través del método *elegir_opcion*), y el programa se encargará de hacer las llamadas a los métodos de estas consultas.

La primera opción, consiste en introducir el número de una receta para obtener los nombres de las recetas similares según su elaboración. Por lo tanto, se llama a los métodos *elegir_receta* con el parámetro p1 y *elaboracion_similar* con los parámetros p1 (el *DataFrame* original), el número de la receta, y p6 junto a p7 que guardan el modelo y corpus utilizado para las elaboraciones. La segunda opción, también consiste en introducir un número de receta, para buscar recetas similares en cuanto a los ingredientes que se usan. Por lo tanto, se llama a la misma función *elegir_receta*, y al método *ingrediente_similar* con los parámetros, p1, el número de la receta, y p3 junto a p4 (que guardan el modelo y corpus utilizado para los ingredientes).

La tercera opción trata de conseguir las recetas elaborables dependiendo de los ingredientes que elija el usuario. Para ello, se obtiene la lista de ingredientes con *elegir_ingrediente*, y se llama al método *recetas_posibles* con esta misma lista, junto al parámetro p2 que contiene el *DataFrame* preprocesado de los ingredientes. En cambio, la cuarta opción consiste en obtener las partes de las elaboraciones que se puedan hacer con un cierto ingrediente, junto al nombre de la receta de la elaboración. Por eso, se llama a la misma función *elegir_ingrediente*, y *utilidad_ingrediente* con el parámetro p5 que contiene el *DataFrame* de las elaboraciones preprocesadas.

La quinta opción, simplemente trata de imprimir todas las transcripciones de las recetas guardadas en p1, por ello, llama a la función *ver_recetas* con este mismo parámetro. Por último, la sexta opción hace acabar la ejecución del menú.

- elegir_opcion()

  Obtiene la opción que el usuario quiera consultar, simplemente leyendo la información de la terminal. Después de imprimir el mensaje de aviso «elige una opción», el usuario tendrá que introducir un número del 1 al 6. Este número, se devuelve para que la función del menú llame a los métodos necesarios según la opción elegida.

- elegir_receta(p)

  Consigue el número de la receta elegida por el usuario, para las consultas de obtener ingredientes similares o elaboraciones similares de una receta. Para ello, imprime el mensaje «Introduce el número de la receta» advirtiendo al usuario, y lee este número de la terminal. Si la información introducida no corresponde a un número,

se imprimirá un mensaje de error. Además, el número introducido tendrá que ser menor o igual de la cantidad de recetas, por lo que también se tendrá en cuenta, y se imprimirá otro mensaje de error si no se cumple.

- elegir_ingrediente(p)

  Obtiene el ingrediente o los ingredientes que el usuario elija. Para ello, simplemente imprime un mensaje para que el usuario introduzca los ingredientes, y consigue los datos para transmitirlos a las funciones de las consultas necesarias. El valor del parámetro p, varía según la opción que haya elegido el usuario: en el caso de que se quieran encontrar las recetas elaborables con ciertos ingredientes, p tendrá el valor 3, y se leerán los ingredientes introducidos como una lista (ya que puede haber más de uno). Sin embargo, si se quiere obtener la utilidad de un ingrediente concreto, p tendrá el valor 4, por lo que el usuario solo podrá introducir un ingrediente.

- ver_recetas(p)

  Imprime los nombres de las recetas de la estructura de datos definida por p. Junto a este nombre, obtiene el número de la receta para que el usuario pueda identificarlas y realizar las consultas necesarias con las elegidas.

process.py: Contiene los programas necesarios para procesar el texto transcrito. Entre ellos, el programa *tokenizar* crea tokens de la recetas, es decir, separa las frases en palabras. Para el etiquetado de los tokens, el programa *etiquetado* accede al modelo para español de *Stanford POS Tagger*, y clasifica los diferentes tipos de palabras. Además existen los métodos auxiliares *stanford_nltk* y *extraer_tagger* para ayudar con el etiquetado.

En cambio, para la sección de fragmentación, el programa *fragmentar* crea conjuntos de tokens que ayudan a la comprensión del texto, ya que agrupa las palabras en unidades con sentido. El método auxiliar *arbol_a_lista* también pertenece a la parte de la fragmentación. Por último, para la parte del preprocesamiento, existen los métodos *preprocesar* y *devolver_pos*, que limpian en texto de palabras redundantes y obtiene la raíz de los tokens, para facilitar la comprensión del lenguaje.

- tokenizar(p1,p2)

  A partir de la estructura de datos tipo *DataFrame* que describe el parámetro p1, tokeniza la columna deseada, es decir, separa el texto en palabras (o tokens), y

Receta	token
albóndigas con champiñones	[‘2’, ‘huevos’, ‘sal’, ‘pimienta’, ‘15’, ‘gramos’, ‘de’, ‘migas’, ‘de’,‘pan’, ‘remojadas’, ‘en’, ‘leche’, . . .]

Tabla 4.2: *DataFrame* correspondiente a la fila «albóndigas con champiñones» con las columnas de nombre de la receta y los token

crea una lista con ellas. Las dos posibles columnas para tokenizar son la de «Ingredientes» o «Elaboración», y dependerá del valor del parámetro p2 que columna se tokenizará.

Por lo tanto, para esta tokenización, el programa lee la columna deseada y crea las listas de tokens para todas las recetas. Para ello, la librería *NLTK* ofrece la función *word_tokenize*, que crea las listas automáticamente. Estas nuevas listas se guardan en un nuevo *DataFrame* junto al nombre de las recetas. Así, la nueva estructura de datos guarda el nombre en la columna «Receta», y la lista de los tokens (ya sean de los ingredientes o de la elaboración) en la columna «token». Por ejemplo, la receta «albóndigas con champiñones» para la columna «Ingredientes» representa en la tabla 4.2.

- etiquetado(p)

  Teniendo la estructura *DataFrame* creada en la parte de la tokenización que se indica en el parámetro p, asigna una etiqueta a cada token de la lista de cada receta. El etiquetador o *tagger* usado es el *Stanford Part-Of-Speech Tagger*, que está descargado y comprimido en la carpeta «nlp» del proyecto.

  Para comenzar con el etiquetado, se extrae el *tagger*, y se definen la ubicación del archivo .jar y el modelo en español. Estas dos ubicaciones son necesarias para definir el etiquetador *spanish_tagger* con el que se realizará esta sección del proyecto, llamando al método *StanfordPOSTagger* que ofrece la biblioteca de *NLTK*. Una vez conseguido el *tagger*, se asignan las etiquetas correspondientes a todos los tokens, creando así, una lista que contiene tuplas con el token y su correspondiente etiqueta o *tag*: (‘token’, ‘TAG’). Esta operación se realiza para todas las recetas del *DataFrame*.

  Esta información, se guarda en el mismo *DataFrame* del parámetro p1, simplemente añadiendo una nueva columna llamada «token_POS». Así, la nueva estructura de datos guardará el nombre de la receta en «Receta», la lista de los tokens en la columna «token», y la lista de tuplas con los anteriores tokens y sus correspondiente

Receta	token	token_POS
albóndigas con champiñones	[‘2’, ‘huevos’, ‘sal’, ‘pimienta’,‘15’, ‘gramos’, ‘de’, ‘migas’, ‘de’,‘pan’, ‘remojadas’, ‘en’, ‘leche’, . . .]	[(’2’, ’NUM’), (’huevos’,’NOUN’), (’sal’, ’NOUN’), (’pimienta’, ’NOUN’), (’15’, ’NUM’), (’g’,’UNI’), (’de’, ’ADP’), (’migas’, ’NOUN’), (’de’, ’ADP’), (’pan’,’NOUN’), (’remojadas’, ’ADJ’), (’en’, ’ADP’), (’leche’, ’NOUN’), . . .]

Tabla 4.3: *DataFrame* correspondiente a la fila «albóndigas con champiñones» con las columnas de nombre, token, y etiquetas

etiquetas en «token_POS». Por ejemplo, la receta «albóndigas con champiñones» para los ingredientes representa como en la tabla 4.3.

- extraer_tagger()

  Extrae el etiquetador de Stanford del fichero Stanford-postagger-full-2017-06-09.zip. Una vez extraído, borra el fichero .zip original, y deja la nueva carpeta en el mismo directorio, con todos los modelos disponibles.

- stanford_nltk(p)

  Stanford Part-Of-Speech Tagger hace un mejor trabajo al etiquetar todo tipo de palabras del texto, en comparación con el corpus *cess_esp* que ofrece la librería *NLTK*. La principal diferencia es que el primero reconoce todo tipo de verbos especificando el tiempo verbal de estos, mientras que el corpus *cess_esp* no hace el trabajo de un modo tan exhaustivo, y no etiqueta los verbos como debería.

  El problema es, que la nomenclatura del etiquetado no es compatible con la librería *NLTK*, con la que gran parte del proyecto de desarrolla. Por lo tanto, este método convierte las etiquetas propias del modelo utilizado, en etiquetas universales de *NLTK*. Aun así, aunque los modelos de Stanford hagan un buen trabajo, se han definido varias excepciones que no se etiquetaban de forma correcta, y hacían que la comprensión del texto se complicara.

  Una vez hecho esto, se analizan las etiquetas asignadas para hacer la conversión de la lista definida en el parámetro p. Primero, se crea una nueva etiqueta (que no dispone *NLTK*) llamada ‘UNI’, y se refiere a los nombres que indican unidades (como gramos o kilogramos). Después, por cada tupla de la lista (en la que se define el token y su etiqueta), se analiza la etiqueta asignada, y se convierte en una universal.

  Las etiquetas pueden ser ‘NOUN’ para nombres, ‘NUM’ para números, ‘ADJ’ para

Stanford POS tagger	Tags nltk
[('2', 'z0'), ('huevos', 'nc0p000'), ('sal', 'nc0s000'), ('pimienta', 'aq0000'), ('15', 'z0'), ('gramos', 'nc0p000'), ('de', 'sp000'), ('migas', 'nc0p000'), ('de', 'sp000'), ('pan', 'nc0s000'), ('remojadas', 'aq0000'), ('en', 'sp000'), ('leche', 'nc0s000'), . . .]	[('2', 'NUM'), ('huevos', 'NOUN'), ('sal', 'NOUN'), ('pimienta', 'NOUN'), ('15', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), ('migas', 'NOUN'), ('de', 'ADP'), ('pan', 'NOUN'), ('remojadas', 'ADJ'), ('en', 'ADP'), ('leche', 'NOUN'), . . .]

Tabla 4.4: Antes y después de transformar el etiquetado POS

adjetivos, 'ADP' para preposiciones, 'ADV' para adverbios, 'CONJ' para conjunciones, 'DET' para determinantes, 'PRON' para pronombres, 'VERB' para verbos, '.' para signos de puntuación, y 'X' para palabras que no reconozca el corpus.

Después de convertir las etiquetas, se crean tuplas nuevas y se añaden a una lista que se devuelve, consiguiendo así, las etiquetas definitivas para lodos los tokens. Por ejemplo, el antes y el después de las etiquetas para la receta «albóndigas con champiñones» se puede ver en la tabla 4.4.

- fragmentar(p1,p2)

  Es el proceso que crea los fragmentos o *chunks* del texto, también llamado *chunking*. Cada *chunk*, será una unidad con sentido dentro de la receta, y servirá para que después, se pueda extraer la información necesaria de ella. En este caso, el parámetro p1 indica el *DataFrame* con el que se trabajará, y p2 define la columna que se quiera fragmentar.

  Esta distinción de columnas es necesaria, ya que como se ha mencionado, las recetas originales se dividían en tres partes: el nombre, ingredientes, y elaboración. Tanto los ingredientes de la columna «Ingredientes» como la elaboración de la columna «Elaboración» tienen diferentes estructuras. Mientras que en la primera solo se enumeran los ingredientes necesarios, en la segunda aparecen los pasos de la receta que están definidos por verbos.

  Por lo tanto, para la sección de fragmentación, se han definido dos gramáticas diferentes, dependiendo del valor de la columna (parámetro p2 de la función). Si se trata de los ingredientes, los fragmentos tendrán una de las siguientes estructuras:

  1. Un número o determinante, junto a un nombre o una unidad. Después (puede

que no haya, o haya más de uno) una preposición junto a un nombre. Por último, que también será opcional, un adjetivo junto a un nombre también opcional.

(<NUM>|<DET>)(<NOUN>|<UNI>)((<ADP><NOUN>)?)*

(<ADJ>(<ADP><NOUN>)?)?

2. Un nombre, junto a una preposición y otro nombre

   <NOUN><ADP><NOUN>

3. Un nombre, junto a un adjetivo opcional

   <NOUN><ADJ>?

En cambio, si se trata de la elaboración, los fragmentos se dividirán entorno a los verbos, ya que separan diferentes partes de la elaboración. Así, cada fragmento describirá un paso. La estructura, por lo tanto, será un verbo junto a cualquier tipo de palabra, mientras no se trate de un verbo:

<VERB>(<NOUN>|<ADJ>|<ADP>|<ADV>|<CONJ>|<DET>|<NUM>|<PRON>|<.>|<X>)*

Una vez definidas las gramáticas posibles, se crea el parser para poder aplicar las gramáticas a las recetas, gracias a la función *RegexpParser* que ofrece la librería *NLTK*. Se analiza cada receta o fila del *DataFrame*, y se aplica la gramática a la columna «token_POS», que contiene las etiquetas de los tokens. Esto, crea un árbol por cada receta, que muestra los diferentes fragmentos que contiene. Para guardar esta nueva información, se convierte el árbol a formato de lista con la llamada a *arbol_a_lista*.

Esta información, se guarda en el mismo *DataFrame* del parámetro p1, añadiendo una nueva columna llamada «chunks». Por lo tanto, la estructura de datos actualizada guarda el nombre de la receta en «Receta», la lista de los tokens en la columna «token», la lista de tuplas en «token_POS», y los fragmentos del texto en «chunks». Por ejemplo, la receta «albóndigas con champiñones» para los ingredientes representa como en la tabla 4.5.

- arbol_a_lista(p)

  Principalmente, convierte el árbol del parámetro p en una lista. Cada nodo del árbol pertenece a un fragmento del texto creado con la gramática aplicada anteriormente. Además, todos los fragmentos que hayan coincidido con esta gramática, tendrán una etiqueta o *label* llamado «LAB», para poder identificar los fragmentos.

Receta	token	token_POS	chunks
albóndigas con champiñones	[‘2’, ‘huevos’, ‘sal’, ‘pimienta’,‘15’, ‘gramos’, ‘de’, ‘migas’, ‘de’,‘pan’, ‘remojadas’, ‘en’, ‘leche’, . . .]	[(’2’, ’NUM’), (’huevos’, ’NOUN’), (’sal’, ’NOUN’), (’pimienta’, ’NOUN’), (’15’, ’NUM’), (’g’, ’UNI’), (’de’, ’ADP’), (’migas’, ’NOUN’), (’de’, ’ADP’), (’pan’, ’NOUN’), (’remojadas’, ’ADJ’),(’en’, ’ADP’), (’leche’, ’NOUN’), . . .]	[[’2’, ’huevos’], [’sal’], [’pimienta’], [’15’, ’g’, ’de’,’migas’, ’de’, ’pan’, ’remojadas’, ’en’, ’leche’], ...]

Tabla 4.5: *DataFrame* correspondiente a la fila «albóndigas con champiñones» con las columnas de nombre, token, etiquetas, y fragmento

Por lo tanto, gracias a las funciones que ofrece la biblioteca *NLTK* para la gestión de árboles (*nltk.Tree*), se analiza cada nodo y se obtiene el fragmento representado en una lista. Al final, se juntan todos los fragmentos conseguidos en una única lista que devuelve la función.

Siguiendo con el ejemplo, una parte del árbol de los ingredientes de la receta «albóndigas con champiñones» se representa en la figura 4.10.

Figura 4.10: Parte del árbol de la receta «albóndigas con champiñones»

Sin embargo, después de llamar a la función se crea la siguiente lista:

[[’2’, ’huevos’], [’sal’], [’pimienta’], [’15’, ’g’, ’de’, ’migas’, ’de’, ’pan’, ’remojadas’, ’en’, ’leche’], ...]

- preprocesar(p1,p2)

  Se encarga de preprocesar los tokens de cada receta, eliminando las palabras redundantes y obteniendo la raíz de cada palabra. La primera tarea, consiste en detectar las palabras de parada o *stopwords* que aparecen frecuentemente en el lenguaje, como por ejemplo, «de», «en», «y» etc. En el momento del aprendizaje, no aportan ningún valor al texto, por lo que su eliminación facilita el trabajo. La segunda tarea, también llamada *stemming*, consiste en reducir la palabra a una raíz, eliminando

generalmente el sufijo que contiene. De este modo, todas las palabras relacionadas se transforman en la misma, mejorando así el proceso de aprendizaje.

Por un lado, la biblioteca *NLTK* ofrece un corpus con *stopwords* en español, accesibles con la llamada al método *words*. Por otro lado, también facilita la parte de obtener la raíz de las palabras, ya que contiene el método *SnowballStemmer* para español.

El primer parámetro p1 define el último *DataFrame* conseguido, una vez hechas la parte de tokenización, etiquetado y fragmentación. Por lo tanto, el programa se encarga de leer los tokens de la última columna añadida (de fragmentos), y comienza con el preprocesamiento. Primero, comprueba si el token que se está analizando es un *stopword*. Si no es el caso, obtiene la raíz de la palabra y la añade a la nueva lista de fragmentos preprocesada.

Además, el programa también llama a la función *devolver_pos*, para obtener la etiqueta POS que se le ha atribuido a cada token. Si no se trata de un número, una unidad (de gramos, kilogramos, etc), un determinante o un adjetivo, se añadirá a una nueva lista. Con esto, se consigue crear otra nueva lista, pero que solo contenga las palabras clave para la hora del aprendizaje.

De este modo, existen dos listas nuevas, una con los fragmentos del texto preprocesados que se añade a la columna «pp», y la otra «limpia» con los fragmentos preprocesados que solo contenga palabras clave, en la columna «pp_clean». Siguiendo con el ejemplo de «albóndigas con champiñones», el *DataFrame* se representa tal y como se ve en la tabla 4.6.

Al final, se crea un archivo .csv de la estructura *DataFrame*, para acceder a la información fácilmente. El nombre dependerá del valor del parámetro p2: en el caso de tratarse de los ingredientes, el archivo se llamará Ingredientes_pp.csv. Por el contrario, si se trata de las elaboraciones, se llamará Elaboración_pp.csv (ver Anexo B).

- devolver_pos(p1,p2,p3)

  Devuelve la etiqueta asignada a la palabra del parámetro p3, que se encuentra en la fila p1 de la estructura de datos p2. Con la información dada por los parámetros, busca una receta concreta, y consulta la etiqueta asignada por el POS *tagger*.

train.py: Contiene los programas necesarios para crear un modelo de vectores de palabras con su correspondiente vocabulario. Entre estos programas están *obtener_corpus* que

Receta	token	token_POS	chunks	pp	pp_clean
albóndigas con champiñones	[‘2’, ‘huevos’, ‘sal’,‘pimienta’ ,‘15’,‘gramos’, ‘de’, ‘migas’, ‘de’,‘pan’, ‘remojadas’, ‘en’,‘leche’, . . .]	[(’2’, ’NUM’), (’huevos’, ’NOUN’), (’sal’, ’NOUN’), (’pimienta’, ’NOUN’),(’15’, ’NUM’), (’g’, ’UNI’), (’de’, ’ADP’), (’migas’, ’NOUN’),(’de’, ’ADP’), (’pan’, ’NOUN’), (’remojadas’, ’ADJ’), . . .]	[[’2’, ’huevos’], [’sal’], [’pimienta’], [’15’, ’g’, ’de’, ’migas’, ’de’, ’pan’, ’remojadas’, ’en’, ’leche’], ...]	[[’2’, ’huev’], [’sal’], [’pimient’], [’15’, ’g’, ’mig’,’pan’, ’remoj’, ’lech’] , . . .]	[[’huev’], [’sal’], [’pimient’], [’mig’, ’pan’, ’lech’] , . . .]

Tabla 4.6: *DataFrame* correspondiente a la fila «albóndigas con champiñones» con las columnas de nombre, token, etiquetas, fragmento, preprocesado, y preprocesado «limpio»

consigue el corpus con el que se entrenará el modelo, *num_aparicion* que obtendrá información sobre los ingredientes usados más comunes, y *entrenar*, que define los parámetros de la red neuronal que se entrenará con el corpus, creando así dicho modelo.

- obtener_corpus(p)

  Obtiene las palabras necesarias para crear el corpus con el que se entrenará la red neuronal. Estas palabras se adquieren de la estructura de datos *DataFrame* del parámetro p, tokenizada, etiquetada, fragmentada y preprocesada. En concreto, se obtienen los datos de la última columna «pp_clean», en la que solo están las raíces de los token que aportan información valiosa.

  Para ello, se accede a la estructura de datos, y se crea una lista de tokens por cada receta. Al mismo tiempo, todas las listas se juntan en una única, que guardará la información de todas las recetas. Además, también se crea otra lista en la que todos los ingredientes estén juntos, sin importar a que receta pertenezcan. Esto ayudará a obtener los ingredientes más comunes teniendo en cuenta la base de datos completa, gracias al método *num_aparicion*.

- num_aparicion(p)

  Dada la lista del parámetro p con todos los ingredientes que se encuentran en todas las recetas, obtiene tanto los ingredientes, como los pares de ingredientes más co-

munes. A estos pares de ingredientes también se les llama 2-grams, ya que analizan que dos palabras aparecen comúnmente juntos.

Gracias al método *counter* de la librería *collections*, se obtienen las palabras que más se repiten en toda la lista, junto a su número de apariciones. Además, la librería *NLTK* contiene la función *ngrams*, con la que se puede definir el número de palabras conjuntas que se quieren analizar. Por ejemplo, en este caso, interesan los pares de palabras, por lo que n será 2.

La funcionalidad de este programa es tan solo imprimir los resultados obtenidos, viendo así, todas las recetas como una unidad conjunta. Además, mediante el método *FeqDist*, se podrá crear un gráfico con la información obtenida.

- entrenar(p1,p2)

  A partir de la lista con el corpus del parámetro p1, define los parámetros de la red neuronal para la tarea de *word embedding*. La biblioteca *gensim* ofrece la herramienta *word2vec*, para convertir las palabras del corpus en un espacio de vectores de palabras.

  El modelo de *word2vec* empleado ha sido skip-gram. De este modo, con las sentencias dadas en corpus, se entrena la red neuronal calculando la probabilidad de cada palabra para que sea vecina con cada una de las otras palabras de vocabulario.

  La entrada de la red neuronal es un vector con la cantidad de posiciones del tamaño que tenga el vocabulario. En este vector, se representa una sola palabra, con el valor 1 en su posición correspondiente, y 0 en las demás posiciones. La salida trata de otro vector del mismo tamaño del vocabulario, que representará las probabilidades de cada una de las palabras para que sean vecinas de la palabra de entrada [Ruiz, 2018].

  Por lo tanto, el objetivo es que la salida de la red neuronal sea una distribución de probabilidades. Para ello, se emplea la función *Softmax*, que consigue en que todos los valores se encuentren entre 0 y 1 y que la suma de todos los valores sea 1. Una vez entrenada la red, los pesos de la capa oculta representarán los *word vectors* que se intentan aprender.

  Por lo tanto, gracia a esta función se puede entrenar una red neuronal de una capa con los parámetros deseados en una sola línea. El primer parámetro *sentence* guarda la lista de palabras del corpus, es decir, las sentencias que se quieran utilizar para en entrenamiento. En cambio, el parámetro *size* limitará el número de dimensiones en el espacio de vectores de palabras. Cuanto más alto sea su valor, más alta sera la

complejidad computacional del modelo de aprendizaje. En este caso, se ha elegido el valor de 100, ya que proporciona vectores adecuados para este caso particular.

El valor del parámetro *sg* definirá la arquitectura del modelo deseado, 1 para la arquitectura skip-gram y 0 (default) para la arquitectura CBOW. En general, este último es una mejor opción si se trabaja con un corpus grande, ya que es más eficiente computacionalmente. Sin embargo, como en este caso el corpus de recetas es relativamente pequeño, se ha elegido el modelo skip-gram, como se ha mencionado anteriormente.

Después, *window* definirá el número de palabras vecinas que se tendrán en cuenta para el entrenamiento. En este caso, se han elegido 10, por lo tanto, habrá un total de 20 palabras en el contexto de cada palabra (10 a la izquierda, 10 a la derecha). El parámetro *iter* define el número de iteraciones en las que se analizará cada palabra, que por defecto es 5. En un corpus grande, incluso un valor de 2 puede ser computacionalmente caro, pero como no se trata de este caso particular, se han definido 20 iteraciones.

Después, el parámetro *min_count* hace referencia al número mínimo de veces que tiene que aparecer una palabra, para poder crear un vector con él. Como se trata de un corpus reducido, se ha elegido darle el valor de 2, ya que por el contrario el vocabulario sería insignificante y no se podría obtener ninguna información valiosa con él. Por último, *workers* define el número de procesadores para dedicarse al entrenamiento, que en este caso, son 4.

Una vez elegidos los parámetros del modelo, con una sola llamada al método *word2vec*, se entrena el modelo con el corpus, y se crea el nuevo vocabulario en el que cada palabra será un vector en el espacio de palabras vectores. Después, para visualizar el vocabulario, se llama al método *dibujar*, que creará un gráfico con las posiciones de cada palabra. Para finalizar, se guarda el modelo en el mismo directorio, utilizando el nombre del parámetro p2 y se devuelve.

- dibujar(p1,p2)

  Dibuja los vectores de la lista p1. Estos vectores, tienen una dimensión de 100, tal y como se ha especificado en la configuración para la creación del modelo. Por lo tanto, para dibujar los vectores en el gráfico hay que reducir su tamaño a una dimensión de 2.

  Para ello, se emplea el módulo TSNE de *gensim*. Gracias a él, se construye una distribución de probabilidad sobre parejas de muestras en el espacio original. Por

lo tanto, las muestras más parecidas reciben alta probabilidad de ser escogidas, mientras que las muestras más diferentes reciben baja probabilidad [Burrueco,]. Este concepto de «parecido» se basa en la distancia entre diferentes puntos que construyen en vector. Después, se llevan los puntos del espacio de la alta dimensión al espacio de dos dimensiones de forma aleatoria, definiendo una distribución de probabilidad proporcionada.

Una vez reducida la dimensionalidad, se consigue un vector con dos coordenadas (x e y) para cada punto, y se dibujan en un gráfico. Utilizando las etiquetas del parámetro p2 que se quieran asignar, imprime los nombres correspondientes en cada punto del dibujo, pudiendo así, identificar cada palabra o receta.

consultations.py: Contiene los programas para hacer las consultas, es decir, extraer información una vez que se haya definido el vocabulario del modelo entrenado. Por un lado, el programa *elaboracion_similar* busca recetas con elaboraciones similares a la introducida, y *ingrediente_similar* busca recetas con ingredientes similares. Los métodos *buscar_similar*, *embeed_vector* y *cosine_dist* son funciones auxiliares para buscar las similitudes, ya que consiguen los vectores que representan las recetas, y obtienen las distancias que hay entre ellas. Por otro lado, el programa *recetas_posibles*, junto al método auxiliar *buscar_ingredientes*, obtiene las recetas que se puedan elaborar con los ingredientes introducidos. Por último, el programa *utilidad_ingrediente*, junto al método auxiliar *buscar_ebaloraciones*, consigue las elaboraciones que se puedan hacer con el ingrediente introducido.

- elaboracion_similar(p1,p2,p3,p4)

  Es el método principal que llama a la función *buscar_similar* para encontrar la receta más similar en cuanto a la elaboración a la introducida, y su valor de similitud. El parámetro p1 hace referencia a la estructura *DataFrame* original, creada al cargar las transcripciones. El parámetro p2 indica el nombre de la receta que se quiera analizar, para poder saber cual es la que más se parece a ella. Por último, el parámetro p3 contiene el modelo entrenado con el corpus, p4.

  Después de llamar a la función auxiliar, simplemente imprime los resultados que obtiene en la terminal.

- ingrediente_similar(p1,p2,p3,p4)

Como en el caso anterior, es el método principal que llama a la misma función *buscar_similar* para encontrar la receta con los ingredientes más similares a la introducida, y su valor de similitud. Los parámetros son los mismos que en la función *elaboracion_similar*: p1 hace referencia a la estructura *DataFrame* original, p2 indica el nombre de la receta que se quiera analizar, y el parámetro p3 contiene el modelo entrenado con el corpus, p4.

Después de llamar a la función auxiliar correspondiente, también imprime los resultados que obtiene en la terminal.

- buscar_similar(p1,p2,p3,p4)

  Busca la receta más similar a la p2 con el modelo p3 entrenado con el corpus p4. Primero llama al método *embeed_vector* para calcular los vectores que representen a todas las recetas. Una vez conseguida la lista de todos los vectores, obtiene la lista de distancias que tiene la receta buscada con las demás, llamando al método *cosine_dist*. Esta lista de distancias estará ordenada de forma descendente, y el valor máximo estará en la posición 0.

  Sin embargo, en esta posición se encontrará la misma receta p1, ya que la más parecida (o con la que se guarda menos distancia en el espacio) es ella misma. Por lo tanto, se obtienen la receta y su correspondiente similitud de la posición 1 de la lista, y se devuelven para que el método de la llamada principal imprima la información.

- embeed_vector(p1,p2)

  Se encarga de crear una lista que contenga todos los vectores de las recetas del parámetro p2. Para ello, se basa en los valores del vocabulario aprendido en el modelo de p1. Hasta ahora, se ha definido un vector para cada palabra que esté en el vocabulario del modelo. Sin embargo, para poder tener una visión mas completa de cada receta (y no solo de cada palabra) se puede calcular la media de todas las palabras que contenga la receta. Así, esta media representará a una receta con todas las palabras que la construyen.

  Por lo tanto, por cada receta que encuentre en el corpus, primero se comprueba que las palabras que la construyen existan en el vocabulario (ya que para que esté en él, cada palabra tiene que haber aparecido mínimo 2 veces, tal como se ha configurado al crear el modelo). Se crea una nueva lista con cada palabra de la receta que sí exista en el vocabulario, y se calcula la media de todos los vectores de palabras. Para ello, se utiliza la función *mean* de la librería *numpy*, que calcula la media de

todos los vectores introducidos. Por lo tanto, esta media simbolizará el vector de la receta completa.

Todos los vectores de recetas se guardan en una lista, que el programa devuelve.

- cosine_dist(p1,p2)

  Crea la lista con las similitudes de coseno para todos los vectores de recetas de p1, frente a la receta número p2. La similitud de coseno entre dos vectores en el espacio de vectores, es una medida que calcula el coseno del ángulo entre ellos. Esta, es una métrica de orientación, y no de magnitud, y puede ser vista como la comparación de dos documentos (o en este caso, recetas) en un espacio normalizado [Perone, 2013]. Por lo tanto, esta similitud puede decir que tan relacionados están estos dos documentos, centrado en el ángulo, y no en la magnitud. Teniendo en cuenta que el producto escalar de dos vectores en un espacio euclídeo se define como el producto de sus módulos por el coseno del ángulo que forman [Wikipedia, 2020]:

  $$A \cdot B = |A||B| \cos \theta$$

  La similitud se puede calcular de la siguiente manera:

  $$\cos \theta = \frac{A \cdot B}{|A||B|}$$

  Hay que tener en cuenta que puede haber dos vectores apuntando a puntos que estén muy lejos en cuanto a distancia, pero al mismo tiempo pueden tener un ángulo pequeño entre ellos. Por ejemplo, en un documento aparece la palabra «sal» 200 veces, y en otro 50 veces, la distancia Euclidiana puede ser grande, pero en ángulo entre ellos será pequeño porque apuntarán a la misma dirección, que es lo importante a la hora de comparar el texto.

  Gracias al método *cosine_similarity* que proporciona la librería *sklearn*, se calcula una matriz de similitud, dada la lista que contiene los vectores de las recetas. En esta matriz, se buscan todas las similitudes que tengan que ver con la receta número p2, y se guardan una lista ordenada de forma descendente, es decir, de mayor a menor. Al mismo tiempo, también se guarda el orden correspondiente de las recetas dependiendo de su similitud, para después obtener el nombre de la receta junto al valor.

  A continuación, se crea una estructura *heatmap* gracias a la librería *seaborn*, con los valores de la matriz original calculada. De este modo, se visualiza la similitud

que tiene cada receta respecto a las otras. Por último, se devuelve tanto la lista que guarda el orden de las recetas más similares, como la lista con el valor de las similitudes.

- recetas_posibles(p1,p2)

  Dada la lista de ingredientes del parámetro p1, obtiene las recetas que contengan todos esos ingredientes. La búsqueda se hace en la estructura *DataFrame* de p2. Para que una receta sea válida, se tendrán que utilizar todos los ingredientes de la lista, por lo que es posible que no se encuentre ninguna, si los ingredientes deseados no tienen nada que ver entre ellos.

  Primero, para poder comenzar con la búsqueda, se preprocesa la lista de ingredientes eliminando posibles *stopwords* y obteniendo la raíz de la palabra. Después, para la búsqueda en el *DataFrame*, se llama a la función auxiliar *buscar_ingredientes*, que devuelve la lista de todas las recetas posibles, y los ingredientes que coinciden con la búsqueda. Si la lista devuelta es vacía, no se habrá encontrado ninguna receta elaborable. En es caso contrario, se imprimirán los resultados en la terminal.

- buscar_ingredientes(p1,p2)

  Dada la lista de ingredientes preprocesada del parámetro p1, busca las recetas posibles con esos mismos ingredientes en la estructura de datos p2. Para ello, se analiza cada receta, y se obtienen los ingredientes que coincidan con la búsqueda. También se utiliza un contador, ya que para que una receta sea válida, tiene que contener todos los ingredientes buscados.

  La búsqueda se hace en la columna «pp_clean» del *DataFrame*, ya que la lista de ingredientes a buscar está también preprocesada. De este modo, se eliminan los sufijos de las palabras, y se tratan como a la misma. Por ejemplo, si se busca el ingrediente «albóndiga», también se tendrán en cuenta los ingredientes con la palabra «albóndigas».

  Por lo tanto, en el caso de contener todos los ingredientes, la receta se añadirá a la lista que después se devolverá para imprimir los resultados. Aun así, los ingredientes añadidos a la lista se obtendrán de la columna «chunks» y no de «pp_clean», ya que contiene las listas de ingredientes antes de preprocesarlas, con más información. Por ejemplo, si la búsqueda coincide con el ingrediente «migas», se devolverá el ingrediente «15 gramos de migas de pan remojadas en leche» para la receta «albóndigas con champiñones».

- utilidad_ingrediente(p1,p2)

  Dado el ingrediente del parámetro p1, consigue la parte de la elaboración de las recetas en las que se usa este ingrediente, en la estructura p2. Para ello, lo preprocesa para eliminar cualquier marca y obtener la raíz, y llama al método *buscar_elaboraciones*, consiguiendo así la lista de recetas con el paso en el que se menciona el ingrediente.

  Una vez obtenidos los resultados, imprime cómo se podría usar el ingrediente, junto al nombre de la receta a la que pertenece la elaboración. En el caso de que la lista de recetas posibles sea vacía, no se habrá encontrado el ingrediente en ninguna elaboración.

- buscar_elaboraciones(p1,p2)

  Busca en la columna «pp_clean» de la estructura de datos p2, la posible aparición del ingrediente p1. Si este ingrediente se usa en la elaboración, consigue el paso correspondiente, y obtiene toda información sobre él. Por lo tanto, aunque la búsqueda se haga en la columna preprocesada, se obtiene la parte en la que esté toda la información, en la columna «chunks».

  Las distintas partes de las elaboraciones se separan mediante verbos, ya que estos definen el principio de un nuevo paso en la receta. Por lo tanto, si la búsqueda coincide, se añadirá el paso completo en el que se use el ingrediente, y se guardará el nombre de la receta. Las coincidencias se añadirán a la lista, para que después, el método principal imprima toda la información.

5. CAPÍTULO

Uso y resultados del proyecto

5.1. Modo de uso

El primer paso, antes de poner en marcha el sistema de transcripción, será ejecutar el archivo dependencias.sh de la carpeta «transcripcion». Con ello, se instalarán las bibliotecas necesarias para este apartado del proyecto. Después, será necesario exportar la ubicación de las credenciales de Google Cloud, mediante el comando *export* y la variable *GOOGLE_APPLICATION_CREDENTIALS*.

En este momento, los pasos para hacer la transcripción del audio estáran listos, por lo que se podrá ejecutar el programa principal del fichero main_transcripcion.py. Al final de la ejecución, tal y como se muestra en la figura 5.1, se imprimirá el mensaje «Se ha completado la transcripción de todos los archivos», si todo se realiza correctamente.

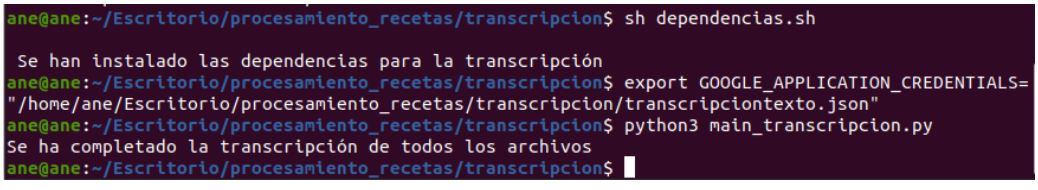


Figura 5.1: Ejecución de la fase de transcripción

Aquí termina la ejecución de la parte de la transcripción. Se crearán dos nuevas carpetas, tal y como muestra la figura 5.2: «base_datos», donde se descargarán todos los archivos

de audio de la base de datos, y «transcripciones», donde se guardará la transcripción de cada receta en un archivo .txt independiente.

Figura 5.2: Ficheros creados en la fase de transcripción

Después, para la parte del procesamiento del lenguaje natural, también hay que ejecutar el fichero dependencias.sh de la carpeta «nlp». De este modo, se instalarán los módulos necesarios para la parte del PLN. A continuación, se podrá ejecutar el método principal del fichero main_nlp.py como se muestra en la figura 5.3, el cual llama a todas las funciones necesarias para procesar las recetas.

```
ane@ane-VirtualBox:~/Escritorio/procesamiento_recetas/nlp$ sh dependencias.sh

Se han instalado las dependencias para el pln

ane@ane-VirtualBox:~/Escritorio/procesamiento_recetas/nlp$ python3 main_nlp.py
Cargando ficheros...
Procesando los ingredientes...
Procesando las elaboraciones...
```

Figura 5.3: Ejecución de la fase del procesamiento del lenguaje natural

Una vez procesadas las recetas, tanto la parte de los ingredientes como la de las elaboraciones, se habrán creado varios ficheros en el mismo directorio, tal y como muestra la figura 5.4. Por un lado, el archivo recetas.csv guardará las transcripciones tal y como están, separadas en tres columnas diferentes: el nombre de la receta, los ingredientes, y la elaboración. Por otro lado, los archivos Ingredientes_pp.csv y Elaboración_pp.csv guardarán la información preprocesada de los ingredientes y la elaboración respectivamente. Cada archivo .csv preprocesado consta de seis columnas, en la que cada fila representará una receta distinta: el nombre, los token, los token con su correspondiente etiqueta, la división en fragmentos, la división en fragmentos preprocesada, y la división de fragmentos preprocesada y limpia. Todos los archivos .csv se encuentran en el Anexo B.

Además, también se habrán creado dos modelos que posicionan los vectores de palabras en el espacio de vectores, uno con el corpus de los ingredientes, y otro con el corpus de elaboraciones. Ambos modelos, tienen el formato .w2v, y también se encuentran en el mismo directorio de la carpeta «nlp».

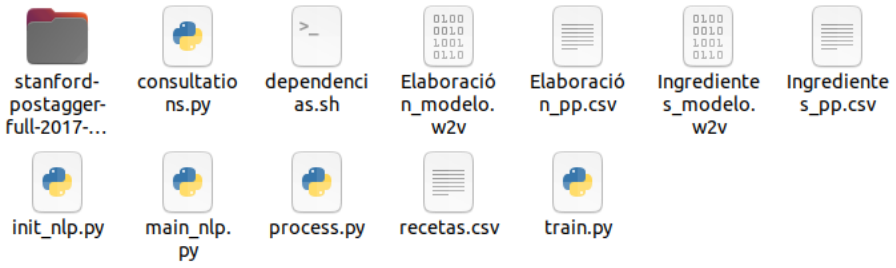


Figura 5.4: Ficheros creados en la fase del procesamiento del lenguaje natural

Con la ejecución del programa principal, además de crear los nuevos archivos, se inicia el menú de la figura 5.5, con el cual se podrá extraer cierta información de las recetas. Este menú ofrece varios tipos de consulta, como obtener recetas similares según la elaboración e ingredientes, obtener las recetas que se puedan elaborar con ciertos ingredientes, ver la utilidad de un ingrediente o ver los nombres de todas las recetas que construyen la base de datos.

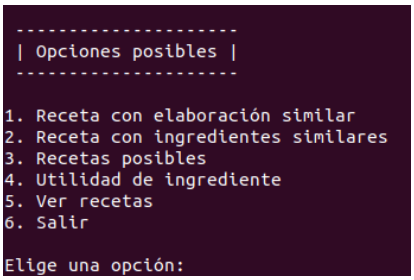


Figura 5.5: Menú de la ejecución del procesamiento del lenguaje natural

1. Receta con elaboración similar: al elegir la primera opción, el sistema pedirá el número de la receta en la que se quiera basar la búsqueda. Una vez hecho esto, el programa imprimirá el nombre de la receta introducida (la que coincida con el número) y el nombre de la receta más similar en cuanto a la elaboración. Por último, devolverá la similitud de la elaboración entre las dos recetas (calculada mediante la distancia de coseno). Se puede ver una muestra de la ejecución en la figura 5.6.

2. Receta con ingredientes similares: con la segunda opción, el sistema también pedirá el número de la receta en la que se quiera basar la búsqueda. En este caso, el programa imprimirá el nombre de la receta introducida (la que coincida con el número) y el nombre de la receta más similar en cuanto a los ingredientes. También

```
Elige una opción: 1
--------------------

Introduce el número de la receta: 4

Receta introducida:  canelones de espinacas
Receta con elaboración similar:  canelones rellenos de verdura
Similitud:  0.9999804
```

Figura 5.6: Primera opción del menú: búsqueda de receta con elaboración similar

se devuelve la similitud entre los ingredientes de las dos recetas, calculada por la distancia de coseno. Se puede ver una muestra de la ejecución en la figura 5.7.

```
Elige una opción: 2
--------------------

Introduce el número de la receta: 4

Receta introducida:  canelones de espinacas
Receta con ingredientes similares:  canelones rellenos de paté
Similitud:  0.99928945
```

Figura 5.7: Segunda opción del menú: búsqueda de receta con ingredientes similares

3. Recetas posibles: con la tercera consulta, el programa pedirá el ingrediente o los ingredientes que se quieran buscar. Basándose en la columna de ingredientes utilizados en las recetas, se devolverá en nombre de la receta en la que se hayan encontrado todos los ingredientes indicados. Además, también devolverá la información complementaria en torno a ese ingrediente, en el caso de que la haya, como se muestra en la ejecución de la figura 5.8.

```
Elige una opción: 3
--------------------

Introduce un ingrediente, o varios separados por comas: tomate,patatas


La receta  crema de calabaza  contiene los ingredientes:
->  un vaso de tomate natural
->  dos patatas nuevas
La receta  lentejas con verduras  contiene los ingredientes:
->  dos tomates
->  patatas
La receta  pizza napolitana  contiene los ingredientes:
->  4 tomates maduros
->  salsa de tomate
La receta  tortilla especial de patata  contiene los ingredientes:
->  4 patatas
->  salsa de tomate
```

Figura 5.8: Tercera opción del menú: búsqueda de recetas posibles con ciertos ingredientes

4. Utilidad de ingrediente: mediante la cuarta consulta, el programa pide un solo ingrediente, para que se pueda buscar en todas la elaboraciones de las recetas, y obtener los pasos en los que se emplea dicho ingrediente. El sistema imprimirá el nombre de la receta en la que se use el ingrediente, junto a la parte de la elaboracion en la que se emplea, como se muestra en la figura 5.9.

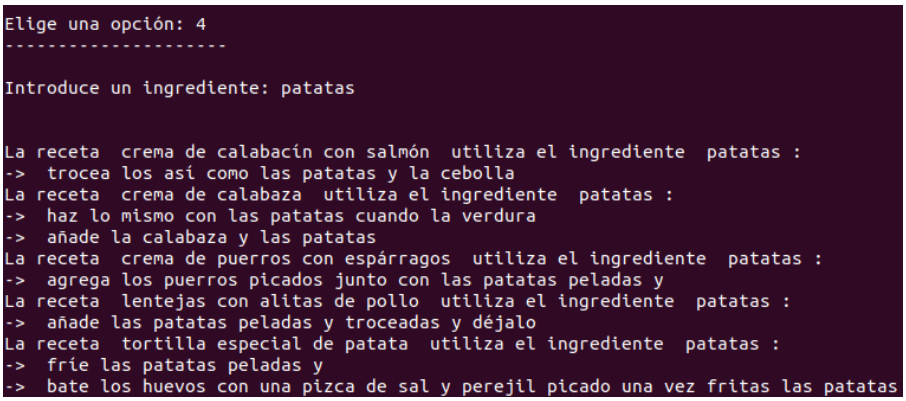


Figura 5.9: Cuarta opción del menú: búsqueda de utilidad de un ingrediente

5. Ver recetas: a través de la quinta consulta, se podrá ver la lista de todas las recetas de la base de datos. Esta función puede ayudar para recordar al usuario cuales son los nombres de estas recetas, y su correspondiente numeración, ya que puede útil para la primera y segunda consulta. Se puede ver una muestra de la ejecución en la figura 5.10, ya que imprime las primeras 16 recetas.

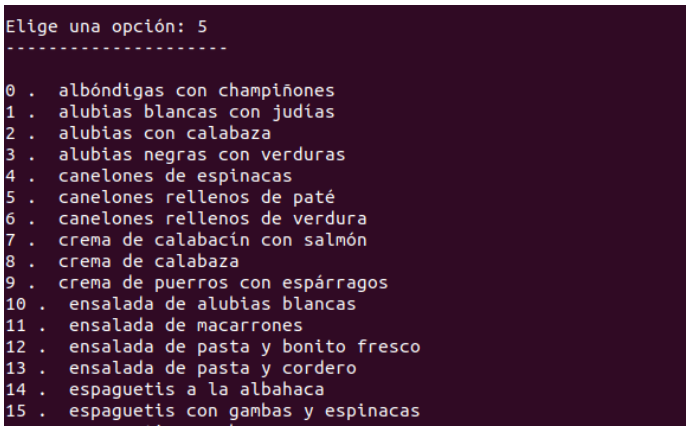


Figura 5.10: Quinta opción del menú: ver los nombres y número de las recetas

6. Salir: la última opción no se trata de una consulta, sino del modo para terminar con la ejecución del menú, si no se quieren obtener más datos. Una vez que el usuario

salga de la ejecución del menú, no se podrá acceder a él directamente, ya que habrá que ejecutar todo el procesamiento del lenguaje otra vez.

5.2. Resultados obtenidos

Una vez procesadas las recetas y preparadas para extraer información de ellas, se han obtenido varios resultados. En cuanto a la base de datos creada, se han conseguido tanto las palabras como los pares de palabras más frecuentes, para obtener una visión general de la composición de las recetas. Por un lado, para la parte de los ingredientes, las treinta palabras más usadas han sido las que se muestran en la figura 5.11.

```
Palabras más comunes:  [('sal', 37), ('aceit', 36), ('agu', 27), ('tomat', 22), ('pimient', 17), (
'cebollet', 16), ('ajo', 14), ('ques', 14), ('sals', 13), ('ceboll', 11), ('dient', 11), ('perejil
', 10), ('huev', 9), ('zanahori', 9), ('puerr', 8), ('litr', 8), ('vas', 7), ('bechamel', 7), ('cu
char', 7), ('cuart', 6), ('oliv', 6), ('patat', 6), ('pan', 4), ('harin', 4), ('carn', 4), ('lonch
', 4), ('alubi', 4), ('espinac', 4), ('atun', 4), ('mantequill', 4), ('pic', 4), ('macarron', 4),
('hoj', 4), ('lech', 3), ('jamon', 3), ('calabaz', 3), ('canelon', 3), ('piñon', 3), ('calabacin',
 3), ('vinagr', 3), ('aceitun', 3), ('gamb', 3), ('espaguetis', 3), ('garbanz', 3), ('verdur', 3),
 ('lasañ', 3), ('nat', 3), ('lentej', 3), ('pizz', 3), ('cerd', 2)]
```

Figura 5.11: Palabras más comunes en los ingredientes

Como era de esperar, palabras como «sal», «aceite» y «agua» son los ingredientes más básicos, por lo que aparecen en casi todas las recetas. Los demás ingredientes dependen de las recetas elegidas para la base de datos. Por ejemplo, como se puede ver, muchas de las recetas añadidas están elaboradas con tomate, pimienta, cebolletas, ajo, etc. La figura 5.12 muestra un gráfico con estas palabras, para obtener una visión más general.

En cambio, los diez pares de palabras más comunes han sido los de la figura 5.13. En este caso también, se puede ver el uso de los ingredientes más comunes «aceite», «agua» y «sal», mediante las combinaciones más usadas entre ellas. Además, se han creado pares de palabras con varios de los ingredientes más usados, como salsa de tomate o diente de ajo.

En cuanto a la parte de la elaboración, las palabras más comunes han sido las de la figura 5.14.

Como se puede ver, en las elaboraciones las palabras más usadas son una mezcla de ingredientes (como el aceite, la sal, el agua, el tomate, etc.) y acciones. Las diferentes acciones o pasos en la elaboración se definen a través de verbos, por lo que palabras como «añad» del verbo añadir, «sirv» del verbo servir «coc» de cocinar o «cuec» de cocer, son las más comunes. En la figura 5.15 se ve un gráfico con la frecuencia de estas palabras.

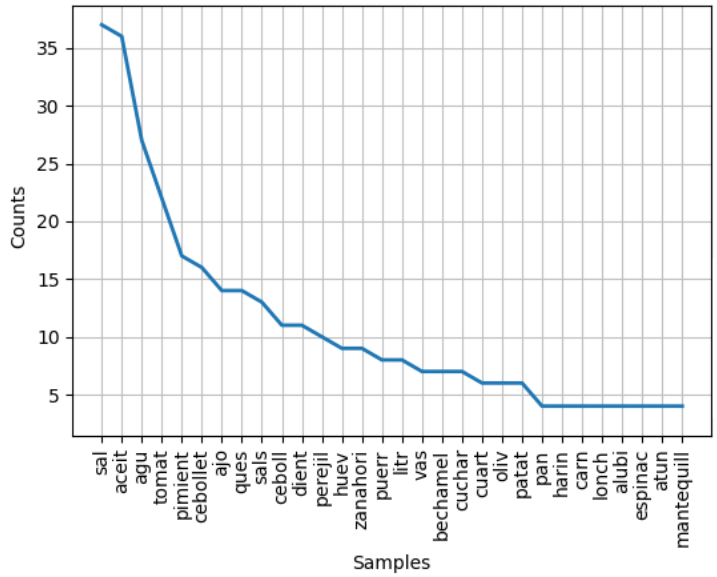


Figura 5.12: Gráfico de palabras más comunes en los ingredientes

```
2-gram más comunes:  [(('sals', 'tomat'), 11), (('dient', 'ajo'), 11), (('sal', 'agu'), 11), (('ac
eit', 'oliv'), 6), (('aceit', 'sal'), 5), (('agu', 'aceit'), 5), (('agu', 'sal'), 5), (('sal', 'pi
mient'), 4), (('aceit', 'agu'), 4), (('tomat', 'cebollet'), 4)]
```

Figura 5.13: Pares de palabras más comunes en los ingredientes

```
Palabras más comunes:  [('aceit', 43), ('minut', 41), ('sal', 38), ('añad', 38), ('agu', 38), ('si
rv', 22), ('tomat', 21), ('coc', 21), ('cuec', 19), ('bien', 17), ('past', 17), ('pon', 16), ('sar
ten', 15), ('mezcl', 14), ('ques', 14), ('pimient', 13), ('sals', 13), ('agreg', 13), ('dej', 13),
 ('cebollet', 13), ('coloc', 13), ('fuent', 13), ('cazuel', 12), ('fueg', 12), ('verdur', 12), ('d
espues', 11), ('si', 11), ('ajo', 11), ('bechamel', 11), ('horn', 11), ('pic', 10), ('aproxim', 10
), ('alubi', 9), ('encim', 9), ('perejil', 9), ('huev', 8), ('qued', 8), ('escurr', 8), ('rehog',
8), ('patat', 8), ('cort', 8), ('garbanz', 8), ('carn', 7), ('ceboll', 7), ('troc', 7), ('punt', 7
), ('vez', 7), ('zanahori', 6), ('chorr', 6), ('puerr', 6)]
```

Figura 5.14: Palabras más comunes en las elaboraciones

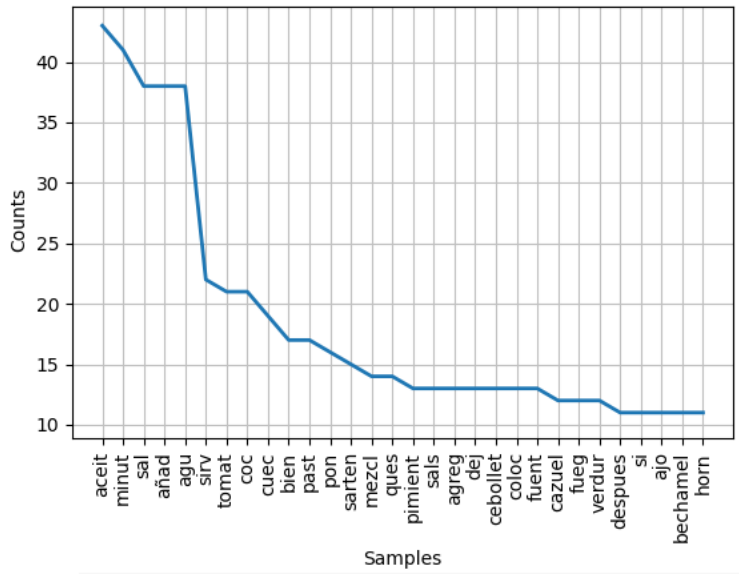


Figura 5.15: Gráfico con palabras más comunes en las elaboraciones

Sin embargo, los pares de palabras mas usadas son las que aparecen en la figura 5.16. En este caso también se ve el uso de ingredientes más comunes, junto a las acciones típicas con las que se preparan ciertos alimentos, como cocer la pasta, poner a punto de sal, echar un chorro de aceite, etc.

```
2-gram más comunes:  [(('agu', 'sal'), 18), (('sarten', 'aceit'), 11), (('sals', 'tomat'), 9), (('
minut', 'aproxim'), 8), (('cuec', 'past'), 7), (('minut', 'sirv'), 7), (('punt', 'sal'), 6), (('ch
orr', 'aceit'), 6), (('past', 'agu'), 6), (('sirv', 'cuec'), 6)]
```

Figura 5.16: Pares de palabras más comunes en las elaboraciones

Con esta información, se obtiene una visión general para la preparación de las recetas. Para poder extraer otro tipo de información mediante las consultas, se han creado dos corpus, uno para los ingredientes, y otro para las elaboraciones. Con ellos, se ha entrenado un modelo de vectores de palabras, obteniendo así un vocabulario, en el que todas las palabras aprendidas se representan en un espacio de vectores. Aun así, las dimensiones de estos vectores de palabras son en este caso, de 100, por lo que se ha tenido que reducir esta dimensión a 2, pudiendo así crear una representación gráfica.

El vocabulario para la sección de los ingredientes está compuesto por 66 elementos de 490 palabras que contenía el corpus. Después de crear la representación de este vocabulario en 2 dimensiones, se ha conseguido el gráfico de la figura 5.17, en el que se muestra la posición de las palabras.

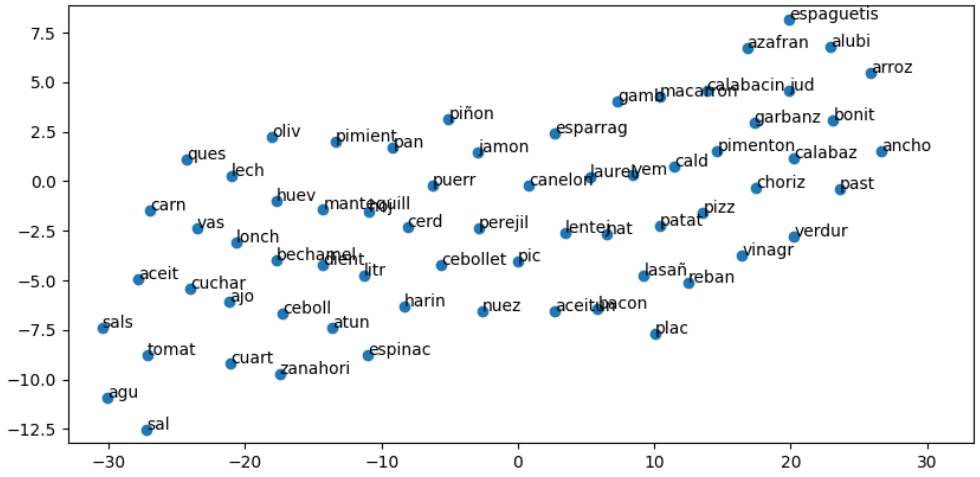


Figura 5.17: Vocabulario del modelo creado con el corpus de los ingredientes

En cambio, el vocabulario para la sección de las elaboraciones es más amplio, ya que está compuesto por 186 elementos de 1282 palabras que contenía su corpus. También se ha creado la representación del vocabulario en 2 dimensiones, consiguiendo el gráfico que se muestra en la figura 5.18.

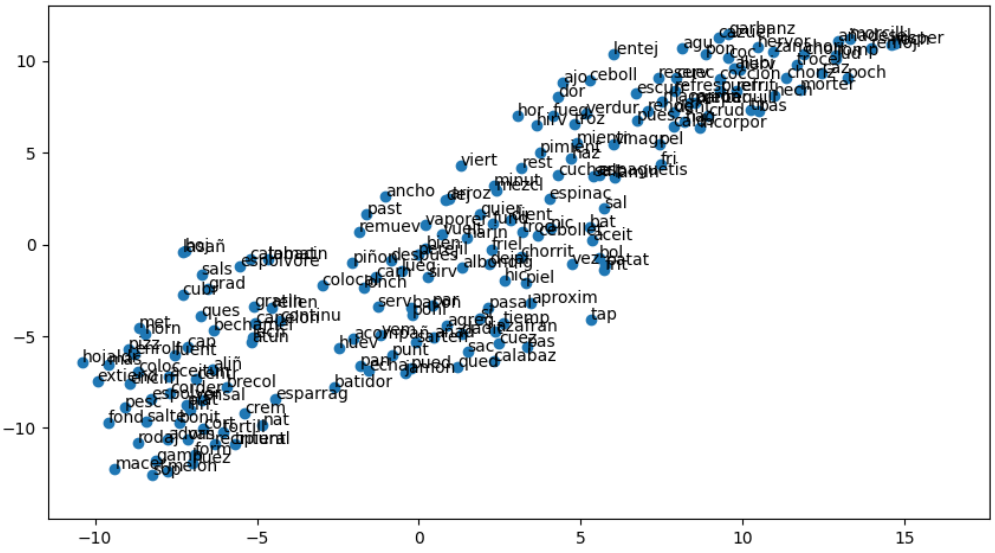


Figura 5.18: Vocabulario del modelo creado con el corpus de las elaboraciones

Gracias a estas representaciones, se puede observar como se sitúan diferentes palabras en el espacio, de acuerdo a su contexto. Aun así, estas palabras representan el vocabulario de todas las recetas conjuntas, como si fuera una sola unidad. Para poder representar cada

receta en el espacio, se ha obtenido la media entre todas las palabras que construyan la receta. Por lo tanto, la media de ciertos vectores de palabras representarán a una receta concreta, también representada por un vector.

Para la sección de los ingredientes, las recetas se sitúan de la manera que se muestra en la figura 5.19, teniendo en cuenta que cada valor representa el número de la receta de la base de datos.

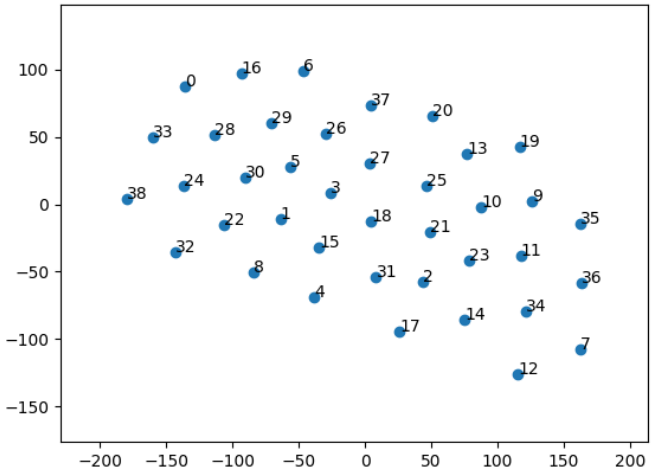


Figura 5.19: Recetas según el modelo creado con el corpus de los ingredientes

En cambio, en las elaboraciones, las diferentes recetas se sitúan como se puede ver en la figura 5.20.

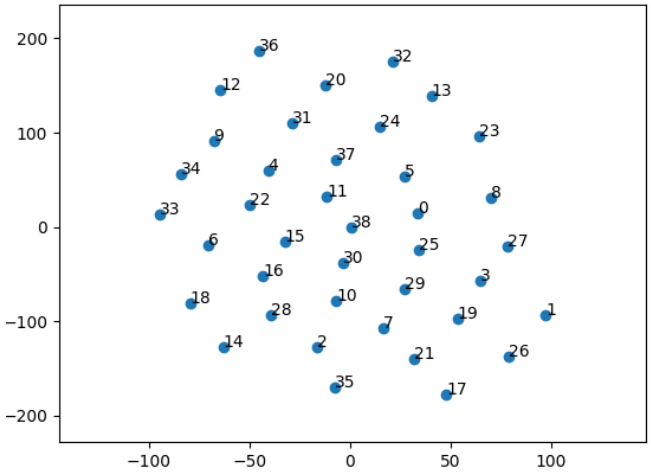


Figura 5.20: Recetas según el modelo creado con el corpus de las elaboraciones

Para calcular la similitud que hay entre las diferentes recetas, se ha usado la distancia de coseno, que define la similitud entre dos documentos o textos. Obteniendo estos valores de similitud en una matriz, se han representado en dos *heatmaps* diferentes, primero la sección de ingredientes (en la figura 5.21) y después la elaboración (en la figura 5.22). Hay que tener en cuenta que la mayor similitud de una receta, será con ella misma, por lo tanto, los valores más altos estarán en la diagonal principal. Además, se trata de una matriz simétrica, ya que la similitud entre una receta A y B, es la misma entre B y A.

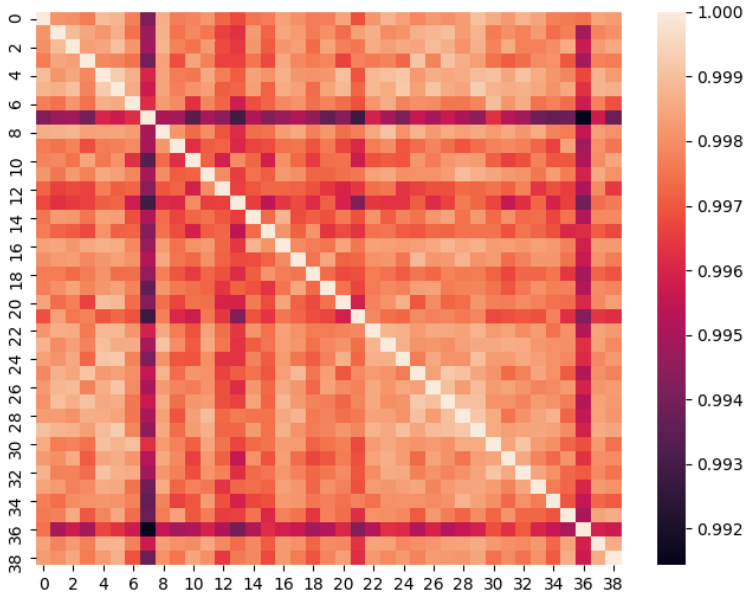


Figura 5.21: *Heatmap* de las similitudes entre ingredientes de recetas

Como se puede observar, todas las similitudes entre recetas se mueven entre valores muy altos. Esto se puede deber a que muchas de las recetas que aparecen en la base de datos son parecidas. Por ejemplo, existen tres recetas de alubias, otras tres de canelones, tres de ensaladas, tres de espaguetis, etc. Es decir hay varias recetas de la misma categoría, con elaboraciones e ingredientes similares. Además, el único tema que se trata es la cocina, por lo que al fin y al cabo, no hay diferencias demasiado grandes en el contexto de las palabras. Por lo tanto, se puede decir que en el proyecto se trata un caso muy concreto de las recetas de cocina, en las que las similitudes son muy grandes.

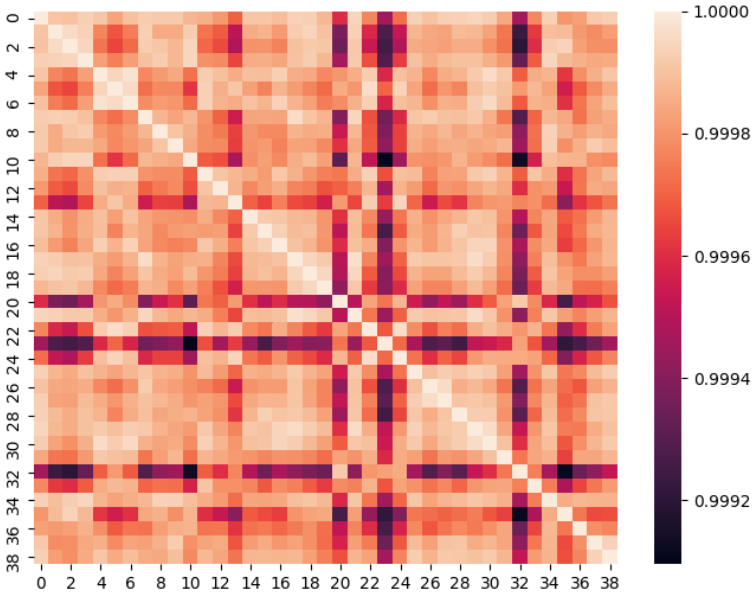


Figura 5.22: *Heatmap* de las similitudes entre elaboraciones de recetas

6. CAPÍTULO

Conclusiones

Después de realizar el proyecto, se han sacado varias conclusiones, tanto de las herramientas utilizadas, como conclusiones generales del desarrollo del proyecto. Además, se han analizado varias mejoras o cambios posibles para el futuro.

6.1. Conclusiones de las herramientas utilizadas

Por un lado, la principal herramienta utilizada para la fase de la transcripción ha sido Google Cloud. Gracias a Cloud Storage, se ha almacenado la base de datos fácilmente, y el acceso a ella ha sido sencillo a través de sus librerías. Con la API de Speech-to-Text, la transcripción a texto también ha sido sencilla. Además, no ha habido ningún problema en cuanto al idioma, ya que la plataforma proporciona servicios en español en la mayoría de casos.

Por lo tanto, se puede decir que es una plataforma fácil de utilizar, y que proporciona mucha información a través de la documentación y tutoriales rápidos que ofrece en su página. Aun así, los servicios tienen sus limitaciones, tanto por la cantidad de espacio de almacenamiento, como para el tiempo de trascripción de audio. En este caso, no se ha tenido ningún problema, ya que se trata de una base de datos relativamente pequeña, y una duración de audio dentro de los límites de Cloud. De todos modos, es un aspecto a tener en cuenta para proyectos más grandes en cuanto a tamaño de los datos.

Por otro lado, para la parte del procesamiento del lenguaje, hay que decir que existen

muchas librerías en Python que facilitan el trabajo. La principal librería empleada, *NLTK*, ofrece una infinidad de funciones para todo tipo de tareas del procesamiento del lenguaje, empezando por la tokenización, hasta el etiquetado, fragmentación, preprocesado, extracción de entidades reconocidas, etc. Además, a parte de proporcionar sus propias funciones, admite la combinación con otros software como el de Stanford (que se ha usado para la parte de *POS tagging*).

Igualmente, *NLTK* ofrece documentación extensa sobre todas las diferentes tareas que se pueden realizar en cuanto a este procesamiento. Sin embargo, la principal limitación es que la mayoría de funcionalidades principales solo están disponibles para inglés, por lo que hay que buscar extensiones de la librería dependiendo del idioma con el que se quiera trabajar. Estas extensiones son completamente compatibles, pero en algunos casos no se obtienen tan buenos resultados como con las funcionalidades principales en inglés.

Una vez procesados los datos con *NLTK*, también se han utilizado las librerías *gensim* y *scikit-learn*, que sirven para el modelado semántico del texto y el aprendizaje automático. Ambas son muy utilizadas en el campo del procesamiento del lenguaje, ya que ofrecen funciones que pueden ser de gran ayuda. Por ejemplo, gracias a una única función de *gensim*, *word2vec*, se ha podido crear un modelo para representar todo el vocabulario en el espacio de vectores, entrenando el corpus a través de una red neuronal.

Por lo tanto, se puede decir que existen muchas herramientas útiles para Python, que facilitan el trabajo y ayudan tanto en la transcripción, como en el procesamiento del lenguaje. Además se ofrecen muchas posibilidades diferentes, que aunque no se hayan utilizado en este proyecto concreto, pueden ser adecuadas para otro tipo de tareas en el área del lenguaje natural.

6.2. Conclusiones del proyecto

Como conclusión general del proyecto, se puede decir que se trata de un caso muy concreto en cuanto a recetas de cocina. La base de datos contiene 39 recetas cortas, de 40 a 95 segundos, y se pueden encontrar varias recetas de la misma categoría con diferentes ingredientes o elaboraciones. Por lo tanto, no es una base de datos muy diversa, ni con grandes cantidades de información.

Las funcionalidades elegidas para la extracción del conocimiento de las recetas, se han creado con la finalidad de ayudar a obtener información básica sobre ellas. Encontrar recetas con ingredientes o elaboraciones similares, obtener las elaboraciones posibles con

ciertos ingredientes concretos y conocer cual es el uso de un ingrediente pueden ser consultas útiles para poder conocer más recetas. Aun así, se podrían haber añadido funcionalidades más complejas, que habrían requerido más aprendizaje y desarrollo.

Se ha cumplido el objetivo principal del proyecto, que era obtener conocimiento de procesos de fabricación a partir de sus descripciones en lenguaje natural. Además, se han estudiado diferentes funcionalidades de la plataforma Google Cloud, y se ha aprendido a poner en marcha un proyecto usando sus servicios. También se ha investigado sobre el procesamiento del lenguaje natural, de las posibilidades que ofrece, y de las diferentes aplicaciones que se pueden crear con su uso.

En cuanto a los objetivos personales, se ha conseguido crear un proyecto sin conocimiento previo de las herramientas planteadas a utilizar, y resolver las dudas y dificultades de forma autónoma e independiente. Además, se ha aprendido a identificar diferentes tareas del proyecto, y a gestionar tanto el desarrollo, como la documentación de este. Por último se han puesto en práctica diferentes competencias adquiridas durante el grado, aplicando el conocimiento y las metodologías aprendidas en diferentes cursos.

6.3. Posibles mejoras

En cuanto a las posibles mejoras del proyecto, por un lado, se podría ampliar la base de datos, incluyendo recetas diferentes y más diversas. También se podrían añadir diferentes características para cada receta, como el origen, el tiempo de elaboración, las valoraciones de cocineros, etc. De este modo, la información extraída del texto sería más enriquecedora y completa. Por otro lado, se podrían añadir más tipos de consultas, con funcionalidades más complejas, y extraer otro tipo de información del texto. Por ejemplo, se podría añadir una función de recomendaciones de recetas según diferentes valoraciones, predecir el lugar de origen de la receta según los ingredientes empleados, etc.

Del mismo modo, se podría cambiar la forma de extraer la información, creando un sistema de diálogo con el que el usuario pudiera interactuar. Así, se podrían hacer las preguntas con la información que se quisiera adquirir, y el sistema se encargaría de entender la pregunta, buscar la respuesta, y devolvérsela al usuario. También se podrían usar más servicios de Google Cloud, como por ejemplo, Text-to-Speech. De este modo, en sistema podría obtener los datos de las consultas del usuario, y dar una respuesta por voz.

Por lo tanto, este proyecto trata tan solo de un acercamiento hacia el procesamiento del

lenguaje natural y la extracción de conocimiento, por lo que se podría alargar mucho más añadiendo nuevas funcionalidades.

Anexos

A. ANEXO

Código del proyecto

El código del proyecto está disponible a través del siguiente enlace: https://github.com/anearburua/procesamiento_recetas.git

Existen tres carpetas principales en el proyecto procesamiento_recetas: **transcripcion**, **nlp** y **otros**.

- **transcripcion**: contiene los ficheros necesarios para la fase de la transcripción.
- **nlp**: contiene los ficheros necesarios para la fase del procesamiento del lenguaje natural.
- **otros**: contiene las transcripciones de las recetas de la base de datos. Si el usuario no tiene una cuenta en Google Cloud, no podrá hacer la transcripción a través de sus servicios. Por lo tanto, si se quieren probar las funcionalidades de la parte PLN sin este primer paso, podrá hacerlo con los archivos de las transcripciones incluidas, añadiéndolas a la carpeta **transcripcion**.

Por seguridad, la carpeta **transcripcion** no contiene el archivo JSON con el que se accede a la cuenta de servicio de Google Cloud. Por lo tanto, para poder ejecutar esta fase será necesario crear una cuenta nueva.

Además, la carpeta **nlp** no contiene los modelos POS de Stanford Natural Language, ya que su tamaño es demasiado grande. Para conseguirlo, se tendrá que acceder la web https://nlp.stanford.edu/software/tagger.shtml, y descargar la versión 3.8.0.

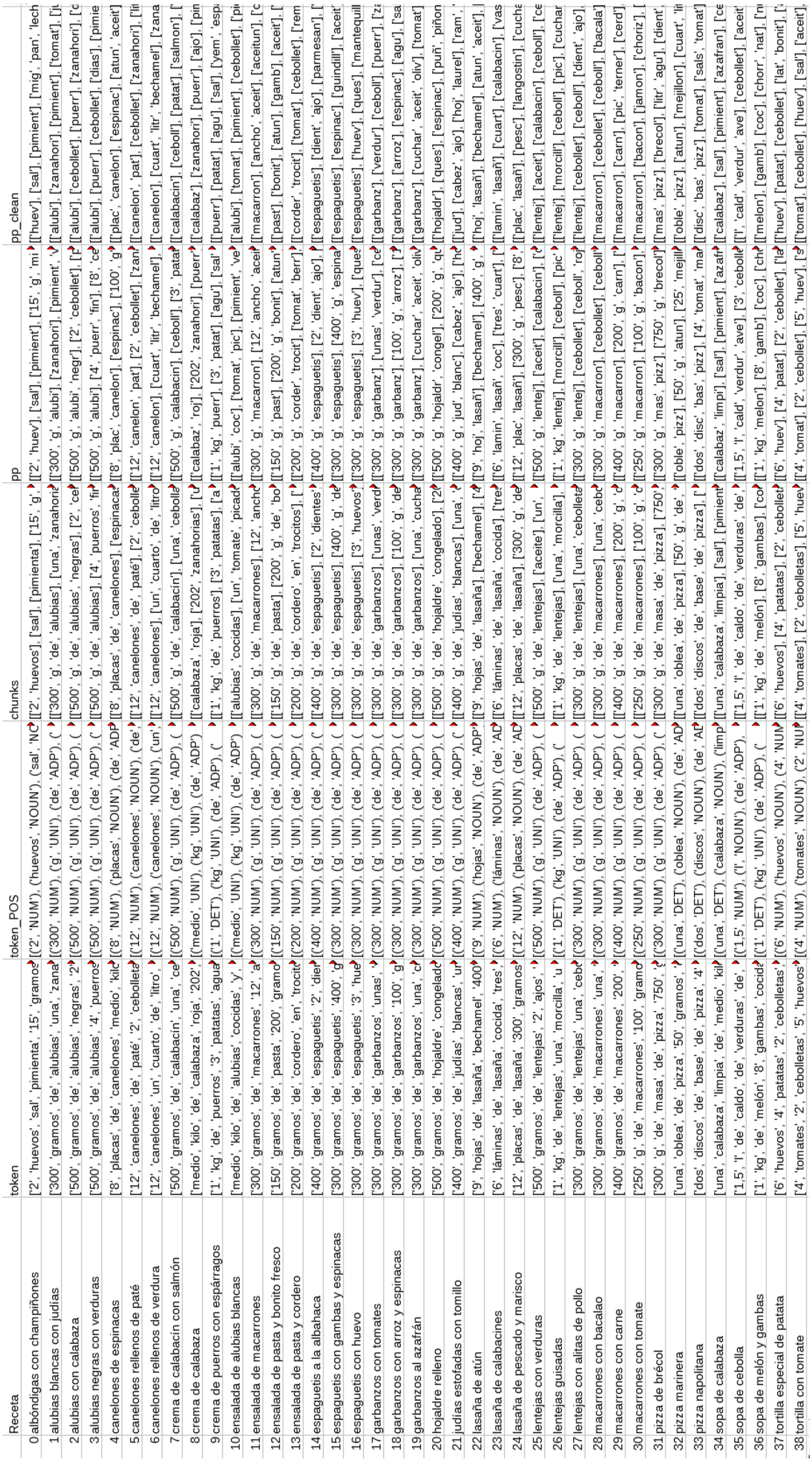
B. ANEXO

Archivos CSV

Con las ejecuciones del programa se han conseguido varios archivos de formato .csv para guardar la información obtenida. Entre ellos se encuentran las tablas de **recetas.csv** (figura B.1), **Ingredientes_pp.csv** (figura B.2) y **Elaboración_pp.csv** (figura B.3).

	Receta	Ingredientes	Elaboración
0	albóndigas con champiñones	2 huevos sal pimienta 15 gramos de migas de pan remojadas en leche 500 gramos de char	mezcla bien la carne con los huevos la sal la pimienta y la miga de pan forma las albóndigas
1	alubias blancas con judías	300 gramos de alubias una zanahoria un pimiento verde un tomate gramos de judías verdes	las alubias estarán en remojo desde la víspera comienza poniendo a cocer las alubias con la
2	alubias con calabaza	500 gramos de alubias negras 2 cebolletas un puerro una zanahoria 3 dientes de ajo 400 gr	pon las alubias en una cazuela con abundante agua fría a cocer junto con las verduras picad
3	alubias negras con verduras	500 gramos de alubias 4 puerros finos 8 cebolletas finas 2 dias un pimiento un pimiento verd	pon a cocer las alubias con abundante agua y sal cuando rompa a hervir añade toda la verdu
4	canelones de espinacas	8 placas de canelones medio kilo de espinacas 100 gramos de atún en aceite salsa de toma	cuece la pasta en agua con sal y un chorrito de aceite escurre y reservapara preparar la bech
5	canelones rellenos de paté	12 canelones de paté 2 cebolletas zanahorias un cuarto litro de bechamel y queso rallado 4	cuece la pasta en agua con sal y un chorro de aceite una vez cocida escurre la y reserva para
6	canelones rellenos de verdura	12 canelones un cuarto de litro de bechamel 2 zanahorias 2 puerros 50 gramos de un cuarto	quién quiere cuecen las zanahorias las espinacas y la coliflor por separado y pican las cuece
7	crema de calabacín con salmón	500 gramos de calabacín una cebolla 3 patatas 200 gramos de salmón ahumado litro de bel	pela los calabacines y trocea los así como las patatas y la cebolla pon todo en una cazuela co
8	crema de calabaza	medio kilo de calabaza roja 202 zanahorias un puerro un de ajo medio pimiento verde un va	pica la verdura y ponla a rehogar en una cazuela con el vaso de aceite a fuego lento mientras
9	crema de puerros con espárragos	1 kg de puerros 3 patatas agua sal 24 yemas de espárragos aceite de oliva 12 rebanadas d	limpia bien los puerros con agua a hervir y agrega los puerros picados junto con las patatas p
10	ensalada de alubias blancas	medio kilo de alubias cocidas y escurridas un tomate picado un pimiento verde picado una c	cuando estén cocidas las alubias sin que se pasen escurre y pasa por agua fría junto a la ver
11	ensalada de macarrones	300 gramos de macarrones 12 anchoas en aceite 12 aceitunas rellenas últimamente una ce	cuece los macarrones al dente en agua hirviendo con sal después escurre y refresca los en u
12	ensalada de pasta y bonito fresco	150 gramos de pasta 200 gramos de bonito o atún fresco 20 gambas peladas aceite vinagre	cuece la pasta en agua hirviendo con sal y un chorrito de aceite escurre la pasada por agua f
13	ensalada de pasta y cordero	200 gramos de cordero en trocitos un tomate berros cebolleta remolacha pasta aceite vinag	primero mezcla la pasta cocida con el cordero picado luego sazona y aliña a continuación dis
14	espaguetis a la albahaca	400 gramos de espaguetis 2 dientes de ajo 100 gramos de parmesano 100 gramos de albah	primero debes hervir los espaguetis con agua y sal después haz una mezcla en el mortero a
15	espaguetis con gambas y espinacas	300 gramos de espaguetis 400 gramos de espinacas guindilla aceite 200 g de gambas 2 ag	cuecen los espaguetis en agua sal y aceite en otro cazo las espinacas con agua y sal durante
16	espaguetis con huevo	300 gramos de espaguetis 3 huevos un sobre de queso rallado 80 gramos de mantequilla p	cuecen los espaguetis en abundante agua con sal y aceite de 10-12 minutos escurre la pasta
17	garbanzos con tomates	300 gramos de garbanzos unas verduras cebolla puerro y zanahoria dos clavos una cebolla	pone en remojo los garbanzos en la víspera cuece los en agua caliente con sal y un chorro d
18	garbanzos con arroz y espinacas	300 gramos de garbanzos 100 gramos de arroz 100 gramos de espinacas agua sal una ber	pon los garbanzos en remojo en la noche anterior en una cazuela con agua y sal pon a cocer
19	garbanzos al azafrán	300 gramos de garbanzos una cucharada de aceite de oliva virgen 3 tomates maduros un p	remoja los garbanzos durante 12 horas después colócalos en una cazuela con agua hirviendo
20	hojaldre relleno	500 gramos de hojaldre congelado 200 gramos de queso azul 300 g de espinacas cocidas t	haz una bechamel añade el queso y remueve hasta fundir lo agrega también los huevos cocid
21	judías estofadas con tomillo	400 gramos de judías blancas una cabeza de ajo una hoja de laurel dos o tres ramas de tom	pon las judías en remojo en la noche anterior cuece las tapadas a fuego lento en una cazuela
22	lasaña de atún	9 hojas de lasaña bechamel 400 gramos de atún en aceite 4 tomates maduros 3 dientes de	cuece la pasta en abundante agua con sal y aceite refresca y reservada filete a los ajos y dór
23	lasaña de calabacines	6 láminas de lasaña cocida tres cuartos kilogramos de calabacines medio vaso de nata líquid	lava seca y corta en lonchas muy finas de los calabacines al dente en una vaporera con agua
24	lasaña de pescado y marisco	12 placas de lasaña 300 gramos de pescado que tengas para aprovechar 8 langostinos 3 cl	cuece la pasta en agua con sal y resérvala en agua fría sofríe la cebolleta picada y el diente d
25	lentejas con verduras	500 gramos de lentejas 2 ajos aceite un calabacín una cebolla o cebolleta un puerro un pimi	limpia y corta la cebolleta el calabacín la zanahoria el puerro el pimiento verde los dientes de
26	lentejas guisadas	1 kg de lentejas una morcilla una cebolla picada una cucharada de pimentón picante aceite	pon a cocer las lentejas saladas estarán cocidas en unos 40 o 45 minutos aparte cuece la mo
27	lentejas con alitas de pollo	300 gramos de lentejas una cebolleta una cebolla roja 3 dientes de ajo un pimiento morrón	cuecen las lentejas previamente remojadas en agua con una cebolleta sal y un chorro de ace
28	macarrones con bacalao	300 gramos de macarrones una cebolleta o cebolla 300 gramos de bacalao desalado de un	cuece los macarrones en abundante agua con sal y 2 cucharadas de aceite y una vez cocido
29	macarrones con carne	400 gramos de macarrones 200 gramos de carne picada de ternera y cerdo una cebolleta s	cuece la pasta en una coctelera con abundante agua y sal durante unos 20 minutos aproxima
30	macarrones con tomate	250 g de macarrones 100 gramos de bacon o jamón curado 100 gramos de chorizo una cel	cuece los macarrones en abundante agua hirviendo con un chorro de aceite y sal escurre y r
31	pizza de brécol	300 g de masa de pizza 750 gramos de brécol un litro de agua un diente de ajo 100 gramos	lava el brécol saca sus brotes y cuécelo en agua hirviendo con sal una vez cocido escurre lo
32	pizza marinera	una oblea de pizza 50 gramos de atún 25 mejillones un cuarto de litro de salsa de tomate 5	un está bien con la salsa de tomate en la base de la pizza coloca el atún desmigado y la carn
33	pizza napolitana	dos discos de base de pizza 4 tomates maduros o salsa de tomate 14 anchoas en conserva	pica el tomate y extiéndelo sobre las bases de pizza después espolvorea con orégano y sal re
34	sopa de calabaza	una calabaza limpia de medio kilo aproximadamente sal y pimienta azafrán una cebolleta o	corta la calabaza en tacos pequeños y ponla a cocer en agua con sal unos 10 o 15 minutos a
35	sopa de cebolla	1,5 l de caldo de verduras de ave 3 cebolletas aceite de oliva 12 rebanadas de pan tostado	prepara un caldo de ave o de verduras dejando cocer dos o tres horas sus
36	sopa de melón y gambas	1 kg de melón 8 gambas cocidas un chorro de nata líquida nuez moscada sal 200 gramos d	corta los melones por la mitad intentando dar forma dentada para luego servir la sopa en la c
37	tortilla especial de patata	6 huevos 4 patatas 2 cebolletas una lata pequeña de bonito en aceite salsa de tomate perej	fríe las patatas peladas y troceadas en una sartén con abundante aceite junto con la cebollet
38	tortilla con tomate	4 tomates 2 cebolletas 5 huevos sal aceite medio sobre de levadura un ajo 6 triángulos peq	escalda pela y quita las pepitas a los tomates y troceados en una sartén con un poco de aceit

Figura B.1: Contenido del archivo recetas.csv



	Receta	token	token_POS	chunks	pp	pp_clean
0	albóndigas con champiñones	['2', 'huevos', 'sal', 'pimienta', '15', 'gramos	[('2', 'NUM'), ('huevos', 'NOUN'), ('sal', 'NC	[['2', 'huevos'], ['sal'], ['pimienta'], ['15', 'g',	[['2', 'huev'], ['sal'], ['pimient'], ['15', 'g', 'mi	[['huev'], ['sal'], ['pimient'], ['mig', 'pan', 'lech
1	alubias blancas con judías	['300', 'gramos', 'de', 'alubias', 'una', 'zana	[('300', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), ('	[['300', 'g', 'de', 'alubias'], ['una', 'zanahoria	[['300', 'g', 'alubi'], ['zanahori'], ['pimient', 'V	[['alubi'], ['zanahori'], ['pimient'], ['tomat'], ['ju
2	alubias con calabaza	['500', 'gramos', 'de', 'alubias', 'negras', '2'	[('500', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), ('	[['500', 'g', 'de', 'alubias', 'negras'], ['2', 'ce	[['500', 'g', 'alubi', 'negr'], ['2', 'cebollet'], ['p	[['alubi'], ['cebollet'], ['puerr'], ['zanahori'], ['c
3	alubias negras con verduras	['500', 'gramos', 'de', 'alubias', '4', 'puerros	[('500', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), ('	[['500', 'g', 'de', 'alubias'], ['4', 'puerros', 'fir	[['500', 'g', 'alubi'], ['4', 'puerr', 'fin'], ['8', 'ce	[['alubi'], ['puerr'], ['cebollet'], ['dias'], ['pimie
4	canelones de espinacas	['8', 'placas', 'de', 'canelones', 'medio', 'kilo	[('8', 'NUM'), ('placas', 'NOUN'), ('de', 'ADP	[['8', 'placas', 'de', 'canelones'], ['espinaca	[['8', 'plac', 'canelon'], ['espinac'], ['100', 'g'	[['plac', 'canelon'], ['espinac'], ['atun', 'aceit']
5	canelones rellenos de paté	['12', 'canelones', 'de', 'paté', '2', 'cebolleta	[('12', 'NUM'), ('canelones', 'NOUN'), ('de',	[['12', 'canelones', 'de', 'paté'], ['2', 'ceboll	[['12', 'canelon', 'pat'], ['2', 'cebollet'], ['zan	[['canelon', 'pat'], ['cebollet'], ['zanahori'], ['lit
6	canelones rellenos de verdura	['12', 'canelones', 'un', 'cuarto', 'de', 'litro',	[('12', 'NUM'), ('canelones', 'NOUN'), ('un',	[['12', 'canelones'], ['un', 'cuarto', 'de', 'litro	[['12', 'canelon'], ['cuart', 'litr', 'bechamel'],	[['canelon'], ['cuart', 'litr', 'bechamel'], ['zana
7	crema de calabacín con salmón	['500', 'gramos', 'de', 'calabacín', 'una', 'ce	[('500', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['500', 'g', 'de', 'calabacín'], ['una', 'ceboll	[['500', 'g', 'calabacin'], ['ceboll'], ['3', 'patat	[['calabacin'], ['ceboll'], ['patat'], ['salmon'], ['
8	crema de calabaza	['medio', 'kilo', 'de', 'calabaza', 'roja', '202',	[('medio', 'UNI'), ('kg', 'UNI'), ('de', 'ADP')	[['calabaza', 'roja'], ['202', 'zanahorias'], ['u	[['calabaz', 'roj'], ['202', 'zanahori'], ['puerr'	[['calabaz'], ['zanahori'], ['puerr'], ['ajo'], ['pin
9	crema de puerros con espárragos	['1', 'kg', 'de', 'puerros', '3', 'patatas', 'agua	[('1', 'DET'), ('kg', 'UNI'), ('de', 'ADP'), (	[['1', 'kg', 'de', 'puerros'], ['3', 'patatas'], ['a	[['1', 'kg', 'puerr'], ['3', 'patat'], ['agu'], ['sal'	[['puerr'], ['patat'], ['agu'], ['sal'], ['yem', 'espa
10	ensalada de alubias blancas	['medio', 'kilo', 'de', 'alubias', 'cocidas', 'y',	[('medio', 'UNI'), ('kg', 'UNI'), ('de', 'ADP')	[['alubias', 'cocidas'], ['un', 'tomate', 'picad	[['alubi', 'coc'], ['tomat', 'pic'], ['pimient', 've	[['alubi'], ['tomat'], ['pimient'], ['cebollet'], ['pi
11	ensalada de macarrones	['300', 'gramos', 'de', 'macarrones', '12', 'a	[('300', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['300', 'g', 'de', 'macarrones'], ['12', 'ancho	[['300', 'g', 'macarron'], ['12', 'ancho', 'aceit	[['macarron'], ['ancho', 'aceit'], ['aceitun'], ['c
12	ensalada de pasta y bonito fresco	['150', 'gramos', 'de', 'pasta', '200', 'gramo	[('150', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['150', 'g', 'de', 'pasta'], ['200', 'g', 'de', 'bo	[['150', 'g', 'past'], ['200', 'g', 'bonit'], ['atun'	[['past'], ['bonit'], ['atun'], ['gamb'], ['aceit'], ['
13	ensalada de pasta y cordero	['200', 'gramos', 'de', 'cordero', 'en', 'trocit	[('200', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['200', 'g', 'de', 'cordero', 'en', 'trocitos'], ['	[['200', 'g', 'corder', 'trocit'], ['tomat', 'berr'],	[['corder', 'trocit'], ['tomat'], ['cebollet'], ['rem
14	espaguetis a la albahaca	['400', 'gramos', 'de', 'espaguetis', '2', 'dien	[('400', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), ('	[['400', 'g', 'de', 'espaguetis'], ['2', 'dientes'	[['400', 'g', 'espaguetis'], ['2', 'dient', 'ajo'],	[['espaguetis'], ['dient', 'ajo'], ['parmesan'], ['
15	espaguetis con gambas y espinacas	['300', 'gramos', 'de', 'espaguetis', '400', 'g	[('300', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['300', 'g', 'de', 'espaguetis'], ['400', 'g', 'de	[['300', 'g', 'espaguetis'], ['400', 'g', 'espina	[['espaguetis'], ['espinac'], ['guindill'], ['aceit'
16	espaguetis con huevo	['300', 'gramos', 'de', 'espaguetis', '3', 'hue	[('300', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['300', 'g', 'de', 'espaguetis'], ['3', 'huevos'	[['300', 'g', 'espaguetis'], ['3', 'huev'], ['ques	[['espaguetis'], ['huev'], ['ques'], ['mantequill
17	garbanzos con tomates	['300', 'gramos', 'de', 'garbanzos', 'unas', '	[('300', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['300', 'g', 'de', 'garbanzos'], ['unas', 'verd	[['300', 'g', 'garbanz'], ['unas', 'verdur'], ['ce	[['garbanz'], ['verdur'], ['ceboll'], ['puerr'], ['za
18	garbanzos con arroz y espinacas	['300', 'gramos', 'de', 'garbanzos', '100', 'g	[('300', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['300', 'g', 'de', 'garbanzos'], ['100', 'g', 'de	[['300', 'g', 'garbanz'], ['100', 'g', 'arroz'], ['	[['garbanz'], ['arroz'], ['espinac'], ['agu'], ['sa
19	garbanzos al azafrán	['300', 'gramos', 'de', 'garbanzos', 'una', 'c	[('300', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['300', 'g', 'de', 'garbanzos'], ['una', 'cucha	[['300', 'g', 'garbanz'], ['cuchar', 'aceit', 'oliv	[['garbanz'], ['cuchar', 'aceit', 'oliv'], ['tomat']
20	hojaldre relleno	['500', 'gramos', 'de', 'hojaldre', 'congelado	[('500', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['500', 'g', 'de', 'hojaldre', 'congelado'], ['2	[['500', 'g', 'hojaldr', 'congel'], ['200', 'g', 'qu	[['hojaldr'], ['ques'], ['espinac'], ['puñ', 'piñon
21	judías estofadas con tomillo	['400', 'gramos', 'de', 'judías', 'blancas', 'ur	[('400', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), ('	[['400', 'g', 'de', 'judías', 'blancas'], ['una', '	[['400', 'g', 'jud', 'blanc'], ['cabez', 'ajo'], ['h	[['jud'], ['cabez', 'ajo'], ['hoj', 'laurel'], ['ram', '
22	lasaña de atún	['9', 'hojas', 'de', 'lasaña', 'bechamel', '400	[('9', 'NUM'), ('hojas', 'NOUN'), ('de', 'ADP'	[['9', 'hojas', 'de', 'lasaña'], ['bechamel'], ['4	[['9', 'hoj', 'lasañ'], ['bechamel'], ['400', 'g',	[['hoj', 'lasañ'], ['bechamel'], ['atun', 'aceit'], [
23	lasaña de calabacines	['6', 'láminas', 'de', 'lasaña', 'cocida', 'tres',	[('6', 'NUM'), ('láminas', 'NOUN'), ('de', 'AD	[['6', 'láminas', 'de', 'lasaña', 'cocida'], ['tre	[['6', 'lamin', 'lasañ', 'coc'], ['tres', 'cuart'], ['	[['lamin', 'lasañ'], ['cuart'], ['calabacin'], ['vas
24	lasaña de pescado y marisco	['12', 'placas', 'de', 'lasaña', '300', 'gramos	[('12', 'NUM'), ('placas', 'NOUN'), ('de', 'AD	[['12', 'placas', 'de', 'lasaña'], ['300', 'g', 'de	[['12', 'plac', 'lasañ'], ['300', 'g', 'pesc'], ['8',	[['plac', 'lasañ'], ['pesc'], ['langostin'], ['cucha
25	lentejas con verduras	['500', 'gramos', 'de', 'lentejas', '2', 'ajos', '	[('500', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['500', 'g', 'de', 'lentejas'], ['aceite'], ['un',	[['500', 'g', 'lentej'], ['aceit'], ['calabacin'], ['c	[['lentej'], ['aceit'], ['calabacin'], ['ceboll'], ['ce
26	lentejas guisadas	['1', 'kg', 'de', 'lentejas', 'una', 'morcilla', 'u	[('1', 'DET'), ('kg', 'UNI'), ('de', 'ADP'), (	[['1', 'kg', 'de', 'lentejas'], ['una', 'morcilla'],	[['1', 'kg', 'lentej'], ['morcill'], ['ceboll'], ['pic'	[['lentej'], ['morcill'], ['ceboll'], ['pic'], ['cuchar
27	lentejas con alitas de pollo	['300', 'gramos', 'de', 'lentejas', 'una', 'cebo	[('300', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['300', 'g', 'de', 'lentejas'], ['una', 'cebolleta	[['300', 'g', 'lentej'], ['cebollet'], ['ceboll', 'roj	[['lentej'], ['cebollet'], ['ceboll'], ['dient', 'ajo'],
28	macarrones con bacalao	['300', 'gramos', 'de', 'macarrones', 'una', '	[('300', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['300', 'g', 'de', 'macarrones'], ['una', 'cebo	[['300', 'g', 'macarron'], ['cebollet'], ['ceboll'	[['macarron'], ['cebollet'], ['ceboll'], ['bacala']
29	macarrones con carne	['400', 'gramos', 'de', 'macarrones', '200', '	[('400', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['400', 'g', 'de', 'macarrones'], ['200', 'g', 'c	[['400', 'g', 'macarron'], ['200', 'g', 'carn'], ['	[['macarron'], ['carn'], ['pic', 'terner'], ['cerd'],
30	macarrones con tomate	['250', 'g', 'de', 'macarrones', '100', 'gramo	[('250', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), (	[['250', 'g', 'de', 'macarrones'], ['100', 'g', 'c	[['250', 'g', 'macarron'], ['100', 'g', 'bacon'],	[['macarron'], ['bacon'], ['jamon'], ['choriz'], [
31	pizza de brécol	['300', 'g', 'de', 'masa', 'de', 'pizza', '750', '	[('300', 'NUM'), ('g', 'UNI'), ('de', 'ADP'), ('	[['300', 'g', 'de', 'masa', 'de', 'pizza'], ['750',	[['300', 'g', 'mas', 'pizz'], ['750', 'g', 'brecol']	[['mas', 'pizz'], ['brecol'], ['litr', 'agu'], ['dient',
32	pizza marinera	['una', 'oblea', 'de', 'pizza', '50', 'gramos', '	[('una', 'DET'), ('oblea', 'NOUN'), ('de', 'AD	[['una', 'oblea', 'de', 'pizza'], ['50', 'g', 'de', '	[['oble', 'pizz'], ['50', 'g', 'atun'], ['25', 'mejill	[['oble', 'pizz'], ['atun'], ['mejillon'], ['cuart', 'lit
33	pizza napolitana	['dos', 'discos', 'de', 'base', 'de', 'pizza', '4'	[('dos', 'DET'), ('discos', 'NOUN'), ('de', 'AD	[['dos', 'discos', 'de', 'base', 'de', 'pizza'], ['	[['dos', 'disc', 'bas', 'pizz'], ['4', 'tomat', 'ma	[['disc', 'bas', 'pizz'], ['tomat'], ['sals', 'tomat']
34	sopa de calabaza	['una', 'calabaza', 'limpia', 'de', 'medio', 'kil	[('una', 'DET'), ('calabaza', 'NOUN'), ('limp	[['una', 'calabaza', 'limpia'], ['sal'], ['pimient	[['calabaz', 'limpi'], ['sal'], ['pimient'], ['azafr	[['calabaz'], ['sal'], ['pimient'], ['azafran'], ['ce
35	sopa de cebolla	['1,5', 'l', 'de', 'caldo', 'de', 'verduras', 'de',	[('1,5', 'NUM'), ('l', 'NOUN'), ('de', 'ADP'),	[['1,5', 'l', 'de', 'caldo', 'de', 'verduras', 'de',	[['1,5', 'l', 'cald', 'verdur', 'ave'], ['3', 'ceboll	[['l', 'cald', 'verdur', 'ave'], ['cebollet'], ['aceit'
36	sopa de melón y gambas	['1', 'kg', 'de', 'melón', '8', 'gambas', 'cocid	[('1', 'DET'), ('kg', 'UNI'), ('de', 'ADP'), (	[['1', 'kg', 'de', 'melón'], ['8', 'gambas'], ['co	[['1', 'kg', 'melon'], ['8', 'gamb'], ['coc'], ['ch	[['melon'], ['gamb'], ['coc'], ['chorr', 'nat'], ['n
37	tortilla especial de patata	['6', 'huevos', '4', 'patatas', '2', 'cebolletas',	[('6', 'NUM'), ('huevos', 'NOUN'), ('4', 'NUM	[['6', 'huevos'], ['4', 'patatas'], ['2', 'cebollet	[['6', 'huev'], ['4', 'patat'], ['2', 'cebollet'], ['la	[['huev'], ['patat'], ['cebollet'], ['lat', 'bonit'], ['
38	tortilla con tomate	['4', 'tomates', '2', 'cebolletas', '5', 'huevos'	[('4', 'NUM'), ('tomates', 'NOUN'), ('2', 'NU	[['4', 'tomates'], ['2', 'cebolletas'], ['5', 'huev	[['4', 'tomat'], ['2', 'cebollet'], ['5', 'huev'], ['s	[['tomat'], ['cebollet'], ['huev'], ['sal'], ['aceit'],

Figura B.2: Contenido del archivo Ingredientes_pp.csv

	Receta	token	token_POS	chunks	pp	pp_clean
0	albóndigas con champiñones	['mezcla', 'bien', 'la', 'carne', 'con', 'los', 'hu	[('mezcla', 'VERB'), ('bien', 'ADV'), ('la', 'DE	[('mezcla', 'bien', 'la', 'carne', 'con', 'los', 'h	[('mezcl', 'bien', 'carn', 'huev', 'sal', 'pimien	[('mezcl', 'bien', 'carn', 'huev', 'sal', 'pimient
1	alubias blancas con judías	['las', 'alubias', 'estarán', 'en', 'remojo', 'de	[('las', 'DET'), ('alubias', 'NOUN'), ('estará	[('estarán', 'en', 'remojo', 'desde', 'la', 'visp	[('remoj', 'visper'], ['comienz'], [('pon'], ['coc	[('remoj', 'visper'], ['comienz'], [('pon'], ['coc
2	alubias con calabaza	['pon', 'las', 'alubias', 'en', 'una', 'cazuela', '	[('pon', 'VERB'), ('las', 'DET'), ('alubias', '	[('pon', 'las', 'alubias', 'en', 'una', 'cazuela',	[('pon', 'alubi', 'cazuel', 'abund', 'agu', 'fri']	[('pon', 'alubi', 'cazuel', 'agu'], ['coc', 'verdu
3	alubias negras con verduras	['pon', 'a', 'cocer', 'las', 'alubias', 'con', 'abu	[('pon', 'VERB'), ('a', 'ADP'), ('cocer', 'VERB	[('pon', 'a'], ['cocer', 'las', 'alubias', 'con', 'a	[('pon'], ['coc', 'alubi', 'abund', 'agu', 'sal'],	[('pon'], ['coc', 'alubi', 'agu', 'sal'], ['romp'], [
4	canelones de espinacas	['cuece', 'la', 'pasta', 'en', 'agua', 'con', 'sal	[('cuece', 'VERB'), ('la', 'DET'), ('pasta', 'N	[('cuece', 'la', 'pasta', 'en', 'agua', 'con', 'sa	[('cuec', 'past', 'agu', 'sal', 'chorrit', 'aceit']	[('cuec', 'past', 'agu', 'sal', 'chorrit', 'aceit'], [
5	canelones rellenos de paté	['cuece', 'la', 'pasta', 'en', 'agua', 'con', 'sal	[('cuece', 'VERB'), ('la', 'DET'), ('pasta', 'N	[('cuece', 'la', 'pasta', 'en', 'agua', 'con', 'sa	[('cuec', 'past', 'agu', 'sal', 'chor', 'aceit', 'y	[('cuec', 'past', 'agu', 'sal', 'chor', 'aceit', 've
6	canelones rellenos de verdura	['quién', 'quiere', 'cuecen', 'las', 'zanahoria', '	[('quién', 'PRON'), ('quiere', 'VERB'), ('cue	[('quiere', 'cuecen', 'las', 'zanahorias', 'las'	[('quier'], ['cuec', 'zanahori', 'espinac', 'coli	[('quier'], ['cuec', 'zanahori', 'espinac', 'colifl
7	crema de calabacín con salmón	['pela', 'los', 'calabacines', 'y', 'trocea', 'los '	[('pela', 'VERB'), ('los', 'DET'), ('calabacine	[('pela', 'los', 'calabacines', 'y'], ['trocea', 'l '	[('pel', 'calabacin'], ['troce', 'asi', 'patat', 'ce	[('pel', 'calabacin'], ['troce', 'asi', 'patat', 'cel
8	crema de calabaza	['pica', 'la', 'verdura', 'y', 'ponla', 'a', 'rehog	[('pica', 'NOUN'), ('la', 'DET'), ('verdura', '	[('ponla', 'a'], ['rehogar', 'en', 'una', 'cazuel	[('pon'], ['rehog', 'cazuel', 'vas', 'aceit', 'fue	[('pon'], ['rehog', 'cazuel', 'vas', 'aceit', 'fuel
9	crema de puerros con espárragos	['limpia', 'bien', 'los', 'puerros', 'con', 'agua	[('limpia', 'ADJ'), ('bien', 'CONJ'), ('los', 'D	[('hervir', 'y'], ['agrega', 'los', 'puerros', 'pic	[('herv'], ['agreg', 'puerr', 'pic', 'junt', 'patat	[('herv'], ['agreg', 'puerr', 'patat'], ['troc', 'sal
10	ensalada de alubias blancas	['cuando', 'estén', 'cocidas', 'las', 'alubias', 'e	[('cuando', 'CONJ'), ('estén', 'VERB'), ('coc	[('estén', 'cocidas', 'las', 'alubias', 'sin', 'qu	[('coc', 'alubi'], ['pas'], ['pas', 'escurr'], ['pas', 'agu', 'agu',	[('coc', 'alubi'], ['pas'], ['pas', 'agu', 'verdur',
11	ensalada de macarrones	['cuece', 'los', 'macarrones', 'al', 'dente', 'e	[('cuece', 'VERB'), ('los', 'DET'), ('macarro	[('cuece', 'los', 'macarrones', 'al', 'dente', 'e	[('cuec', 'macarron', 'dent', 'agu'], ['hirv', 'sa	[('cuec', 'macarron', 'dent', 'agu'], ['hirv', 'sa
12	ensalada de pasta y bonito fresco	['cuece', 'la', 'pasta', 'en', 'agua', 'hirviendo	[('cuece', 'VERB'), ('la', 'DET'), ('pasta', 'N	[('cuece', 'la', 'pasta', 'en', 'agua'], ['hirvien	[('cuec', 'past', 'agu'], ['hirv', 'sal', 'chorrit',	[('cuec', 'past', 'agu'], ['hirv', 'sal', 'chorrit',
13	ensalada de pasta y cordero	['primero', 'mezcla', 'la', 'pasta', 'cocida', 'c	[('primero', 'ADV'), ('mezcla', 'VERB'), ('la',	[('mezcla', 'la', 'pasta', 'cocida', 'con', 'el', 'c	[('mezcl', 'past', 'coc', 'corder', 'pic', 'lueg',	[('mezcl', 'past', 'corder', 'lueg'], ['aliñ', 'cont
14	espaguetis a la albahaca	['primero', 'debes', 'hervir', 'los', 'espaguet	[('primero', 'ADV'), ('debes', 'VERB'), ('her	[('debes'], ['hervir', 'los', 'espaguetis', 'con',	[('deb'], ['herv', 'espaguetis', 'agu', 'sal', 'de	[('deb'], ['herv', 'espaguetis', 'agu', 'sal', 'de
15	espaguetis con gambas y espinacas	['cuecen', 'los', 'espaguetis', 'en', 'agua', 's	[('cuecen', 'VERB'), ('los', 'DET'), ('espagu	[('cuecen', 'los', 'espaguetis', 'en', 'agua', '	[('cuec', 'espaguetis', 'agu', 'sal', 'aceit', 'ca	[('cuec', 'espaguetis', 'agu', 'sal', 'aceit', 'ca
16	espaguetis con huevo	['cuecen', 'los', 'espaguetis', 'en', 'abundar	[('cuecen', 'VERB'), ('los', 'DET'), ('espagu	[('cuecen', 'los', 'espaguetis', 'en', 'abunda	[('cuec', 'espaguetis', 'abund', 'agu', 'sal', '	[('cuec', 'espaguetis', 'agu', 'sal', 'aceit', 'mi
17	garbanzos con tomates	['pone', 'en', 'remojo', 'los', 'garbanzos', 'el	['pone', 'VERB'), ('en', 'ADP'), ('remojo', 'N	['pone', 'en', 'remojo', 'los', 'garbanzos', 'el	[('pon', 'remoj', 'garbanz', 'visper'], ['cuec', '	[('pon', 'remoj', 'garbanz', 'visper'], ['cuec', '
18	garbanzos con arroz y espinacas	['pon', 'los', 'garbanzos', 'en', 'remojo', 'en	[('pon', 'VERB'), ('los', 'DET'), ('garbanzos	[('pon', 'los', 'garbanzos', 'en', 'remojo', 'en	[('pon', 'garbanz', 'remoj', 'noch', 'anterior',	[('pon', 'garbanz', 'remoj', 'noch', 'cazuel', 'a
19	garbanzos al azafrán	['remoja', 'los', 'garbanzos', 'durante', '12', '	[('remoja', 'VERB'), ('los', 'DET'), ('garbanz	[('remoja', 'los', 'garbanzos', 'durante', '12',	[('remoj', 'garbanz', '12', 'hor', 'despues', 'cazue	[('remoj', 'garbanz', 'hor', 'despues', 'cazue
20	hojaldre relleno	['haz', 'una', 'bechamel', 'añade', 'el', 'que',	['haz', 'VERB'), ('una', 'DET'), ('bechamel'	[('haz', 'una', 'bechamel'], ['añade', 'el', 'qu	[('haz', 'bechamel'], ['añad', 'ques', 'remuev	[('haz', 'bechamel'], ['añad', 'ques', 'remuev
21	judías estofadas con tomillo	['pon', 'las', 'judías', 'en', 'remojo', 'en', 'la'	['pon', 'VERB'), ('las', 'DET'), ('judías', 'NO	[('pon', 'las', 'judías', 'en', 'remojo', 'en', 'la	[('pon', 'jud', 'remoj', 'noch', 'anterior'], ['cu	[('pon', 'jud', 'remoj', 'noch'], ['cuec', 'tap', 'fi
22	lasaña de atún	['cuece', 'la', 'pasta', 'en', 'abundante', 'agu	[('cuece', 'VERB'), ('la', 'DET'), ('pasta', 'N	[('cuece', 'la', 'pasta', 'en', 'abundante', 'ag	[('cuec', 'past', 'abund', 'agu', 'sal', 'aceit', '	[('cuec', 'past', 'agu', 'sal', 'aceit'], ['reserv',
23	lasaña de calabacines	['lava', 'seca', 'y', 'corta', 'en', 'lonchas', 'm	[('lava', 'NOUN'), ('seca', 'ADJ'), ('y', 'CON	[('corta', 'en', 'lonchas', 'muy', 'finas', 'de',	[('cort', 'lonch', 'fin', 'calabacin', 'dent', 'vap	[('cort', 'lonch', 'calabacin', 'dent', 'vaporer',
24	lasaña de pescado y marisco	['cuece', 'la', 'pasta', 'en', 'agua', 'con', 'sal	[('cuece', 'VERB'), ('la', 'DET'), ('pasta', 'N	[('cuece', 'la', 'pasta', 'en', 'agua', 'con', 'sa	[('cuec', 'past', 'agu', 'sal'], ['reserval', 'agu	[('cuec', 'past', 'agu', 'sal'], ['reserval', 'agu
25	lentejas con verduras	['limpia', 'y', 'corta', 'la', 'cebolleta', 'el', 'c	[('limpia', 'ADJ'), ('y', 'CONJ'), ('corta', 'VE	[('corta', 'la', 'cebolleta', 'el', 'calabacín', 'la	[('cort', 'cebollet', 'calabacin', 'zanahori', 'pi	[('cort', 'cebollet', 'calabacin', 'zanahori', 'pu
26	lentejas guisadas	['pon', 'a', 'cocer', 'las', 'lentejas', 'saladas'	[('pon', 'VERB'), ('a', 'ADP'), ('cocer', 'VER	[('pon', 'a'], ['cocer', 'las', 'lentejas', 'salada	[('pon'], ['coc', 'lentej', 'sal'], ['coc', '40', '45	[('pon'], ['coc', 'lentej'], ['coc', 'minut', 'apart
27	lentejas con alitas de pollo	['cuecen', 'las', 'lentejas', 'previamente', 'r	[('cuecen', 'VERB'), ('las', 'DET'), ('lentejas	[('cuecen', 'las', 'lentejas', 'previamente', 'r	[('cuec', 'lentej', 'previ', 'remoj', 'agu', 'ceb	[('cuec', 'lentej', 'previ', 'agu', 'cebollet', 'sal
28	macarrones con bacalao	['cuece', 'los', 'macarrones', 'en', 'abundar	[('cuece', 'VERB'), ('los', 'DET'), ('macarro	[('cuece', 'los', 'macarrones', 'en', 'abunda	[('cuec', 'macarron', 'abund', 'agu', 'sal', '2'	[('cuec', 'macarron', 'agu', 'sal', 'cuchar', 'ac
29	macarrones con carne	['cuece', 'la', 'pasta', 'en', 'una', 'coctelera'	[('cuece', 'VERB'), ('la', 'DET'), ('pasta', 'N	[('cuece', 'la', 'pasta', 'en', 'una', 'coctelera	[('cuec', 'past', 'cocteler', 'abund', 'agu', 'se	[('cuec', 'past', 'cocteler', 'agu', 'sal', 'minut'
30	macarrones con tomate	['cuece', 'los', 'macarrones', 'en', 'abundar	[('cuece', 'VERB'), ('los', 'DET'), ('macarro	[('cuece', 'los', 'macarrones', 'en', 'abunda	[('cuec', 'macarron', 'abund', 'agu'], ['hirv', '	[('cuec', 'macarron', 'agu'], ['hirv', 'chorr', 'a
31	pizza de brécol	['lava', 'el', 'brécol', 'saca', 'sus', 'brotes', 'y	[('lava', 'VERB'), ('el', 'DET'), ('brécol', 'NO	[('lava', 'el', 'brécol'], ['saca', 'sus', 'brotes'	[('lav', 'brecol'], ['sac', 'brot'], ['cuecel', 'agu	[('lav', 'brecol'], ['sac', 'brot'], ['cuecel', 'agu'
32	pizza marinera	['un', 'está', 'bien', 'con', 'la', 'salsa', 'de', '	['un', 'DET'), ('está', 'VERB'), ('bien', 'ADV	['está', 'bien', 'con', 'la', 'salsa', 'de', 'toma	['bien', 'sals', 'tomat', 'bas', 'pizz'], ['coloc',	['bien', 'sals', 'tomat', 'bas', 'pizz'], ['coloc',
33	pizza napolitana	['pica', 'el', 'tomate', 'y', 'extiéndelo', 'sobre	['pica', 'NOUN'), ('el', 'DET'), ('tomate', 'N	['reparte', 'las', 'aceitunas', 'las', 'anchoas',	['repart', 'aceitun', 'ancho', 'ques', 'cuchar',	['repart', 'aceitun', 'ancho', 'ques', 'cuchar',
34	sopa de calabaza	['corta', 'la', 'calabaza', 'en', 'tacos', 'pequ	['corta', 'VERB'), ('la', 'DET'), ('calabaza', '	['corta', 'la', 'calabaza', 'en', 'tacos', 'pequ	[('cort', 'calabaz', 'tac'], ['ponl'], ['coc', 'agu',	[('cort', 'calabaz', 'tac'], ['ponl'], ['coc', 'agu',
35	sopa de cebolla	['prepara', 'un', 'caldo', 'de', 'ave', 'o', 'de',	[('prepara', 'VERB'), ('un', 'DET'), ('caldo',	[('prepara', 'un', 'caldo', 'de', 'ave', 'o', 'de'	[('prep', 'cald', 'ave', 'verdur'], ['dej'], ['coc',	[('prep', 'cald', 'ave', 'verdur'], ['dej'], ['coc',
36	sopa de melón y gambas	['corta', 'los', 'melones', 'por', 'la', 'mitad', '	[('corta', 'VERB'), ('los', 'DET'), ('melones',	[('corta', 'los', 'melones', 'por', 'la', 'mitad'],	[('cort', 'melon', 'mit'], ['intent'], ['dar', 'form'	[('cort', 'melon', 'mit'], ['intent'], ['dar', 'form'
37	tortilla especial de patata	['fríe', 'las', 'patatas', 'peladas', 'y', 'trocea	['fríe', 'VERB'), ('las', 'DET'), ('patatas', '	['fríe', 'las', 'patatas', 'peladas', 'y'], ['troce	[('fri', 'patat', 'pel'], ['troc', 'sarten', 'abund',	[('fri', 'patat'], ['troc', 'sarten', 'aceit', 'cebolle
38	tortilla con tomate	['escalda', 'pela', 'y', 'quita', 'las', 'pepitas', '	[('escalda', 'NOUN'), ('pela', 'NOUN'), ('y', '	[('quita', 'las', 'pepitas', 'a', 'los', 'tomates',	[('quit', 'pepit', 'tomat', 'troc', 'sarten', 'aceit'	[('quit', 'pepit', 'tomat', 'sarten', 'aceit', 'poch

Figura B.3: Contenido del archivo Elaboración_pp.csv

Bibliografía

[Arguiñano, 2011] Arguiñano, K. (2011). *1.069 recetas*. BAINET.

[Bedre, 2020] Bedre, R. (2020). t-sne in python. https://reneshbedre.github.io/blog/tsne.html.

[Bird et al., 2009] Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*, chapter Preface. O'Reilly Media.

[Burrueco,] Burrueco, D. t-sne. https://www.interactivechaos.com/manual/tutorial-de-machine-learning/t-sne.

[Jackson and Moulinier, 2002] Jackson, P. and Moulinier, I. (2002). *Natural Language Processing for Online Applications: Text Retrieval, Extraction, and Categorization*, chapter Linguistic tools, pages 9–17. John Benjamins.

[Keith, 2019] Keith, D. (2019). A brief history of natural language processing (nlp). https://www.dataversity.net/a-brief-history-of-natural-language-processing-nlp/.

[Liddy, 2001] Liddy, E. D. (2001). *Encyclopedia of Library and Information Science*, chapter Natural Language Processing. Marcel Decker.

[Ling et al., 2015] Ling, W., Dyer, C., Black, A., and Trancoso, I. (2015). Two/too simple adaptations of word2vec for syntax problems. *Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL*, page 1299–1304.

[Perone, 2013] Perone, C. S. (2013). Machine learning :: Cosine similarity for vector space models (part iii). https://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/.

[Rehurek, 2017] Rehurek, R. (2017). gensim documentation. https://buildmedia.readthedocs.org/media/pdf/gensim/stable/gensim.pdf.

[Ruiz, 2018] Ruiz, G. (2018). Introducción a word2vec (skip gram model). https://medium.com/@gruizdevilla/introducción-a-word2vec-skip-gram-model-4800f72c871f.

[Sharma, 2020] Sharma, A. (2020). Top 10 applications of natural language processing (nlp). https://www.analyticsvidhya.com/blog/2020/07/top-10-applications-of-natural-language-processing-nlp/.

[Vásquez et al., 2009] Vásquez, A. C., Huerta, H. V., and Quisp, J. P. (2009). Procesamiento de lenguaje natural. *Revista de ingeniería de sistemas e informática*, pages 46–48.

[Wikipedia, 2020] Wikipedia, t. f. e. (2020). Producto escalar. https://es.wikipedia.org/wiki/Producto_escalar.