DETECCIÓN AUTOMÁTICA DE GRUPOS SEMÁNTICOS EN CORPUS TEXTUALES

Grado en Ingeniería Informática

Escuela de Ingeniería Informática
Autora: Irene María Rodríguez Morales
Tutor: Francisco Javier Carreras Riudavets

Irene María Rodríguez Morales
ULPGC

Agradecimientos

Abuelo, cuando te fuiste una parte de ti se quedó conmigo y espero que en el cielo una parte de mí esté contigo porque cuando te fuiste me fui. Te quiero.

Resumen

Este proyecto consiste en hacer un algoritmo capaz de extraer grupos de palabras relacionadas semánticamente de cualquier corpus textual usando los siguientes recursos y/o herramientas: recursos léxicos, diccionario de significados e ideológico, sinónimos, relaciones morfológicas entre las palabras, servicio de desambiguación funcional, etc.

Este algoritmo será usado en buscadores textuales, por temas o conceptos, en donde el usuario podrá elegir cierto grado de rigidez o flexibilidad en la relación semántica que poseen los términos de un mismo grupo. Por ejemplo, si el usuario elige el 20% se mostrarán las relaciones encontradas mediante sinónimos y/o antónimos. Si elige el 60%, se mostrarán las encontradas por diccionarios, etc.

El algoritmo será una herramienta de inteligencia artificial muy útil para aplicaciones del procesamiento del lenguaje natural y podrá iniciar su tarea sin ninguna información adicional. Además, se implementa una aplicación de escritorio para que el usuario tenga un programa en el que pueda elegir una carpeta con fichero/s .txt para analizar extrayendo sus grupos semánticos haciendo así más visual la utilidad del algoritmo.

Las tecnologías que utiliza este proyecto incluyen Visual Studio, que utiliza .NET y C#, y el lenguaje de programación Python.

Abstract

This project consists of making an algorithm capable of extracting groups of semantically related words from any textual corpus using the following resources and/or tools: lexical resources, meaning and ideological dictionary, synonyms, morphological relationships between words, functional disambiguation service, etc.
This algorithm will be used in textual search engines, by themes or concepts, where the user will be able to choose a certain degree of rigidity or flexibility in the semantic relationship that the terms of the same group have. For example, if the user chooses 20% the relationships found through synonyms and/or antonyms will be displayed. If the user chooses 60% those found by dictionaries will be displayed, etc.

The algorithm will be a very useful artificial intelligence tool for natural language processing applications and will be able to start its task without any additional information. In addition, it is implemented a desktop application so that the user has a program in which they can choose a folder with .txt file/s to analyze, extracting their semantic groups, thus making the utility of the algorithm more visual.

The technologies that this project uses includes Visual Studio, which uses .NET and C#, and Python programming language.

Contenido

Índice de figuras

1. Introducción

Un campo semántico es un conjunto de palabras que están relacionadas semánticamente. En otras palabras, un campo semántico son palabras que comparten un mismo significado o tienen una relación semántica cercana, un significado próximo. En este proyecto, cuando se menciona el término "semántica", se refiere a "semántica léxica" que se define según la RAE, *Real Academia Española*, como "*Rama de la semántica que estudia el significado de las palabras, así como las diversas relaciones de sentido que se establecen entre ellas*" [1].

Las palabras pertenecientes a un mismo campo semántico pueden estar relacionadas por diferentes razones. Una de ellas es el concepto conocido como "sinonimia" el cual la RAE define como "*Condición de sinónimo*" [2]. Esto es que una palabra respecto de la otra comparte un mismo significado o muy equivalente. Otro motivo por el que las palabras de un mismo campo semántico pueden estar relacionadas es por la "antonimia" que, a diferencia de la sinonimia, una palabra respecto de otra tiene un significado completamente diferente. La RAE define un antónimo como "*Dicho de una palabra: Que, respecto de otra, expresa una idea opuesta o contraria*" [3]. Además, las palabras de un mismo campo semántico pueden estar agrupadas debido a su hiperonimia e hiponimia. Es decir, la hiperonimia es la palabra que conceptualmente engloba otras palabras que son hipónimos. Los hipónimos se refieren a palabras que son más específicas que otra palabra de la que son parte que es el hiperónimo. Por ejemplo, "manzana" es un hipónimo de "fruta" que es un hiperónimo. A su vez, "fruta" puede ser el hipónimo de "comida" o "alimento". Esta relación más profunda se tendrá en cuenta en este proyecto mediante el uso de diccionarios ideológicos que a lo largo de la memoria se irán explicando.

Una vez teniendo estos conceptos claros, se puede investigar de qué forma un algoritmo, que es de lo que se trata principalmente este trabajo, puede encontrar relaciones semánticas entre palabras. Para ello, se pueden seguir los siguientes puntos:

1. Seleccionar una palabra o término base.
2. Identificar los campos semánticos a los que pertenece la palabra o término base.
3. Utilizar una herramienta de procesamiento de lenguaje natural, como Stanford NLP [4], *Natural Language Processing*, WordNet [5] u otras bibliotecas similares, para analizar y extraer información sobre las relaciones semánticas entre palabras. A lo largo de la memoria, se verá por cuál herramienta se optó y el porqué.
4. Usar los resultados obtenidos para representar gráficamente o imprimir las relaciones semánticas entre palabras, como sinónimos, antónimos, hiperónimos, hipónimos, cohipónimos, entre otros.
5. Evaluar y validar los resultados obtenidos para garantizar su precisión y utilidad.

Es importante tener en cuenta que los resultados pueden variar dependiendo de la herramienta de procesamiento de lenguaje natural utilizada y de la complejidad de la tarea. Por lo tanto, es necesario evaluar y ajustar los resultados obtenidos en función de las necesidades específicas de cada aplicación.

Para identificar los campos semánticos a los que pertenece una palabra o término base, hay varias estrategias que se pueden seguir:

1. Análisis semántico manual: Consiste en analizar la definición de la palabra y sus contextos de uso para determinar a qué conceptos está asociada.
2. Uso de diccionarios y enciclopedias: Estos recursos pueden proporcionar información sobre los campos semánticos a los que pertenece una palabra.
3. Análisis automatizado: Se puede utilizar herramientas de procesamiento de lenguaje natural, como bibliotecas de NLP, para identificar los campos semánticos a los que pertenece una palabra. Estas herramientas a menudo utilizan técnicas y análisis de frecuencia de palabras para identificar las relaciones semánticas entre términos.

En cualquier caso, es importante tener en cuenta que la identificación de los campos semánticos puede ser subjetiva y que diferentes fuentes pueden clasificar las palabras de manera diferente. Por lo tanto, es importante evaluar cuidadosamente la precisión y fiabilidad de la información obtenida. Un problema importante se encuentra en la polisemia. La RAE define la polisemia como "*Pluralidad de significados de una expresión lingüística*" [6], lo que quiere decir que una palabra puede tener varios significados.

Lo mejor es utilizar todos los recursos posibles para asegurar un buen resultado.

No es posible implementar un algoritmo para identificar los campos y las relaciones semánticas entre palabras en un único código. Sin embargo, aquí están los cinco pasos para identificar los campos y las relaciones semánticas:

1. Preprocesamiento: Realizar una limpieza y normalización de los datos de texto para extraer solo las palabras relevantes. Aquí entrará el término de palabra en forma canónica. La forma canónica de una palabra es la palabra quitándole su género y su número, en el caso de los verbos es su infinitivo. Es decir, si se tiene "niñito" su palabra canónica es "niño". En el caso de los verbos, si se tiene "caminando" su palabra canónica es "caminar".
2. Construcción del corpus: Construir un corpus a partir del texto preprocesado para tener una representación de los datos en un formato adecuado para su análisis.
3. Análisis semántico: Aplicar técnicas de procesamiento de lenguaje natural y análisis semántico para extraer información semántica del corpus, como palabras relacionadas, sinónimos y antónimos. En este apartado, se hará uso de las estrategias anteriormente mencionadas.
4. Identificación de campos semánticos: Utilizar técnicas de agrupamiento y clasificación para identificar los campos semánticos a los que pertenecen las palabras.
5. Visualización: Visualizar los resultados del análisis semántico. En este proyecto, se realizará una aplicación de escritorio.

Estos últimos cinco pasos son los que se llevarán a cabo para implementar dicho algoritmo de detección automática de grupos semánticos en corpus textuales.

1.1 Estado actual

Actualmente, no se encuentra disponible ningún algoritmo, programa o modelo, utilizado para entrenar redes neuronales, para obtener los campos semánticos presentes en un texto. Es por ello, que se ha tenido que hacer una ardua investigación antes de comenzar a realizar este proyecto.

La inteligencia artificial está en constante evolución y sus usos se han hecho más presentes en nuestros días. Se está utilizando en diversos campos que van desde, por ejemplo, el sector empresarial hasta la medicina manejando grandes cantidades de datos y automatizando tareas repetitivas ayudando, así, a la toma de decisiones y la resolución de problemas complejos causando que los humanos puedan enfocarse en otras actividades más creativas y únicas que provoquen un cambio positivo en nuestra sociedad.

En este trabajo final de grado, no se podrá trabajar con redes neuronales debido a que no existe ningún modelo para entrenar que tenga un conjunto de palabras. En otras palabras, no se ha creado ningún modelo de palabras para trabajar mediante redes neuronales en ningún idioma. No obstante, se ha investigado mucho acerca de este tema en este trabajo para encontrar recursos capaces de resolver el problema propuesto.

En primer lugar, se encuentra el recurso WordNet que es una red de palabras, o base de datos, que contiene muchas palabras relacionadas semánticamente mediante conjuntos llamados *"synsets"* en donde se almacenan, por ejemplo, los sinónimos de una palabra. También, asocia antónimos, hiperónimos e hipónimos, que serán los utilizados junto a los sinónimos en este proyecto, entre otros. Es un gran recurso léxico que contiene diferente información sobre sustantivos, verbos, adjetivos y adjetivos. En este caso, trabajaremos con sustantivos, verbos y adjetivos ya que son las principales categorías gramaticales que nos aporta información para identificar campos semánticos entre palabras.

Se decidió utilizar este recurso y se comenzó a buscar información sobre él. En un principio, se encontró una base de datos WordNet en idioma español, pero estaba realizado por tres alumnas de la Universidad Autónoma de Barcelona para realizar su proyecto final de carrera [7]. Por lo tanto, los resultados según sus estudios eran muy buenos, pero había algunos fallos sin corregir. No daba mucha seguridad y robustez utilizar este recurso, aunque estuviese muy bien, porque se podría cometer pequeños errores y se quiere hacer este proyecto lo mejor posible utilizando una base más sólida y perfeccionada.

Entonces, se siguió investigando y se encontró la librería *NLTK, natural language toolkit,* de Python que incorpora WordNet [8]. Esta librería solo puede ser utilizada mediante lenguaje Python y este proyecto se realiza en C#. Sin embargo, si se necesita acceder a las funciones de NLTK, se puede llamar a la biblioteca NLTK desde C# a través de una API o utilizando un puente de Python para C#, como Python for .NET [9] o IronPython [10]. También se podría usar la clase System.Diagnostics.Process [11]. NLTK está disponible en español. Esta librería se encuentra muy completa y actualizada por los desarrolladores.

No obstante, existe una implementación de NLTK en C# que se llama SharpNLP [12]. Pero no es tan completo y no está tan actualizado como la implementación original de NLTK en Python.

También existe la librería Stanford NLP, pero fue entrenada principalmente en el idioma inglés. Stanford NLP devuelve sinónimos, antónimos e hipónimos de una palabra, como WordNet.

Otra librería que se descubrió investigando es Freeling [13] pero no tiene lenguaje español. Una solución a estos problemas, si se decide usar alguna herramienta en inglés, es utilizar la biblioteca Microsoft Translator Text API [14] para traducir palabras a otros idiomas. La API proporciona una solución de traducción automática que se integra con la plataforma Microsoft Translator. Esta solución podía funcionar, pero no hacía que el algoritmo fuera eficiente en términos de ejecución causando de esta manera mucho tiempo de espera para obtener la salida.

Existen otras librerías que sí soportan el lenguaje español, pero no son tan eficientes como WordNetLib [15].

Por estas razones, se decidió finalmente, utilizar NLTK para Python.

El proyecto será utilizado por el IATEXT, Instituto Universitario de Análisis y Aplicaciones Textuales de la ULPGC (Universidad de las Palmas de Gran Canaria). El Instituto ha otorgado dos recursos muy eficientes y que trabajan perfectamente para utilizar en el algoritmo propuesto. Uno de estos recursos es el Servicio de Lematización que es un servicio web que lematiza palabras del español permitiendo obtener la forma canónica de cada palabra y su categoría gramatical, entre otras opciones, pero para este proyecto solo tendremos en cuenta esos dos usos. Y, por otro lado, otro servicio concedido es el Servicio de Familias el cual permite conocer la palabra de la cual proviene una palabra, palabra primitiva, y las palabras derivadas de una palabra.

Por último, este Instituto cuenta con diccionarios de sinónimos y antónimos, diccionarios de definiciones y diccionario de ideas afines y/o ideológicos que se utilizarán para desarrollar el programa.

Con estas herramientas actuales que se han encontrado e investigado, se desarrollará el siguiente *TFG*, trabajo fin de grado.

1.2 Objetivos iniciales

Se quiere realizar un algoritmo que sea lo suficientemente inteligente para que detecte campos semánticos en textos. Los recursos que se utilizarán son los que se mencionaron anteriormente.

En un principio, no se contemplaba la posibilidad de realizar una interfaz gráfica debido al tiempo, pero se ha podido llevar a cabo su implementación.

En la primera pantalla, el usuario tendrá que elegir un directorio en donde se encontrará los ficheros para analizar en formato .txt. Una vez elegido, se mostrará una ventana que enseñará el mensaje "Cargando" mientras se carga la barra de proceso.

En la pantalla final, el usuario tendrá cuatro botones, cada botón representa el grado de rigidez y precisión en el conjunto de palabras de un campo semántico. Por ejemplo, en el primer botón saldrán los sinónimos/antónimos/hipónimos/hiperónimos encontrados usando WordNet y el diccionario de sinónimos y antónimos. En el segundo botón, estos grupos se complementarían utilizando el servicio de familias encontrando sinónimos/antónimos/hipónimos/hiperónimos también y el diccionario de sinónimos y antónimos. En el tercer botón, se usarán los resultados anteriores más el diccionario de ideas afines e ideológico. Y en el cuarto botón, se unirán los tres resultados anteriores añadiendo el diccionario de definiciones. Estos resultados finales se guardan en otro fichero .txt nuevo.

Otro de los objetivos más importantes, es el poder crear por primera vez un algoritmo que sea capaz de detectar campos semánticos en cualquier texto porque nunca se realizado. Esto supone el objetivo de superar este reto con constancia, paciencia e investigando muchas horas el cómo se puede resolver dicho problema.

1.3 Justificación de las competencias específicas cubiertas

Una de las competencias específicas cubiertas que se tiene en cuenta en este proyecto, según se describe en la guía docente de la asignatura TFG, es la siguiente:

CI1: Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

Se puede justificar esta competencia debido a la capacidad que se debe tener para imaginar la resolución de este problema planteado. Se necesita pensar cómo se diseñará el algoritmo y diseñarlo de forma muy clara para no cometer errores desde el principio que hagan retroceder al punto inicial. Se debe tener el concepto de lo que se quiere y conocer bien de los recursos de los que se disponen. Se deben seleccionar los resultados y evaluarlos para verificar que son fiables y que dan seguridad y calidad al programa. Los servicios cedidos por el IATEXT no se pueden compartir a ninguna persona, solo lo puede utilizar la persona que realiza este TFG y no puede compartirlo con nadie ya que tiene que cumplir principios éticos y la norma que se establece en el acuerdo de TFG.

CI8: Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados

Esta es otra competencia que se cumple en este proyecto ya que este algoritmo será utilizado en diferentes aplicaciones en el futuro y debe obtener los mejores resultados posibles dando robustez, eficiencia y seguridad. Cuando se comenzó este proyecto, solo se tenía la idea de utilizar C#, pero investigando se descubrió que el lenguaje más apropiado para utilizar WordNet es Python.

1.4 Aportaciones

1.4.1 Entorno socioeconómico

Son muchas las veces en que los traductores tienen que analizar largos textos encontrando relaciones entre palabras en diferentes idiomas. Este algoritmo es de gran utilidad para ellos, ya que pueden encontrar campos semánticos en español ahorrando así tiempo al hacerlo automáticamente. Podrían ejecutar el programa, ver el resultado y comprobar manualmente si los resultados son correctos o no. No se puede asegurar al 100% que el algoritmo encontrará todos los campos semánticos de un texto, pero en este proyecto se ha tenido de idea siempre poner palabras de más que de menos. Por lo tanto, en la mayoría de los casos el algoritmo tiene una alta probabilidad de hallarlos. Es decir, todas las palabras relacionadas del texto aparecerán en algún conjunto de los campos semánticos si encontró relación, el único fallo que puede ocurrir es que la palabra se haya añadido en un campo semántico equivocado.

Asimismo, será una herramienta que utilizará el IATEXT para obtener automáticamente resultados El IATEXT creó una página en la que se pueden observar los campos semánticos de diferentes textos, pero se realizó manualmente. Este algoritmo lo haría automáticamente lo que permitiría a los investigadores y miembros del Instituto acelerar el proceso de análisis de textos.

También, se puede utilizar como una herramienta educativa en centros escolares. Cuando los alumnos estén aprendiendo a identificar campos semánticos durante un ejercicio y quieran corregir si lo hicieron bien, podrán utilizar la herramienta para comprobar si acertaron o fallaron. Además, el profesor podrá corregir el ejercicio de manera más rápida si se trata de un examen.

Igualmente, puede ser una aplicación muy útil en temas de investigación policial. Teniendo esta base, se puede crear un modelo y crear un conjunto de palabras "*peligrosas*". Estas palabras podrían ser "bomba", "muerte" y "pistola" y se relacionarían con el terrorismo. Si se extrajera mediante otras herramientas, como es el *scraping* [16], el texto de una página web muy extensa y se quiere analizar si hay peligro, este algoritmo sería muy útil. Se recogería el texto de la página web haciendo un script o scraping y se pasaría la información a un fichero .txt. Todo esto se haría automáticamente. Luego, si salen resultados con palabras peligrosas o se ha modificado el algoritmo base para que solo salgan palabras peligrosas, se podría confirmar que el texto trata sobre el "terrorismo" y así la policía estaría al tanto de la situación e investigaría. También, podría utilizarse en recogida de información de redes sociales de mensajes que publican las personas o en conversaciones móviles para evitar el suicidio.

1.4.2 Entorno técnico

A nivel técnico este proyecto aporta nuevos recursos e investigaciones de cómo extraer palabras relacionadas de un texto. Esto puede ser especialmente útil para mejorar la precisión y la relevancia de los resultados de búsqueda en los motores de búsqueda y en otros sistemas que procesan grandes cantidades de texto. Además, la capacidad de extraer palabras relacionadas de un texto puede tener aplicaciones en la traducción automática, el análisis de sentimientos, la detección de fraudes y muchas otras áreas de la inteligencia artificial y el procesamiento del lenguaje natural. En general, el desarrollo de nuevas técnicas y recursos para la extracción de palabras relacionadas es un paso importante en la mejora de la capacidad de las computadoras para entender y procesar el lenguaje humano.

1.4.3 Entorno científico

Con este algoritmo como base, el ámbito científico puede avanzar mucho. Puede utilizar este algoritmo y crear un modelo de entrenamiento mediante redes neuronales mejorando el tiempo de búsqueda de palabras. Aporta nuevas maneras de analizar palabras, de resolución de este problema que no se había realizado y del que tan poca información se tiene.

1.5 Normativa y legislación

Para este proyecto se ha firmado el “Contrato de cesión de derechos de autor” en la que el cedente es el autor de este TFG y el cesionario el tutor. El cedente es autor en el sentido que otorga el Texto Refundido de la Ley de Propiedad Intelectual (aprobado por el Real Decreto Legislativo 1/1996, de 12 de abril) y el titular compartido de los derechos de explotación de la siguiente obra: Detección automática de grupos semánticos en corpus textuales. En adelante, dicha obra será denominada la "Obra". El cedente está interesado en ceder los derechos de explotación de la Obra y el cesionario, a su vez, está interesado en aceptar dicha cesión, por lo que, habiendo llegado las Partes, libre y espontáneamente, a una coincidencia mutua de sus voluntades, formalizan el CONTRATO DE CESIÓN DE DERECHOS DE AUTOR, en adelante, el "Contrato", al objeto de constituir y regular su acuerdo, el cual se regirá por nueve estipulaciones que se leyeron al principio de realizar este proyecto. Una de las estipulaciones más importantes, mencionada anteriormente, es que el cedente no puede dar a terceros los derechos de la Obra y tampoco puede enseñar los recursos utilizados a ninguna persona. Se comprometen a tomar todas las medidas necesarias para mantener en secreto y confidencialmente todas las informaciones y/o conversaciones que se hayan obtenido durante la realización del TFG.

1.6 Estructura de este documento

Este documento se debide en cuatro partes fundamentales que se exponen a continuación:

1. Introducción: en este apartado se escribe sobre el estado actual, los objetivos de este TFG, justificar las competencias específicas que cubre, indicar las aportaciones que genera a nivel socioeconómico, técnico y científico y la normativa y legislación,
2. Desarrollo: en esta sección se explica detalladamente cómo se ha realizado este proyecto mostrando todos los pasos que se han realizado.
3. Conclusiones: en las conclusiones se han obtenido algunos resultados que hacen que se pueda comentar sobre este proyecto y la verificación de los objetivos inicialmente propuestos.
4. Trabajos futuros: se escribe sobre posibles trabajos futuros que pueden realizarse utilizando de base este trabajo.
5. Fuentes de información: en esta última sección se pueden leer las diferentes fuentes de información que se han usado para la realización del algoritmo.

1.7 Lenguajes, tecnologías y recursos

1.7.1 Lenguajes

Para empezar, se explicarán los lenguajes que se han utilizado. Por un lado, se ha usado el lenguaje C# y Python. C# es un lenguaje de programación desarrollado por Microsoft como fragmento de su tecnología .NET [17], será explicada en este apartado a continuación, mientras que Python es un lenguaje de programación de alto nivel que destaca por su facilidad de uso ya que se puede utilizar para desarrollar todo tipo de proyectos y/o aplicaciones.

Figura 1. Logo C#

Figura 2. Logo de Python

1.7.2 Tecnologías y recursos

El entorno en donde se ha programado es Visual Studio con la versión 2022. Es un entorno de desarrollo integrado para, en este proyecto, Windows, aunque también está disponible para macOS. Es compatible con múltiples lenguajes en el que se encuentra C#.

Figura 3. Logo Visual Studio

Una de las tecnologías que se ha llevado a cabo ha sido .NET. Es una plataforma de programación de software creada por Microsoft. La tecnología .NET es un conjunto de herramientas y recursos que los desarrolladores utilizan para crear programas y servicios que pueden funcionar en diferentes dispositivos y sistemas operativos. Proporciona un espacio donde se ejecutan y funcionan estas aplicaciones, junto con una colección de bibliotecas que contienen funciones y características predefinidas que los desarrolladores pueden utilizar en sus proyectos.

Figura 4. Imagen de Microsoft explicando .NET

Cabe destacar, que se han utilizado servicios WCF [18], *Windows Communication Formation*. Los dos servicios de este tipo que se han utilizado son el "Servicio de Lematización" y el "Servicio de Familias" proporcionado por el IATEXT. El Servicio de Lematización que es un servicio web que lematiza palabras del español permitiendo obtener la forma canónica de cada palabra y su categoría gramatical, entre otras opciones, pero para este proyecto solo tendremos en cuenta esos dos usos. Y, por otro lado, otro servicio concedido es el "Servicio de Familias" el cual permite conocer la palabra de la cual proviene una palabra, palabra primitiva, y las palabras derivadas de una palabra. Estos dos servicios serán recursos indispensables para la realización de este proyecto.

Figura 5. Representación del IATEXT en la ULPGC

Por otro lado, se tiene una base de datos léxica llamada Wordnet. Esta base de datos agrupa palabras en conjuntos de sinónimos, antónimos, hiperónimos e hipónimos, y agrupa por otras relaciones, pero para este proyecto se tienen en cuenta solo esos cuatro tipos, llamados synsets.

Se ha utilizado la librería que integra Python llamada NLTK, *Natural Language Toolkit*, para encontrar sinónimos, antónimos, hiperónimos e hipónimos en la base de datos léxica Wordnet. Es decir, la librería NLTK ya integra su base de datos Wordnet disponible en español para utilizarse.

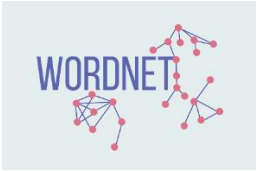

Figura 6. Representación de estructura de datos Wordnet

Figura 7. Representación logo NLTK de Python

Otro de los recursos que se puede utilizar es el diccionario de sinónimos y antónimos que permite buscar todos los sinónimos y antónimos de las palabras.

Igualmente, se tiene un diccionario de ideas afines que es también conocido como diccionario ideológico. Es un diccionario que agrupa las palabras por ideas, por conceptos, por ideologías.

Por último, se utilizará un diccionario de definiciones que contiene la definición de cada palabra incluyendo sus acepciones.

2. Desarrollo

En esta primera sección, se comentará todo lo relacionado con el desarrollo de este proyecto explicado paso a paso. En primer lugar, se explicará el estudio que se tuvo que realizar previamente para realizar el algoritmo. Seguidamente, una vez explicado los diferentes recursos, se mostrarán los pasos seguidos para la implementación del proyecto. Una vez hecho esto, se visualizará cómo se verán los resultados con su correspondiente interfaz gráfica.

2.1 Estudio previo

Para entender este proyecto de la mejor manera posible, se repasarán muchos conceptos que son necesarios entender antes de comenzar con la explicación de la implementación del algoritmo utilizando como recurso principal el diccionario de la Real Academia Española. Asimismo, se explicará, como se dijo en la sección del estado actual, los diferentes recursos elegidos y sus razones.

La RAE define campo semántico como *"Conjunto de unidades léxicas de una lengua que comparten un núcleo común de rasgos de significado"* [20]. Dicho de otra forma, es un conjunto de palabras que están relacionadas semánticamente. Es decir, son palabras que comparten un mismo significado o tienen una relación semántica cercana, un significado próximo. En este proyecto, cuando se menciona el término "semántica", se refiere a "semántica léxica" que se define según la RAE como "*Rama de la semántica que estudia el significado de las palabras, así como las diversas relaciones de sentido que se establecen entre ellas*" [21]. Con estas definiciones se puede concluir que es un conjunto de palabras que están relacionadas debido a que tienen un significado igual o parecido entre ellas. Por ejemplo, "perro" y "can" están relacionadas semánticamente porque ambas comparten que se están refiriendo al mismo mamífero. También, "perro", "can", "ser vivo", "fauna" … están relacionadas con relaciones semánticas más lejanas a las anteriores, incluso, se podría decir que la temática de este campo semántico se puede etiquetar como "animales".

Las palabras pertenecientes a un mismo campo semántico pueden estar relacionadas por diferentes razones. Una de ellas es el concepto conocido como "sinonimia" el cual la RAE define como "*Condición de sinónimo*" [22]. Esto es que una palabra respecto de la otra comparte un mismo significado o muy equivalente. Otro motivo por el que las palabras de un mismo campo semántico pueden estar relacionadas es por la "antonimia" que, a diferencia de la sinonimia, una palabra respecto de otra tiene un significado completamente diferente. La RAE define un antónimo como "*Dicho de una palabra: Que, respecto de otra, expresa una idea opuesta o contraria*" [23]. Además, las palabras de un mismo campo semántico pueden estar agrupadas debido a su hiperonimia e hiponimia. Es decir, la hiperonimia es la palabra que conceptualmente engloba otras palabras que son hipónimos. Los hipónimos se refieren a palabras que son más específicas que otra palabra de la que son parte que es el hiperónimo. Por ejemplo, "manzana" es un hipónimo de "fruta" que es un hiperónimo. A su vez, "fruta" puede ser el hipónimo de "comida" o "alimento". Esta relación más profunda se tendrá en cuenta en este proyecto mediante el uso de diccionarios ideológicos que a lo largo de la memoria se irán explicando.

Para simplificar la explicación de estos conceptos se puede sintetizar de esta manera:

1. Campo semántico: conjunto de palabras que están relacionadas debido a que sus significados son equivalentes o muy similares.
2. Sinónimo: *palabra que, respecto de otra, tiene el mismo significado o muy parecido, como empezar y comenzar* [24].
3. Antónimo: *palabra que, respecto de otra, expresa una idea opuesta o contraria, como virtud y vicio, claro y oscuro o antes y después* [25].
4. Hiperónimo: *"Palabra cuyo significado está incluido en el de otras. Pájaro es hiperónimo de jilguero y de gorrión"* [26].
5. Hipónimo: "*Palabra cuyo significado incluye el de otra. Gorrión es hipónimo de pájaro"* [27].

Para encontrar los campos semánticos que existen en un texto, se tendrá encuentra estas relaciones de sinonimia, antonimia, hiperonimia e hiponimia. Como se dijo en la introducción, se pueden obtener estas relaciones utilizando alguna herramienta de procesamiento de lenguaje natural, *NLP*, obteniendo de esta manera un análisis automatizado. Aun así, repasaremos los conceptos explicados en el apartado de la introducción para aclararlos.

Para obtener los campos semánticos de un texto, se pueden seguir los siguientes pasos:

1. Seleccionar una palabra en forma canónica, es decir, escribir el término con número singular y en género masculino.
2. Identificar los campos semánticos a los que pertenece la palabra observando las demás palabras que conforman el texto.
3. Utilizar una herramienta de procesamiento de lenguaje natural para analizar y extraer información sobre las relaciones semánticas entre palabras.
4. Usar los resultados obtenidos para representar gráficamente o imprimir las relaciones semánticas entre palabras.
5. Comprobar los resultados obtenidos para garantizar su precisión y fiabilidad.

Es importante tener en cuenta que los resultados pueden variar dependiendo de la herramienta de procesamiento de lenguaje natural utilizada y de la complejidad de la tarea. Por lo tanto, es necesario evaluar y ajustar los resultados obtenidos en función de las necesidades específicas de cada aplicación y utilizar todo el mayor número de recursos que se pueda.

Para identificar los campos semánticos a los que pertenece una palabra se pueden seguir los siguientes pasos:

1. Análisis semántico manual: Consiste en analizar la definición de la palabra y sus contextos de uso para determinar a qué conceptos está asociada.
2. Uso de diccionarios y enciclopedias: Estos recursos pueden proporcionar información sobre los campos semánticos a los que pertenece una palabra.
3. Análisis automatizado: Se puede utilizar herramientas de procesamiento de lenguaje natural, como bibliotecas de NLP, para identificar los campos semánticos a los que pertenece una palabra. Estas herramientas a menudo utilizan técnicas y análisis de frecuencia de palabras para identificar las relaciones semánticas entre términos.

Un problema importante se encuentra es la polisemia. La RAE define la polisemia como *"Pluralidad de significados de una expresión lingüística"* [28], lo que quiere decir que una palabra puede tener varios significados. Se deberá tener en cuenta este problema más adelante para que el algoritmo tenga en cuenta las acepciones de una palabra.

No es posible implementar un algoritmo para identificar los campos y las relaciones semánticas entre palabras en un único código. Sin embargo, aquí están los cinco pasos para identificar los campos y las relaciones semánticas:

1. Preprocesamiento: Realizar una limpieza y normalización de los datos de texto para extraer solo las palabras relevantes. En este proyecto, solo se tienen que escoger del texto los sustantivos, adjetivos y verbos en forma canónica.
2. Construcción del corpus: Construir un corpus a partir del texto preprocesado para tener una representación de los datos en un formato adecuado para su análisis.
3. Análisis semántico: Aplicar técnicas de procesamiento de lenguaje natural y análisis semántico para extraer información semántica del corpus, como palabras relacionadas, sinónimos y antónimos. En este apartado, se hará uso de las estrategias anteriormente mencionadas.
4. Identificación de campos semánticos: Utilizar técnicas de agrupamiento y clasificación para identificar los campos semánticos a los que pertenecen las palabras.
5. Visualización: Visualizar los resultados del análisis semántico, como grafos que muestren las relaciones entre las palabras. En este proyecto, se realizará una aplicación de escritorio.

Estos últimos cinco pasos son los que se llevarán a cabo para implementar dicho algoritmo de detección automática de grupos semánticos en corpus textuales.

Cabe destacar lo siguiente. Antes de comenzar a utilizar algún recurso, se debe extraer los sustantivos, adjetivos y verbos del texto ya que estas categorías gramaticales aportan información para encontrar campos semánticos. Una vez que se han extraído estas tres categorías gramaticales, se transforman a forma canónica, que, para recordar, es la palabra en número singular y en género masculino. Para poder obtener lo mencionado se utiliza un recurso llamado "Servicio de Lematización" proporcionado por el IATEXT. Este paso se comentará en la sección de implementación, pero se nombra en este apartado para recordar que es el primer paso que se tiene que realizar antes de usar cualquier recurso.

Una vez que se han repasado estos conceptos y los pasos a seguir para detectar campos semánticos en un texto, se puede analizar los recursos que se tienen disponibles para desarrollar el algoritmo.

En estos momentos, no se encuentra disponible ningún algoritmo, programa o modelo, utilizado para entrenar redes neuronales, para obtener los campos semánticos presentes en un texto. En otras palabras, no se ha creado ningún modelo de palabras para trabajar mediante redes neuronales en ningún idioma. No obstante, se ha investigado mucho acerca de este tema en este trabajo para encontrar recursos capaces de resolver el problema propuesto.

Se decidió finalmente utilizar NLTK para Python. Dicho esto, utilizando la librería NLTK podemos realizar un análisis automático de las palabras para encontrar relaciones de sinonimia, antonimia, hiperonimia e hiponimia.

El proyecto será utilizado por el IATEXT. El Instituto ha otorgado dos recursos muy eficientes y que trabajan perfectamente para utilizar en el algoritmo propuesto. Uno de estos recursos es el Servicio de Lematización que es un servicio web que lematiza palabras del español permitiendo obtener la forma canónica de cada palabra y su categoría gramatical, entre otras opciones, pero para este proyecto solo tendremos en cuenta esos dos usos. Y, por otro lado, otro servicio concedido es el "Servicio de Familias" el cual permite conocer la palabra de la cual proviene una palabra, palabra primitiva, y las palabras derivadas de una palabra.

Por ahora se tiene los dos servicios del IATEXT y el recurso WordNet utilizado mediante la librería NLTK en Python. Se seguirá explicando más recursos encontrados.

Otro de los recursos que se puede utilizar es el diccionario de sinónimos y antónimos que permite buscar todos los sinónimos y antónimos de las palabras. Este recurso complementa al recurso WordNet ya que, si este último no encuentra relación de sinonimia o antonimia entre palabras, este segundo recurso lo revisa. Es decir, se hace una doble comprobación lo que hace más seguro el resultado. Este diccionario es muy preciso, muy extenso y está muy bien escrito. Tiene muchos datos y mejoran muchísimo el resultado del WordNet. Es más, podría sustituir al WordNet pero como se quiere obtener un algoritmo lo más seguro posible, se utilizan los dos recursos por si acaso.

Igualmente, se tiene un diccionario de ideas afines que es también conocido como diccionario ideológico. Es un diccionario que agrupa las palabras por ideas, por conceptos, por ideologías. Este recurso es muy útil, puede ser uno de los más útiles porque está muy bien documentado y contiene muchas palabras. El diccionario aporta palabras de una manera más abstracta. Es decir, si se tiene la palabra "calor" en este diccionario puede aparecer la palabra "verano". En un principio a simple vista no parece que tengan ninguna relación semántica, pero si se quiere ver desde una visión más amplia si la tiene. Usando este recurso los resultados han mejorado en gran medida.

Por último, se utilizará un diccionario de definiciones que contiene la definición de cada palabra incluyendo sus acepciones. Anteriormente, dijimos que un problema que podía existir era el caso de la polisemia. Utilizando este diccionario, podremos saber a qué se refiere una palabra polisémica leyendo sus acepciones. Este diccionario está compuesto por la unión de seis diccionarios que se agregaron automáticamente lo que hace que se tenga mucha información de cada palabra.

Tras este profundo estudio teórico se puede decir que las herramientas a utilizar son el Servicio de Lematización y el Servicio de Familias desarrollados, ambos, por la ULPGC en el IATEXT, WordNet mediante la librería NLTK de Python, un diccionario de sinónimos y antónimos, un diccionario de definiciones y un diccionario ideológico/ideas afines. Asimismo, para llamar al script de Python que contiene la librería NLTK se utilizará la clase System.Diagnostics.Process.

A continuación, se explicará cuál es la idea principal de este algoritmo y cómo utilizar estos recursos.

2.2 Ideas principales

El resumen propone el desarrollo de un algoritmo inteligente capaz de identificar campos semánticos en textos mediante el uso de recursos como diccionarios, sinónimos, relaciones morfológicas y desambiguación funcional. Estos grupos de palabras relacionadas semánticamente serán utilizados en buscadores de temas o conceptos, permitiendo al usuario ajustar el grado de rigidez o flexibilidad en las relaciones semánticas. El objetivo es crear una herramienta de procesamiento del lenguaje natural que pueda operar sin información adicional o a partir de grupos de palabras predefinidos. Además, se planea implementar este algoritmo utilizando .NET y el lenguaje C#.

Inicialmente, no se consideraba la posibilidad de incluir una interfaz gráfica debido a restricciones de tiempo, pero finalmente se logró su implementación. En la primera pantalla, el usuario debe seleccionar un directorio que contenga los archivos en formato .txt que desea analizar. A continuación, se mostrará una ventana de carga con un mensaje de "Cargando" y una barra de progreso.

En la pantalla final, el usuario encontrará cuatro botones que representan diferentes grados de rigidez y precisión en los conjuntos de palabras de un campo semántico. Por ejemplo, el primer botón mostrará sinónimos, antónimos, hipónimos y hiperónimos encontrados utilizando WordNet y diccionarios de sinónimos y antónimos. El segundo botón complementará estos grupos utilizando el servicio de familias para encontrar sinónimos, antónimos, hipónimos e hiperónimos, además del diccionario de sinónimos y antónimos. El tercer botón utilizará los resultados anteriores junto con el diccionario de ideas afines e ideológico. Finalmente, el cuarto botón combinará los tres conjuntos anteriores y agregará el diccionario de definiciones. Los resultados finales se guardarán en un nuevo archivo .txt.

Uno de los objetivos más importantes de este proyecto es crear un algoritmo capaz de detectar campos semánticos en cualquier texto, lo cual no se ha logrado previamente. Esto implica superar este desafío con constancia, paciencia e invertir muchas horas en la investigación para resolver este problema.

2.3 Idea final

Una vez explicado el estudio previo y las ideas principales, se obtuvo el siguiente resumen.

1. Para algunas personas los campos semánticos solo lo forman las palabras cuya categoría gramatical son sustantivos mientras que, para otras personas, consideran que deberían incluirse los adjetivos y verbos obteniendo así una forma de obtener campos semánticos más generalizada. Ambas son correctas y depende de la opinión del usuario, por lo tanto, se decidió que este pudiese elegir si quiere un análisis de campo semánticos "puro", sustantivos únicamente, o un análisis de red de palabras, añadiendo a los sustantivos los verbos y adjetivos que se encuentren. Esta decisión debe tomarla en la pantalla principal de la aplicación de escritorio.
2. Dependiendo de la decisión anterior del usuario, el algoritmo deberá tener en cuenta los sustantivos o estos junto a las dos categorías gramaticales mencionadas anteriormente. Para este proceso, se utilizará el Servicio de Lematización ya que nos permite obtener la categoría gramatical que se busca, en este caso, sustantivo, adjetivo y verbo, y pasándolas a forma canónica.
3. Igualmente, en la primera pantalla, el usuario deberá elegir el directorio donde se encuentran los ficheros a analizar.
4. Llegados a este punto, se sigue con la idea principal que se tenía. Se mostrará una pantalla que posee una barra de progreso en donde se podrá observar el estado en dónde, aproximadamente, se encuentra el algoritmo, y estimar cuánto tiempo queda para acabar su ejecución. En este paso se usarán todos los recursos mencionados anteriormente que serán explicados en el apartado de la implementación del algoritmo.
5. Una vez que se termine el proceso de ejecución del algoritmo, se mostrará una tercera pantalla con los resultados. Aparecerán cuatro botones, cada botón representa el grado de rigidez y precisión en el conjunto de palabras de un campo semántico. En el primer botón saldrán los sinónimos/antónimos/hipónimos/hiperónimos encontrados usando Wordnet y el diccionario de sinónimos y antónimos. En el segundo botón, estos grupos se complementarían utilizando el servicio de familias encontrando sinónimos/antónimos/hipónimos/hiperónimos también y el diccionario de sinónimos y antónimos. En el tercer botón, se usarán los resultados anteriores más el diccionario de ideas afines e ideológico. Y en el cuarto botón, se unirán los tres resultados anteriores añadiendo el diccionario de definiciones. Estos resultados finales se guardan en otro fichero .txt nuevo. Asimismo, en la tercera pantalla se mostrará un botón llamado "Inicio" con el que se podrá realizar de cero el análisis llevando al usuario a la página principal sin necesidad de cerrar la aplicación y ejecutarla de nuevo para realizar un nuevo análisis de campos semánticos.

3. Implementación

En este apartado, se describirán los pasos a seguir para desarrollar el algoritmo, el modelo arquitectónico utilizado y se mostrarán las partes del código más interesantes. Se llamará diccionario externo a los recursos utilizados en formato .txt que se usan como diccionarios convencionales.

3.1 Lógica

Para comenzar con la implementación, se deben tener claros los distintos pasos que realizará el algoritmo. Esto se refiere al punto 4 del apartado "Idea final". Tiene que ver con la lógica, la manera de empezar a programar, cómo estructurar las ideas y utilizar los recursos de la manera más eficiente posible.

Como primer paso, se tiene que usar el Servicio de Lematización. En otras palabras, el usuario escogerá entre las dos opciones mencionadas; Análisis puro o Red de palabras. Una vez hecho esto, el programa deberá de extraer, utilizando el servicio de lematización, los sustantivos del fichero a analizar, o ficheros, y pasarlos a su forma canónica en el caso de que el usuario escogiese la opción de análisis puro. En caso, de que escoja la otra opción, deberá hacerse el mismo proceso, pero añadiendo los verbos y adjetivos. Estas palabras pasadas a forma canónica se guardan en una estructura de datos, que en este caso es una lista, List<String>.

Una vez hecho esto, se seguirán utilizando el resto de los recursos. En primer lugar, haremos un análisis utilizando Wordnet por la librería NLTK. En este paso, se buscarán si en esa lista, se encuentran relaciones de sinónimos, antónimos, hiperónimos y/o hipónimos entre las palabras que la componen. Si es así, se guardan esas relaciones encontradas en un diccionario (estructura de datos). Para buscar los sinónimos, por ejemplo, de una palabra, se pasa esa palabra como argumento a un script de Python el cual contiene la lógica para realizar dicha búsqueda.

Después, se utilizará el diccionario externo de sinónimos y antónimos para complementar el resultado anterior. El proceso es el mismo, se buscará si en ese diccionario externo, se encuentran relaciones de sinónimos y/o antónimos entre las palabras que componen la lista creada por la salida del "Servicio de Lematización", es decir, la lista que contiene las palabras del fichero en el formato adecuado para el análisis. Si es así, se guardan esas relaciones encontradas en otro diccionario (estructura de datos).

Estos dos diccionarios (estructura de datos) nuevos creados conformarán el resultado del primer botón. Entonces, se deben combinar ambos diccionarios (estructura de datos) para añadir relaciones comunes o no comunes y complementarse nuevas relaciones entre palabras ya existentes en alguno de esos diccionarios (estructura de datos) para obtener un mejor resultado. En otras palabras, la combinación de estos dos diccionarios (estructura de datos) es el resultado que se mostrará en el botón 1.

Para continuar, se usará el recurso de "Servicio de Familias". Es decir, en la lista obtenida por la salida del otro servicio se buscarán palabras derivadas y/o primitivas de cada palabra buscando relaciones entre ellas en un primer método. Si se encuentran, esas relaciones se añaden a un nuevo diccionario (estructura de datos). Asimismo, en otro método, estas palabras derivadas y/o primitivas se buscan en el diccionario de sinónimos y antónimos para buscar relaciones. Al final, la combinación de ambos métodos se unirá.

Por lo tanto, el botón 2 estará compuesto por las relaciones guardadas en el botón 1 añadiendo estas nuevas relaciones realizadas por el Servicio de Familias.

Para continuar, se tienen dos diccionarios externos de ideas afines, ideológico en otras palabras. Se buscará cada palabra de ese diccionario (estructura de datos) obtenido por el primer diccionario que contiene las palabras del fichero en formato normal en cada uno de estos diccionarios. Se creará un diccionario (estructura de datos) a partir de las relaciones encontradas entre ambos diccionarios (estructura de datos).

Entonces, el botón 3 quedará formado por la unión de las relaciones guardadas en el diccionario (estructura de datos) del botón 2 con las relaciones encontradas mediante estos dos recursos ideológicos externos.

Finalmente, el botón 4 estará formado por la unión de los resultados del botón 3 más las relaciones nuevas encontradas usando el diccionario externo de definiciones. El diccionario de definiciones es el recurso más abstracto y difícil de utilizar para reconocer si dos palabras pertenecen a un mismo campo semántico. En este proyecto, se tiene un parámetro numérico que indica cuántas coincidencias de palabras deben de tener dos definiciones para considerar que existe una relación entre ellas. En este caso, el diccionario de definiciones está lematizado. Esto quiere decir que no existen palabras de distintas categorías gramáticas a sustantivos, verbos y/o adjetivos. Por defecto, el parámetro se tiene a un valor 15 pero se puede cambiar siempre. Entre menor sea el número, menor precisión tendrá el resultado ya que es muy probable que 5 palabras en definiciones sean comunes, por ejemplo. Pero entre mayor sea el número, mayor precisión tendrá los resultados para este apartado del botón 4.

3.2 Modelo arquitectónico

El modelo arquitectónico utilizado para este TFG es el MVC (*Modelo-Vista-Controlador*). El patrón de diseño MVC es una arquitectura de software que se utiliza muchísimo para el desarrollo de diversas aplicaciones. Su principal característica es que las responsabilidades se separan en 3 componentes fundamentales; el Modelo, la Vista y el Controlador.

El modelo suele representar los datos y la lógica de negocio. En este proyecto, el modelo se compone principalmente de iniciar el proceso del algoritmo obteniendo las palabras del fichero, lematizarlas, escribir la salida de los resultados en un fichero nuevo y hacer llamadas a diferentes métodos que se encuentran en distintas clases. Es responsable de almacenar y manipular los datos, así como de realizar operaciones y cálculos relacionados con la lógica empresarial. El Modelo actúa como una capa de abstracción que encapsula la información y proporciona métodos para acceder y modificar los datos. El modelo son las palabras del fichero principalmente.

La vista es la representación visual de los datos en el modelo. Se encarga de mostrar la información al usuario final y proporciona una interfaz gráfica a través de la cual los usuarios interactúan con la aplicación. La vista se centra únicamente en la presentación de los datos y no realiza ninguna manipulación o procesamiento de estos. Serán los forms, las tres pantallas, de la aplicación.

El controlador actúa como intermediario entre el modelo y la vista. Recibe las acciones y eventos generados por el usuario a través de la Vista y los traduce en operaciones y cambios en el modelo. También se encarga de actualizar la vista con los cambios realizados en el modelo. El controlador coordina la interacción entre el modelo y la vista, asegurando que ambas partes se mantengan sincronizadas y actualizadas. El controlador de este proyecto son las clases que manejan los siguientes recursos; Wordnet, diccionario de sinónimos y antónimos, Servicio de Familias, diccionario de ideas afines, diccionario de definiciones y las uniones de los diferentes resultados.

La principal ventaja del patrón MVC es su capacidad para separar claramente las responsabilidades y mejorar la modularidad y mantenibilidad del código. Permite cambios independientes en cada componente sin afectar a los demás, lo que facilita la colaboración en equipos de desarrollo y permite una evolución más flexible de la aplicación. Además, al separar la lógica de negocio de la presentación, el MVC promueve un diseño más ordenado y facilita la reutilización de componentes.

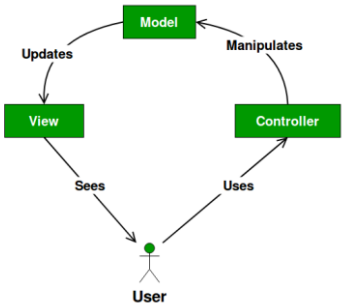

Figura 8. [24] Diagrama del funcionamiento del MVC

En el código las distintas clases quedan ordenadas de la siguiente manera:

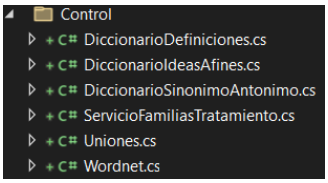

Figura 9. Visualización paquete control

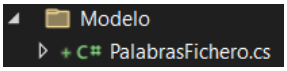

Figura 10. Visualización paquete modelo

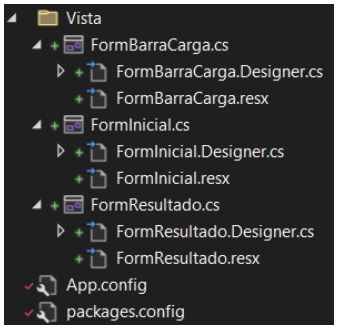

Figura 11. Visualización paquete vista

3.3 Código paso a paso

En esta sección de la memoria, se explicará el código realizado.

Para empezar, el programa comienza en la clase "FormInicial.cs", que corresponde a la primera pantalla que verá el usuario. En esta clase se encuentra el método "buttonSeleccionarDir_Click", que permite al usuario elegir el tipo de análisis y seleccionar el directorio donde se encuentra el archivo a analizar. Hay dos opciones de análisis disponibles: grupo semántico o red de palabras. El usuario elige una de las dos opciones seleccionando el botón correspondiente.

El método se encarga de abrir una ventana de diálogo con las carpetas del sistema cuando el usuario hace clic en "Seleccionar directorio". Esto le permite elegir el directorio que contiene el archivo a analizar. Una vez seleccionado el directorio, el usuario puede hacer clic en "Aceptar" o "Cancelar". Si elige "Aceptar", se abre otro diálogo para confirmar el directorio seleccionado por si desea cambiarlo. Si hace clic en "Cancelar", el diálogo con las carpetas del sistema se cierra y se vuelve a la pantalla inicial.

Además, el usuario debe elegir uno de los botones para realizar el tipo de análisis deseado. Por defecto, se selecciona el análisis de grupos semánticos "puros". Una vez que el usuario ha elegido el tipo de análisis y ha confirmado el directorio con el archivo que desea analizar, se cargan los diccionarios en memoria mediante el método "cargarDiccionarios()" de la clase "CargaDiccionarios.cs".

Una vez que los diccionarios se han cargado en memoria, se instancia la clase "PalabrasFichero.cs" pasando como parámetros la elección del usuario para el tipo de análisis y los diccionarios ya cargados. Por último, se llama al método "iniciar()" de esta última clase, pasándole la ruta del directorio seleccionado por el usuario en formato de cadena.

"**CargaDiccionario.cs**" se encuentra en el paquete de "Almacenamiento". Esta clase es la encargada de cargar los diccionarios en memoria para que la ejecución del programa sea más rápida. Se llama desde "**FormInicial.cs**" para que desde el principio los diccionarios ya estén preparados para su uso. Asimismo, cada vez que en el método "iniciar", de la clase "**PalabrasFicheros.cs**" que se explicará mejor más adelante, se llame a una tarea cuyo recurso sea un diccionario, son todos los hilos que utilizan diccionarios excepto el hilo encargado del servicio de familias, se llama a los métodos "get()" de la clase "**CargaDiccionario.cs"** que ya contienen la información de los diccionarios almacenada en memoria.

En relación con esta clase encargada de cargar diccionarios en memoria, se puede destacar que en su constructor instancia cinco diccionarios imprescindibles para este proyecto. El primero se llama "diccionarios" y es la unión de todos los diccionarios que utilizamos como recursos. La instancia "diccionarioSinonimosAntonimos" se refiere al diccionario que contiene los sinónimos y antónimos de todas las palabras. Cabe destacar que estos diccionarios se obtienen leyendo sus respectivos ficheros en formato .txt. El "diccionarioIdeasAfinesPrimero" hace referencia al primer diccionario de ideas afines que se tiene. El "diccionarioIdeasAfinesSegundo" es el relacionado con el segundo diccionario que se posee de ideas afines que complementa al anterior. Por último, el

diccionario “DiccionarioDefiniciones” es el recurso asociado con el diccionario de definiciones de las distintas palabras del idioma español.

Sin embargo, el método más importante de esta clase es el que se explica a continuación. Este método es “cargarDiccionarios” que llama a cuatro métodos auxiliares más que se encargan de leer los ficheros correspondientes a cada diccionario y almacenarlos en memoria. Asimismo, en este punto se guardan la unión de todos estos diccionarios en la variable que los engloba todos, como se dijo anteriormente, conocida como “diccionarios”.

Una vez que el método “buttonSeleccionarDir_Click” de la clase “**FormInicial.cs**” haya finalizado, se cierra ese formulario, primera ventana, y se muestra la segunda que contiene la barra de progreso.

Después, los pasos siguientes que realiza el algoritmo se hacen desde la clase modelo “**PalabrasFichero.cs**”. El camino que realiza este algoritmo se encuentra en el método “iniciar” de esta clase. Este método tiene como función realizar un proceso de carga y procesamiento de datos y buscar relaciones de palabras en segundo plano mientras se muestra una barra de progreso, que es la segunda ventana de la aplicación. Al método “iniciar()” se le pasa por parámetro en formato string la ruta del directorio que el usuario eligió para el análisis semántico.

En otras palabras, el método “iniciar” es aquel que inicia todo el proceso de análisis semántico. A partir de la ruta del directorio que ha seleccionado el usuario se crea, en primer lugar, una instancia de la vista de barra de carga, que es aquella en la que se le indica al usuario el progreso del análisis. Tras esto, se crea un hilo que se encargará de obtener las palabras del fichero deseadas según la elección de análisis seleccionada. Este hilo, se compone de más hilos que se encargan de ejecutarse paralelamente de manera asíncrona. Cada hilo de este conjunto de hilos se encarga de ejecutar una tarea relacionada con un recurso diferente para obtener relaciones de palabras. La barra de carga se va actualizando según las tareas realizadas. Se debe esperar a que los hilos acaben todo el proceso de sus tareas asociadas y esto se consigue con la instrucción Task.WaitAll(). Una vez hecho esto, se llaman a los métodos de uniones para realizar las distintas combinaciones de resultados. Para una mejor visualización del estado de la barra de progreso, los valores oscilan entre 100 y 400. Cuando se llega al valor 400, significa que el proceso ha finalizado y que, por lo tanto, se debe cerrar el formulario de la barra de carga y mostrar el formulario final con los resultados. Cabe destacar, que las tareas que utilizan diccionarios son ejecutadas cuando previamente se han cargado esos diccionarios en memoria desde la clase “**FormInicial.cs**” que se explicó precedentemente.

Es importante no olvidar que las palabras del fichero cuya categoría gramatical son del tipo sustantivos, verbos y/o adjetivos, una vez lematizadas, se guardan en una lista llamada “listaPalabrasFichero” y son las palabras que se pasan a cada recurso para buscar relaciones entre ellas. Esto se hace mediante un proceso de lematización que se explicará más abajo.

Existen cinco hilos principales dentro del método “iniciar()” que se encargan de utilizar los diferentes recursos para buscar relaciones entre las palabras. Estos hilos son los siguientes; “taskWordnet”, “taskDiccionarioSinAnton”, “taskServicioFamilias”,

“taskDiccionarioIdeasAfines” y “taskDiccionarioDefiniciones”. Estos hilos se ejecutan paralelamente y deben esperar a terminarse de ejecutar todos para seguir el proceso.

Una vez que todos los hilos de los diferentes recursos se hayan ejecutado, usando la instrucción “Task.WaitAll()”, se procede a crear las uniones entre los resultados. Se le denomina, en este proyecto, la palabra “**unión**” a la combinación y fusión de varios resultados, poniendo relaciones comunes y no comunes entre palabras, para crear más precisión en estos. Es decir, si por ejemplo si un recurso me devuelve esta relación “calor, verano,” y otro recurso me da esta relación “temperatura, verano, calor, sombrilla”, se unen para crear una única relación que contenga todas las comunes y no comunes. Entonces, el resultado final sería “calor, verano, temperatura, sombrilla”. En este caso, el algoritmo está preparado para unir estas relaciones cuando existan mínimo dos palabras iguales entre relaciones, pero se puede modificar este parámetro. Las uniones se realizan en la clase “**Uniones.cs**” que se explicará más adelante en este apartado.

Asimismo, se tiene el método obtenerPalabras() que llama al método auxiliar ProcesarFrasesSalidas(). El método obtener palabras, obtiene las palabras del fichero en grupos de 500. Si llega a 500 palabras, se llama al método auxiliar para lematizar estas palabras y se vuelven a recoger las siguientes palabras en otro grupo o grupos. Si no se llega a un grupo de 500, se llama igualmente al método auxiliar y se lematizan las palabras que se han encontrado en el fichero. Es importante darse cuenta de que este método auxiliar tiene en cuenta si el usuario ha elegido un análisis de campos semánticos puros o una red de palabras. Se recuerda que un análisis semántico es conocido por algunas personas como grupos de palabras que tienen relación y solo son sustantivos mientras que para otras personas los grupos semánticos pueden estar compuestos por sustantivos, verbos y/o adjetivos.

Por otro lado, esta clase posee un método auxiliar llamado gestionarBarraCarga() que actualiza el valor de la barra de progreso según el número de tareas que haya finalizado.

Finalmente, en esta clase se encuentra el método encargado de escribir los resultados del análisis en un fichero que se llama “GuardarSalidaEnFichero”. Este fichero tendrá de nombre la fecha de creación de este y se guardará una carpeta nueva creada llamada “Salida” que se encontrará dentro del directorio que el usuario eligió para hacer el análisis. El fichero de salida tendrá los resultados del botón 1, luego los del botón 2, debajo los del botón 3 y finalmente los resultados del botón 4. Estos resultados se pueden ver claramente diferenciados en el fichero ya que se ha programado el método para una clara escritura de estos.

Todas las clases que se explicarán a partir de ahora pertenecen a la clase “control” del método arquitectónico MVC.

El primer recurso que se utiliza de los disponibles para encontrar relaciones entre las palabras es el recurso Wordnet de la librería NLTK de Python. Este proceso del uso de este recurso se encuentra en la clase “**Wordnet.cs**”. Esta clase se compone de cuatro métodos principales; “wordnetSinonimos()”, “wordnetAntonimos()”, “wordnetHiperonimosHiponimos()” y unionSinonimosAntonimosHiperHipo(). Los tres primeros métodos mencionados tienen en común el mismo funcionamiento, solamente cambia qué relación de palabras se está buscando.

Para entender esto mejor, se empezará explicando que hace, por ejemplo, el método “wordnetSinonimos()”, que busca relaciones de sinonimia, y el método unionSinonimosAntonimosHiperHipo() que une las tres relaciones obteniendo un resultado en común.

El método “wordnetSinonimos()” utiliza un script de Python para obtener sinónimos de palabras y crea listas relacionadas de sinónimos a partir de la lista de palabras que se obtiene a partir del método obtenerPalabras() que posee las palabras del fichero según el tipo de análisis elegido. Utiliza programación paralela para procesar las palabras de forma concurrente y utiliza un diccionario para almacenar las listas relacionadas encontradas. Los otros dos métodos, “wordnetAntonimos()” y “wordnetHiperonimosHiponimos()”, hacen lo mismo lo que el primero busca relaciones de antonimia y el segundo relaciones de hiperonimia e hiponimia. Para resumir este apartado, solo se explicará el método “wordnetSinonimos()” pero que se aplica igual a los otros dos métodos. Dicho esto, este método hace lo siguiente:

1. Crea un objeto HashSet llamado processedWords para almacenar las palabras ya procesadas.
2. Utiliza un bucle Parallel.ForEach para iterar sobre la lista llamada listaPalabrasFichero.
3. Verifica si la palabra actual ya ha sido procesada. Si es así, se salta al siguiente elemento de la lista.
4. Establece la ruta de acceso al script de Python llamado "sinon.py" en la variable scriptPath. Este script se utilizará más adelante para obtener sinónimos de una palabra específica.
5. Configura las opciones de inicio del proceso para ejecutar el intérprete de Python. El nombre del archivo ejecutable es "python.exe". Se especifica el argumento del script de Python y se configuran las opciones para redirigir la salida estándar del proceso.
6. Inicia un nuevo proceso con la configuración establecida anteriormente.
7. Lee la salida estándar del proceso, que contiene los resultados de los sinónimos obtenidos del script de Python.
8. Divide la cadena de resultado en un arreglo de cadenas utilizando la coma (',') como separador. Esto proporciona una lista de sinónimos.
9. Crea una lista llamada listaSinonimos y agrega la palabra original a ella.
10. Recorre los sinónimos obtenidos y verifica si aparecen en el resto de la lista original. Si un sinónimo coincide con otra palabra de la lista, se agrega a listaSinonimos y se marca como procesada en processedWords.
11. Si listaSinonimos contiene más de una palabra, se realiza lo siguiente:

a. Verifica si la lista relacionada ya existe en un diccionario llamado listasRelacionadasSinonimos. Para ello, compara si todos los elementos de la lista existen en un par clave-valor del diccionario y si el tamaño de ambas listas es igual.

b. Si la lista relacionada no existe en el diccionario, se agrega utilizando un bloque lock para garantizar la exclusión mutua, ya que se está accediendo a una estructura de datos compartida desde múltiples hilos. Se utiliza listaRelacionadaIndex para generar una clave única para la lista relacionada.

En último lugar, se explicará el método "unionSinonimosAntonimosHiperHipo()". Este método se encarga de buscar coincidencias entre las listas relacionadas de sinónimos, antónimos e hiperónimos/hipónimos y las listas finales existentes en el diccionario listaFinalWordnet. Si encuentra una lista final que coincide con al menos dos palabras de una lista relacionada, agrega las palabras faltantes a esa lista final. Si no encuentra una lista final coincidente, crea una nueva lista final. Es decir, busca si existen palabras en común y si hay mínimo dos palabras en común une esa relación. Si hay palabras que no son comunes con ninguna relación encontrada, se deja como está en el nuevo diccionario de salida.

En otras palabras, itera sobre cada par clave-valor en el diccionario listasRelacionadasSinonimos. Dentro del bucle, busca una lista final en el diccionario listaFinalWordnet que contenga al menos dos palabras del par sinónimos actual. Para ello, itera sobre cada par clave-valor en listaFinalWordnet y cuenta cuántas palabras del par sinónimos se encuentran en la lista final. Si se encuentran al menos dos palabras, se asigna la lista final encontrada a la variable listaFinalEncontrada y se sale del bucle. Si se encuentra una lista final, se agregan al final de dicha lista las palabras del par sinónimos que no estén presentes en ella. Si no se encuentra una lista final, se genera una nueva clave (newKey) para el diccionario listaFinalWordnet y se agrega el par sinónimos completo a dicho diccionario. Se repiten los pasos 1-4 para los diccionarios listasRelacionadasAntonimos y listasRelacionadasHiperHipo.

Llegados a este punto, se puede decir que se tiene la mitad de los resultados del botón 1. Por esta razón, se pasará a explicar la clase "**DiccionarioSinonimoAntonimo.cs**" para unir este nuevo resultado con el anterior obtenido de Wordnet obteniendo, de esta manera, el resultado para el botón 1. Dicha clase se ejecuta en otro hilo paralelamente con los demás como se explicó.

La clase "**DiccionarioSinonimoAntonimo.cs**" es la que se asocia directamente con el hilo relacionado con el recurso de dicho diccionario. Seguidamente, se mencionará las partes más importantes de esta clase.

Para empezar, se puede observar en su constructor que se llama a un método llamado "operacionDiccionarioSinonimosAntonimos()". Este método método busca coincidencias entre las palabras en el diccionario "diccionarioSinonimosAntonimos" y la lista "listaPalabrasFichero". Si encuentra coincidencias, crea una lista relacionada de sinónimos para esas palabras y la agrega al diccionario nuevo creado llamado listasRelacionadasDiccionarioSinAnton.

En resumen, primero declara una variable llamada newKey que se utiliza para generar nuevas claves en el diccionario listasRelacionadasDiccionarioSinAnton. Luego, itera sobre cada línea en la lista "diccionarioSinonimosAntonimos" y divide cada línea en palabras separadas por espacios. A continuación, crea una lista de sinónimos para la palabra principal. Después, itera sobre cada palabra en la lista listaPalabrasFichero y

verifica si la línea actual del diccionario se refiere a esa palabra. Si es así, crea una lista llamada relacionados y agrega la palabra actual y sus sinónimos relacionados. Si la lista relacionados contiene al menos dos palabras, se agrega al diccionario listasRelacionadasDiccionarioSinAnton con una nueva clave newKey.

Con estos recursos ya utilizados, se obtienen los resultados que se corresponden con el botón 1. Pero esto se explicará después ya que se deben ejecutar todos los hilos para hacer las uniones pertinentes para cada botón.

Ahora, se procederá a explicar el método relacionado con el hilo que se encarga del recurso "Servicio de Familias" que se encuentra en la clase "**ServicioFamiliasTratamiento.cs**".

La clase "**ServicioFamiliasTratamiento.cs**" se compone de tres métodos principales; "servicioFamiliasBuscarDerPrim()", "servicioFamiliasBuscarEnDiccionarioDerPrim()" y "unionListaComprobacionConDiccionarioSinAnton()".

Se comenzará el primer método que es "servicioFamiliasBuscarDerPrim()". Este método crea dos diccionarios (primitivas y derivadas) para almacenar las palabras primitivas y derivadas de cada palabra en la lista de palabras listaPalabrasFichero. Itera sobre cada palabra en la lista de palabras y realiza las siguientes acciones:

1. Llama al método ConsultaPrimitivas(word) del objeto servicioFamilias para obtener una lista de objetos InfoPrimitiva que contienen información sobre las primitivas relacionadas con la palabra actual. Si la lista no es nula, se extraen las palabras primitivas y se almacenan en la lista primitivasList.
2. Llama al método ConsultaDerivadas(word) del objeto servicioFamilias para obtener una lista de objetos InfoDerivada que contienen información sobre las derivadas relacionadas con la palabra actual. Si la lista no es nula, se extraen las palabras derivadas y se almacenan en la lista derivadasList.
3. Crea una lista relacionadasList e inicialmente agrega la palabra actual a ella.
4. Verifica si alguna palabra primitiva de primitivasList o derivada de derivadasList se encuentra en la lista listaPalabrasFichero. Si se encuentra y no está en relacionadasList ni ninguna de las listas primitivas o derivadas, se agrega a relacionadasList.
5. Comprueba si la lista relacionadasList ya existe en el diccionario listaComprobarPrimitivasDerivadas. Se utiliza el método SonListasIguales() para verificar si dos listas son iguales independientemente del orden. Si la lista no existe y tiene más de un elemento, se agrega al diccionario listaComprobarPrimitivasDerivadas.

La función auxiliar SonListasIguales() se utiliza para verificar si dos listas son iguales, sin importar el orden de sus elementos. Compara la cantidad de elementos en ambas listas y luego verifica si todos los elementos de lista1 se encuentran en lista2. Si se cumple esta condición, devuelve true; de lo contrario, devuelve false.

Este método busca las palabras primitivas y derivadas para cada palabra en una lista y encuentra las palabras relacionadas en esa lista. Luego, agrupa las palabras relacionadas en listas y las almacena en un diccionario.

Cabe destacar que estos dos métodos que se explican que pertenecen a esta clase hacen uso del recurso "Servicio de Familias" otorgado por el IATEXT de la ULPGC ya que es una herramienta muy fiable y apropiada para usarse en este proyecto.

El siguiente método es "servicioFamiliasBuscarEnDiccionarioDerPrim()". El método realiza lo siguiente:

1. Itera de forma paralela sobre la lista de palabras listaPalabrasFichero.
2. Para cada palabra en la lista, realiza lo siguiente:

   a. Llama al método ConsultaPrimitivas(word) del objeto servicioFamilias para obtener una lista de objetos InfoPrimitiva que contienen información sobre las primitivas relacionadas con la palabra actual. Almacena el resultado en la variable familiaPrimitiva.

   b. Llama al método ConsultaDerivadas(word) del objeto servicioFamilias para obtener una lista de objetos InfoDerivada que contienen información sobre las derivadas relacionadas con la palabra actual. Almacena el resultado en la variable familiaDerivada.

   c. Si existe una lista de primitivas familiaPrimitiva, itera de forma paralela sobre las primitivas y realiza lo siguiente:

   - Llama al método diccionarioSinonimosAntonimos2() con los parámetros correspondientes para buscar sinónimos y antónimos en el diccionario. Se pasa la primitiva como argumento.

   d. Si existe una lista de derivadas familiaDerivada, itera de forma paralela sobre las derivadas y realiza lo siguiente:

   - Llama al método diccionarioSinonimosAntonimos2() con los parámetros correspondientes para buscar sinónimos y antónimos en el diccionario. Se pasa la derivada como argumento.

Para resumir, el método "servicioFamiliasBuscarEnDiccionarioDerPrim()" busca las primitivas y derivadas de cada palabra en la lista y, a través del método "diccionarioSinonimosAntonimos2()", busca sinónimos y antónimos en el diccionario. Luego, almacena las listas relacionadas encontradas en el diccionario listasRelacionadasDiccionarioSinAnton2.

Por último en esta clase, se tiene el método "unionListaComprobacionConDiccionarioSinAnton()" que busca coincidencias entre las listas de palabras relacionadas de sinónimos obtenidas del diccionario y las listas de palabras relacionadas de primitivas y derivadas. Es decir, uno las relaciones encontradas por los anteriores dos métodos de la clase explicados. Si se encuentran dos elementos coincidentes, se unen las listas. Si no se encuentran coincidencias, se agrega una nueva lista al diccionario.

Por otro lado, se tiene la clase "**DiccionarioIdeasAfines.cs**". Esta clase tiene tres métodos muy importantes que son "operacionDiccionarioIdeasAfines()", "operacionDiccionarioIdeasAfines2()" y "unionDosDiccionariosIdeasAfines()". La diferencia del primer método con el segundo radica en que se utilizan diferentes diccionarios de ideas afines pero la idea es similar.

Así, el primer este método busca relaciones entre palabras en un diccionario de ideas afines y crea un diccionario de relaciones (relacionesIdeasAfines). Recorre el diccionario de ideas afines línea por línea y verifica si las palabras en las líneas corresponden a las palabras de la lista listaPalabrasFichero. Si se encuentra una relación, se agrega a relacionActual, y cuando se encuentra una línea que indica el final de una relación, se agrega relacionActual al diccionario relacionesIdeasAfines.

Siempre cuando se buscan palabras en los distintos diccionarios ya cargados en memoria, se tiene que seguir un patrón textual para encontrarlas porque estos diccionarios han sido escritos por otras personas que han separados las palabras mediante, a lo mejor, símbolos, puntos, comas… Por eso, en el código se puede observar que se ha tenido en cuenta rigurosamente estos patrones textuales para buscar las relaciones de palabras lo mejor posible.

De modo semejante, y teniendo en cuenta los patrones textuales, el segundo método busca relaciones entre palabras en un diccionario de ideas afines, utilizando un formato específico en un archivo de texto.

El último método de esta clase es "unionDosDiccionariosIdeasAfines" que, como dice su nombre, combina los resultados del primer método con el segundo. La idea principal de este método es combinar dos diccionarios de ideas afines (relacionesIdeasAfines y relacionesIdeasAfines2) en uno solo, eliminando duplicados y palabras individuales, y mostrando el resultado. Se fusionan las listas que tienen al menos dos palabras en común. Este resultado se guarda en una nueva estructura de datos que es un diccionario.

El resultado de esta clase se unirá luego con el resultado que se obtenga del botón 2, que, a su vez, es la unión del botón 1 con lo nuevo. Es conveniente repasar de nuevo la idea final. El botón 1 se forma uniendo los recursos wordnet y el diccionario de sinónimos y antónimos. El botón 2 se forma uniendo el botón 1 más lo obtenido por el servicio de familias. El botón 3 se forma uniendo el botón 2 más lo obtenido por los diccionarios de ideas afines. Y el botón 4 se formará uniendo el botón 3 con lo nuevo obtenido utilizando el diccionario de definiciones que se explicará a continuación.

Queda por explicar el último hilo que hace referencia al uso de un recurso muy importante de este trabajo y es el relacionado con el diccionario de definiciones.

El diccionario de definiciones se utiliza en clase "**DiccionarioDefiniciones.cs**" el cual solo se compone de un método llamado "operacionDiccionarioDefiniciones()". Este método es el más difícil de entender, pero se intentará explicarlo de la mejor manera posible. En la sección siguiente, se comentará lo más importante de esta explicación del código paso a paso para comprender mejor el procedimiento.

El método busca palabras en el diccionario de definiciones que estén relacionadas con las palabras de la lista de palabras del archivo. Luego, genera un diccionario de palabras relacionadas. En este método hay que tener muy en cuenta si el usuario escogió un análisis semántico o un análisis de tipo red de palabras. Este diccionario al principio no estaba lematizado por lo que el algoritmo tardaba mucho en ejecutarse. Finalmente, se lematizó dejando al lado de cada palabra su categoría gramatical correspondiente. También, hay que tener en cuenta el patrón textual y formato que tiene el archivo para hacer su lectura correcta. Se explicará técnicamente los pasos de este método:

1. Se une el contenido del diccionario de definiciones en un solo string llamado contenidoArchivo.
2. Se divide el contenido del archivo en definiciones separadas utilizando el delimitador "###". El resultado se almacena en el array definiciones.
3. Se itera sobre cada palabra única en la lista de palabras del archivo (listaPalabrasFichero.Distinct()).
4. Para cada palabra actual, se crea una lista llamada listasRelacionadasDefinicionesActual para almacenar las palabras relacionadas con la palabra actual.
5. Se busca la palabra actual en las definiciones iterando sobre cada definición en definiciones.
6. Si la definición contiene la palabra actual, se procede a extraer las palabras que tienen los códigos ":1000", ":1100" y ":3000" dentro de la definición.
7. Se itera sobre cada línea de la definición y se verifica si contiene los códigos mencionados.
8. Si se encuentra una línea con el código ":1000" o, en caso de que estaMarcadoRadioRedPalabras sea verdadero, se encuentra una línea con los códigos ":1100" o ":3000", se extrae el código de la línea y se verifica si está en la lista de palabras del archivo (listaPalabrasFichero).
9. Si el código está en la lista y no coincide con la palabra actual, se agrega a la lista palabrasCodigo.
10. Se cuenta cuántas palabras en la lista palabrasCodigo aparecen en la definición.
11. Si la cantidad de palabras en común es mayor o igual a palabrasComunesEnDefiniciones, se agregan la palabra actual y las palabras de palabrasCodigo a la lista listasRelacionadasDefinicionesActual.
12. Si la lista listasRelacionadasDefinicionesActual contiene más de una palabra, se agrega al diccionario listasRelacionadasDefiniciones, utilizando una nueva clave generada, y se ordena la lista de palabras relacionadas.

Este resultado se unirá a los obtenidos para el botón 3 y formarán los resultados del botón 4.

Se han terminado de explicar las funcionalidades de los cinco hilos principales que realizan las tareas para trabajar con cada recurso. Estos hilos se llamaban, después de la ejecución de la clase "**FormInicial.cs**", dentro de la clase "**PalabrasFichero.cs**".

Como se dijo anteriormente, se utiliza la instrucción "Task.WaitAll()" para esperar a que todos los hilos terminen su ejecución. Una vez hecho esto, se obtienen los resultados de las tareas realizadas asignándolas a una variable cada una, cinco variables en este caso. Luego, se procede a realizar el proceso de uniones para obtener los resultados que se obtendrán en los diferentes botones.

Para el botón 1 se debe de unir el resultado de wordnet y el diccionario de sinónimos y antónimos. Para el botón 2, los resultados del botón 1 más el uso del servicio de familias. Para el botón 3, los resultados del botón 2 y los resultados obtenidos por los diccionarios de ideas afines. Y para el botón 4, los resultados del botón 3 más los resultados del diccionario de definiciones.

Se procede a explicar llegados a este punto la clase "**Uniones.cs**" que se encuentra también el paquete control del MVC. Esta clase posee cuatro métodos principales llamados "iniciarOperacionUnion1()", "iniciarOperacionUnion2()", "iniciarOperacionUnion3()" e "iniciarOperacionUnion4". Los cuatro realizan la misma funcionalidad, pero uniendo diferentes resultados. Para entender esto, solo se explicará el primero de ellos ya que los otros se comportan de manera igual.

El método "iniciarOperacionUnion1()" realiza una operación de unión entre dos diccionarios: listaFinalWordnet y listasRelacionadasDiccionarioSinAnton. El objetivo es combinar las listas de palabras de ambos diccionarios siguiendo ciertas condiciones y generar un nuevo diccionario llamado listaFinal1punto.A continuación, se explica el proceso paso a paso:

1. Se itera sobre cada lista en el diccionario listaFinalWordnet utilizando el bucle foreach.
2. Para cada lista en listaFinalWordnet, se verifica si esa lista ya existe en listaFinal1punto mediante una comprobación de igualdad utilizando SequenceEqual.
3. Además, se verifica si esa lista contiene palabras duplicadas mediante un conteo de ocurrencias de cada palabra en la lista.
4. Si la lista no está presente en listaFinal1punto y no contiene palabras duplicadas, se agrega a listaFinal1punto.
5. Luego, se itera sobre cada lista en el diccionario listasRelacionadasDiccionarioSinAnton utilizando el bucle foreach.
6. Para cada lista en listasRelacionadasDiccionarioSinAnton, se busca si hay una lista en listaFinal1punto que comparta al menos compartirPalabrasEnRelaciones palabras con la lista actual.
7. Si se encuentra una lista en listaFinal1punto que cumple esta condición, se agregan todas las palabras de la lista actual a esa lista en listaFinal1punto.

8. Si no se encuentra ninguna lista en listaFinal1punto que comparta las palabras requeridas, se agrega la lista actual a listaFinal1punto.
9. Después de realizar las uniones, se procede a eliminar los subgrupos de grupos en listaFinal1punto. Esto implica eliminar las listas que son subconjuntos de otras listas más grandes.
10. Finalmente, se muestra el resultado de la operación en la consola, imprimiendo cada lista en listaFinal1punto con su respectivo número de clave.

En resumen, el método toma dos diccionarios de listas y los une en un nuevo diccionario listaFinal1punto. Se realizan comprobaciones de igualdad y de duplicados en las listas, se buscan coincidencias entre las listas y se eliminan los subgrupos.

Una vez que ya se tienen los resultados en listaFinal1Punto, listaFinal2Punto, listaFinal3Punto y listaFinal4Punto se deben mostrar en la ventana de la "**FormResultado.cs**". Para ello, se hace lo siguiente:

```
// Mostrar resultados en la ventana formResultado
formBarraCarga.Invoke(new Action(() =>
{
    formResultado = new FormResultado(listaFinal1, listaFinal2, listaFinal3, listaFinal4);

    formBarraCarga.Hide();
    formResultado.Show();
}));
// Guardar resultados en un fichero de salida
GuardarSalidaEnFichero(listaFinal1, listaFinal2, listaFinal3, listaFinal4);
```

Figura 12. Visualización de cómo se llama a FormResultado, la tercera ventana

Como se puede observar, se esconde el formulario relacionado con la barra de progreso y se muestra el formulario que contendrá los resultados pasándoselos por parámetro. Cabe destacar, que mientras, los resultados se escriben en un fichero de salida que, como se explicó, tendrá como nombre la fecha de creación y se guardará en la carpeta de "Recursos" de la aplicación.

Se ha explicado todo el proceso del algoritmo, pero además se detallará cómo se han creado y que contienen el formulario de resultados y el formulario con la barra de progreso, que para el algoritmo no es muy importante, pero es necesario tenerlo en cuenta.

Por un lado, el formulario que muestra los resultados se encuentra en la clase "**FormResultado.cs**". Esta clase tiene un método llamado "RadioButtonEventHandler()" que permite mostrar los resultados según el botón que se elija de las cuatro opciones. Asimismo, tiene el método "FormResultado_FormClosed()" que cierra la aplicación cuando el usuario le de a la "X" de la ventana y el método "buttonVolverInicio_Click()" que tiene como funcionalidad volver a la primera pantalla para repetir del proceso si le das al botón que tiene escrito "Inicio" sin tener que cerrar y ejecutar de nuevo la aplicación para un nuevo análisis.

Por otro lado, se tiene la clase "**FormBarraCarga.cs**" que se encarga de gestionar la lógica de la segunda ventana correspondiente a la barra de progreso. La clase tiene tres

métodos. El primero de ellos es “getProgreso()” que devuelve el número, que indica el progreso en valor numérico, para saber porqué porcentaje de ejecución se encuentra el algoritmo. También, se tiene un método “setProceso” que es muy importante y, por lo tanto, se mostrará en código para entenderlo mejor:

```
public void setProgreso(int progreso)
{
    this.progreso = progreso;
    barraCarga.Value = progreso;
    Refresh();
}
```

Figura 13. Método setProceso de formBarraCarga

Este método estable el progreso el valor del progreso al valor que se le pasa por parámetro y asignándolo al valor de la barra de progreso. Es muy importante hacer el Refresh() porque sino el valor de la barra de progreso se quedará siempre con el valor que se le pasa como parámetro.

Por último, se tiene el método “FormBarraCarga_FormClosed” que cierra la ventana y termina la ejecución del programa si el usuario clica la “X” de la esquina superior derecha.

Este formulario se ejecuta en segundo plano, por lo tanto el usuario puede minimizar la ventana mientras se está realizando el proceso y hacer otras tareas con el ordenador y/o mover la pantalla de sitio.

3.3.1 Aclaraciones importantes

Como se ha explicado anteriormente, el método “operacionDiccionarioDefiniciones()” puede ser el más difícil de comprender. En resumen, lo que hace este método es comparar las definiciones de cada palabra de la lista y si se encuentran 15 palabras comunes entre estas dos definiciones, el algoritmo supondrá que existe una relación semántica entre ambas. Se recuerda que por defecto el parámetro para encontrar palabras comunes está a 15 pero siempre se puede cambiar. La definición de la primera palabra de la lista se comparará con el resto de las definiciones de las demás palabras. Luego, la segunda palabra de la lista con el resto… y así sucesivamente hasta llegar a la última palabra.

3.4 Código optimizado y detalles

Hay que hacer énfasis en muchos aspectos del código para tener en cuenta el gran reto que ha supuesto programarlo.

En primer lugar, tenemos una clase llamada CargaDiccionarios.cs dentro del parque Almacenamiento que se encarga guardar en memoria los diccionarios para hacer que el algoritmo se ejecute lo más rápido posible.

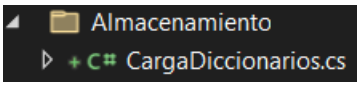

Figura 14. Visualización de Almacenamiento

Asimismo, los diccionarios utilizados para almacenar las relaciones entre palabras son de tipo Dictionary<int, List<string>>. Y la lista que contiene las palabras del fichero lematizadas es de tipo List<String>.

Para agilizar el proceso, se lematizan las palabras en grupos de 500.

```
private void obtenerPalabras(string content)
{
    StringReader stringReader = new StringReader(content);
    this.texto = new ProcesarTextos.Text("", content);
    List<InfoUnaFrase> frases = new List<InfoUnaFrase>(); // Lista para almacenar las frases a enviar al servicio
    int contadorFrases = 0; // Contador de frases

    Parallel.ForEach(texto.GetParagraphs(), paragraph =>
    {
        string parrafo = paragraph.getText() + "\r\n";

        Parallel.ForEach(paragraph.GetSentences(), sentence =>
        {
            InfoUnaFrase infoUnaFrase = new InfoUnaFrase();
            infoUnaFrase.Frase = sentence.getText();
            lock (frases)
            {
                frases.Add(infoUnaFrase);
                contadorFrases++;
            }

            // Verificar si se alcanzó el límite de frases por grupo (500)
            if (contadorFrases >= 500)
            {
                // Enviar las frases al servicio
                List<InfoUnaFrase> frasesSalidas = servicioLematizacion.ReconocerFrases(frases, "es", false);

                // Procesar las frases devueltas por el servicio
                ProcesarFrasesSalidas(frasesSalidas);

                // Limpiar la lista de frases y reiniciar el contador
                lock (frases)
                {
                    frases.Clear();
                    contadorFrases = 0;
                }
            }
        });
    });

    // Verificar si quedan frases pendientes por enviar al servicio
    if (frases.Count > 0)
    {
        // Enviar las frases restantes al servicio
        List<InfoUnaFrase> frasesSalidas = servicioLematizacion.ReconocerFrases(frases, "es", false);

        // Procesar las frases devueltas por el servicio
        ProcesarFrasesSalidas(frasesSalidas);
    }
}
```

Figura 15. Lematizar en grupos de 500

```
private void ProcesarFrasesSalidas(List<InfoUnaFrase> frasesSalidas)
{
    Parallel.ForEach(frasesSalidas, fraseSalida =>
    {
        Parallel.ForEach(fraseSalida.Palabras, palabra =>
        {
            if (palabra.IdCategoria == sustantivo && !listaPalabrasFichero.Contains(palabra.FormaCanonica))
            {
                lock (listaPalabrasFichero)
                {
                    listaPalabrasFichero.Add(palabra.FormaCanonica);
                }
            }

            if (this.radioButtonMarcado.Equals("REDPALABRAS"))
            {
                if (palabra.IdCategoria == verbo && !listaPalabrasFichero.Contains(palabra.FormaCanonica) && !palabrasNoDeseadas.Contains(palabra.FormaCanonica))
                {
                    lock (listaPalabrasFichero)
                    {
                        listaPalabrasFichero.Add(palabra.FormaCanonica);
                    }
                }
                if (palabra.IdCategoria == adjetivo && !listaPalabrasFichero.Contains(palabra.FormaCanonica))
                {
                    lock (listaPalabrasFichero)
                    {
                        listaPalabrasFichero.Add(palabra.FormaCanonica);
                    }
                }
            }
        });
    });
}
```

Figura 16. Método auxiliar que lematiza

También, se ha utilizado la clase Task (*System.Threading.Tasks*) que permite que se ejecuten varios hilos paralelamente, programación multihilo, haciendo que la ejecución de la aplicación sea más rápida. A continuación, se muestra un fragmento del código, en la clase del modelo, donde se ha empleado esta clase:

```
public void iniciar(string path)
{
    // Cargar barra de progreso
    this.formBarraCarga = new FormBarraCarga();

    // Lanza hilo secundario
    formBarraCarga.Shown += (sender, e) =>
    {
        // Hilo secundario
        Task.Run(() =>
        {
            // Obtener contenido de fichero
            String[] files = Directory.GetFiles(path);
            String content = "";
            foreach (String filename in files)
            {
                content += System.IO.File.ReadAllText(filename);
            }
            obtenerPalabras(content);

            // Actualizar barra de progreso
            gestionarBarraCarga(100);

            var taskWordnet = Task.Run(() =>
            {
                Wordnet wordnet = new Wordnet(listaPalabrasFichero);
                return wordnet.iniciarOperacionWordnet();
            });

            gestionarBarraCarga(150);

            var taskDiccionarioSinAnton = Task.Run(() =>
            {
                DiccionarioSinonimoAntonimo diccionarioSinonAnton = new DiccionarioSinonimoAntonimo(listaPalabrasFichero, cargaDic
                return diccionarioSinonAnton.iniciarOperacionDiccionarioSinAnton();
            });
```

Figura 17. Visualización parcial de la utilización de multihilos

Como se puede observar, cada tarea realizada por un recurso diferente corresponde con un hilo nuevo creado. Para esperar a que todos los hilos se ejecuten se debe utilizar la siguiente instrucción (Task.WaitAll()):

```
// Esperar a que todas las tareas se completen
Task.WaitAll(taskWordnet, taskDiccionarioSinAnton, taskServicioFamilias, taskDiccionarioIdeasAfines, taskDiccionarioDefiniciones);
```

Figura 18. Visualización de la instrucción que espera a que todos los hilos terminen su ejecución

Una vez que se ejecuten los diferentes hilos para la utilización de los diferentes recursos, se pueden unir los resultados para obtener la salida para cada botón.

```
// Obtener los resultados de las tareas
var listaFinalWordnet = taskWordnet.Result;
var listaFinalSinAnton = taskDiccionarioSinAnton.Result;
var (listaComprobacionListaFinal1, listaUnionSinAntHipeHipo2) = taskServicioFamilias.Result;
var listaFinalIdeasAfines = taskDiccionarioIdeasAfines.Result;
var listaFinalDefiniciones = taskDiccionarioDefiniciones.Result;

// Crear las uniones
// Botón 1: wordnet + diccionario sinónimos y antónimos
Uniones union1 = new Uniones();
Dictionary<int, List<string>> listaFinal1 = union1.iniciarOperacionUnion1(listaFinalWordnet, listaFinalSinAnton);

gestionarBarraCarga(340);

// Botón 2: botón 1 + servicio de familias
Uniones union2 = new Uniones();
Dictionary<int, List<string>> listaFinal2 = union2.iniciarOperacionUnion2(listaFinal1, listaUnionSinAntHipeHipo2);

gestionarBarraCarga(360);
```

Figura 19. Visualización de las uniones entre los resultados

Una vez que los hilos y las uniones hayan terminado, se mostrará la tercera pantalla con los resultados y se creará un fichero con la fecha en la que se obtuvo los resultados con la salida escrita.

Seguidamente, se mostrará los parámetros que podrá modificar el usuario dependiendo de la exactitud que desee usando el diccionario de definiciones que se encuentra en su clase correspondiente llamada DiccionarioDefiniciones.cs:

```
private int palabrasComunesEnDefiniciones = 15; // Indica las palabras comunes que existen entre definiciones
```

Figura 20. Parámetro que se puede cambiar según la precisión de búsqueda que se desee

Por otro lado, las combinaciones de diferentes diccionarios que conforman una unión, que es lo que se muestra al usuario por la interfaz gráfica, puede combinarse dependiendo de las palabras comunes que se encuentren. Es decir, si los dos diccionarios que se quieren fusionar tienen una relación que contiene dos palabras en común, se complementan uniéndose. Por defecto está a dos, pero el usuario también puede cambiarlo.

```
private int compartirPalabrasEnRelaciones = 2; // Número de palabras comunes entre listas para que se fusionen en una sola
```

Figura 21. Parámetro que se puede cambiar para fusionar dos relaciones entre diccionarios

De la misma manera, se han optimizado más clases usando hilos. Una de ellas es la clase "Wordnet.cs" en la que las búsquedas de sinónimos, antónimos e hiperónimos/hipónimos se realiza paralelamente.

```
public Dictionary<int, List<string>> iniciarOperacionWordnet()
{
    // Crear tres tareas para los tres primeros métodos
    Task taskSinonimos = Task.Run(() => wordnetSinonimos());
    Task taskAntonimos = Task.Run(() => wordnetAntonimos());
    Task taskHiperonimosHiponimos = Task.Run(() => wordnetHiperonimosHiponimos());

    // Esperar a que todas las tareas finalicen
    Task.WaitAll(taskSinonimos, taskAntonimos, taskHiperonimosHiponimos);

    // Ejecutar los dos últimos métodos
    unionSinonimosAntonimosHiperHipo();

    return this.listaFinalWordnet;
}
```

Figura 22. Método de la clase principal Wordnet optimizado

También, en la clase "ServicioFamiliasTratamiento.cs" su método principal se encuentra optimizado como se puede observar en la siguiente imagen.

```
public Dictionary<int, List<string>> iniciarOperacionServicioFamilias()
{
    Task sfTask = Task.Run(() => servicioFamiliasBuscarDerPrim());
    Task diccionarioTask = Task.Run(() => servicioFamiliasBuscarEnDiccionarioDerPrim());

    Task.WaitAll(sfTask, diccionarioTask);

    unionListaComprobacionConDiccionarioSinAnton(); // Unir primitivas y derivadas encontradas en SF + diccionario
    return listaComprobarPrimitivasDerivadas;
}
```

Figura 23. Método de la clase principal ServicioFamiliasTratamiento optimizado

Es importante señalar, que el proceso que más tarda en ejecutarse es el uso de Wordnet ya que se pierde tiempo en llamar al script de Python externo para su ejecución. No obstante, se ha intentado optimizar lo máximo posible todo el programa usando muchos hilos y Parallel.ForEach en zonas del código donde se podía sin que afectase la lógica del algoritmo.

3.5 Interfaz gráfica

Como se dijo anteriormente, la aplicación constará de 3 pantallas principales, 3 Windows Foms, que conforman la vista de nuestro modelo arquitectónico. A continuación, se muestran los respectivos diseños.

Primeramente, se muestra la pantalla principal de la aplicación donde el usuario deberá seleccionar qué tipo de análisis desea realizar y el seleccionar el directorio en donde se encuentran el/los ficheros/fichero a analizar.

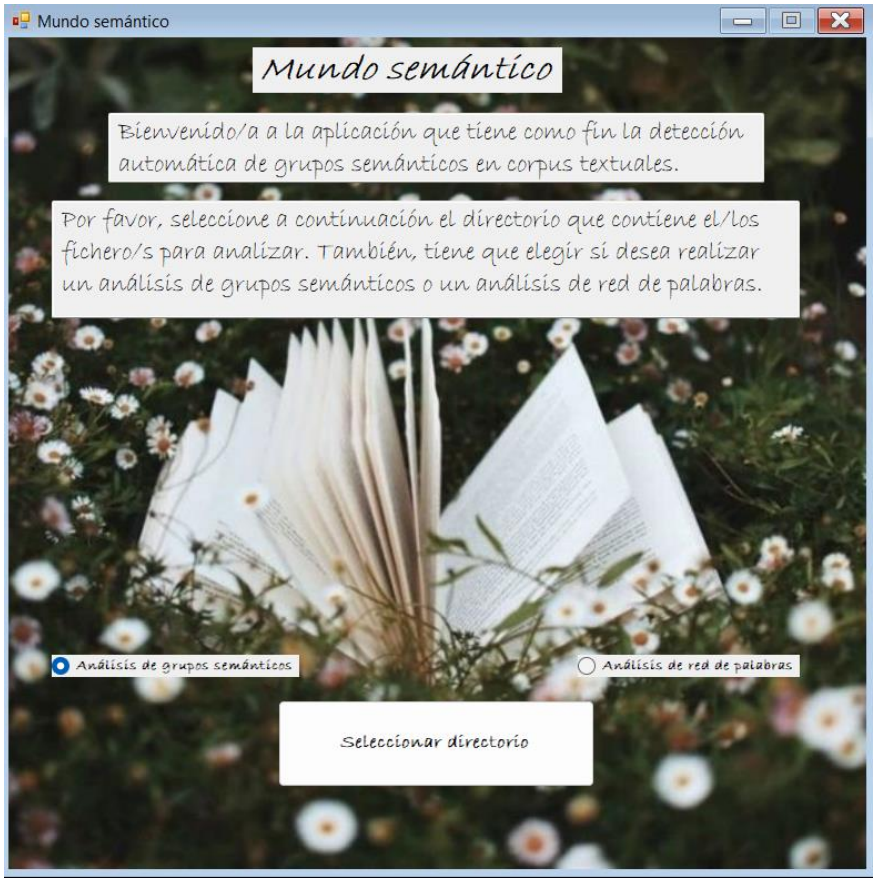

Figura 24. Pantalla principal de la aplicación

A su vez, se muestra la segunda pantalla de la aplicación que se ejecuta en segundo plano mientras el algoritmo realiza su funcionamiento.

Figura 25. Segunda pantalla de la aplicación

Finalmente, se muestra la pantalla final que contendrá los resultados en donde el usuario podrá elegir el botón correspondiente dependiendo de cuánta precisión quiere en los resultados. Asimismo, tiene un botón llamado “Inicio” el cual le permite volver a la primera pantalla y realizar todo el proceso desde cero, es decir, le permite realizar una ejecución del programa.

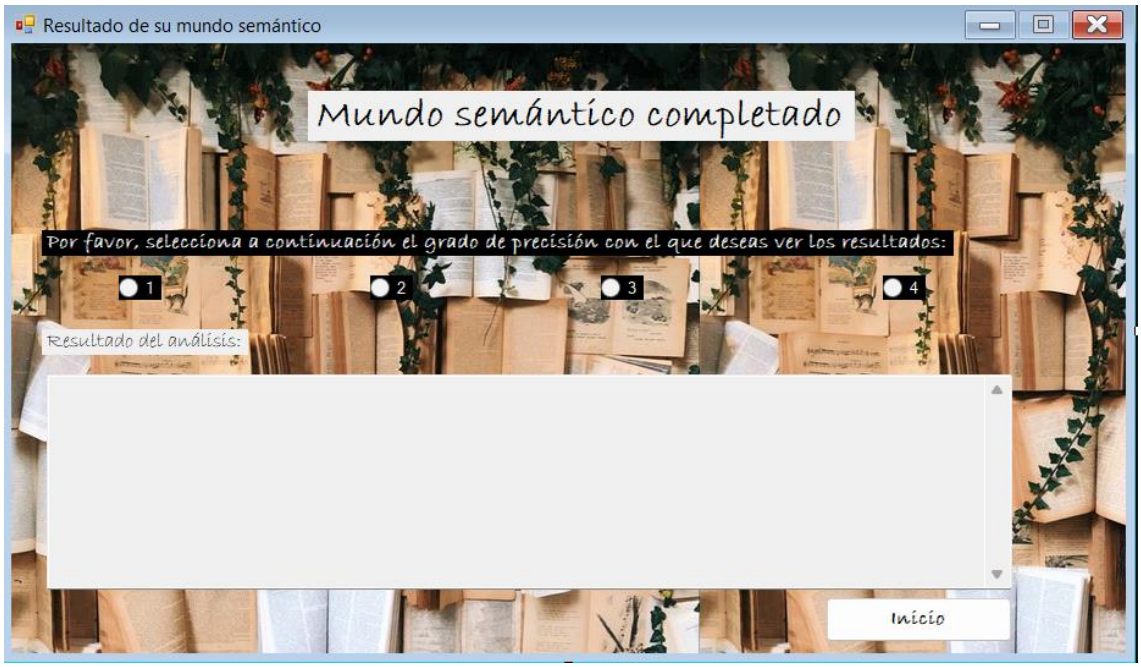

Figura 26. Tercera y última pantalla de la aplicación

Las imágenes que se han elegido no tienen licencia ni copyright y se han escogido teniendo en cuenta la temática de este trabajo de fin de grado.

4. Pruebas y resultados

El tema para tratar en esta sección consiste en poner en práctica el algoritmo para conocer su fiabilidad y sus resultados. Se harán diferentes pruebas para valorar mejor las salidas del algoritmo.

4.1 Prueba 1

4.1.1 Prueba 1.1

Se utilizará en esta prueba este texto de ejemplo: “En la bulliciosa panadería del barrio, el aroma de pan recién horneado envuelve el ambiente. Los panaderos expertos amasan con destreza la masa, creando deliciosas creaciones que despiertan el apetito. El panadero, habilidoso en su oficio, utiliza una amplia gama de palabras derivadas y primitivas para nombrar los productos: panecillos, panadería, panificación. Los clientes disfrutan de los panes frescos y los llevan a sus hogares, donde comparten gratos momentos alrededor de una mesa llena de sabores ancestrales y creativos, un verdadero festín lingüístico y gastronómico”. Se utilizará este primer texto ya que utiliza una gran variedad de palabras derivadas y primitivas y se podrá observar el buen funcionamiento de este.

En la siguiente imagen, se muestra que se empezará a realizar un análisis de grupos semánticos de un fichero que se encuentra en la carpeta “Prueba”.

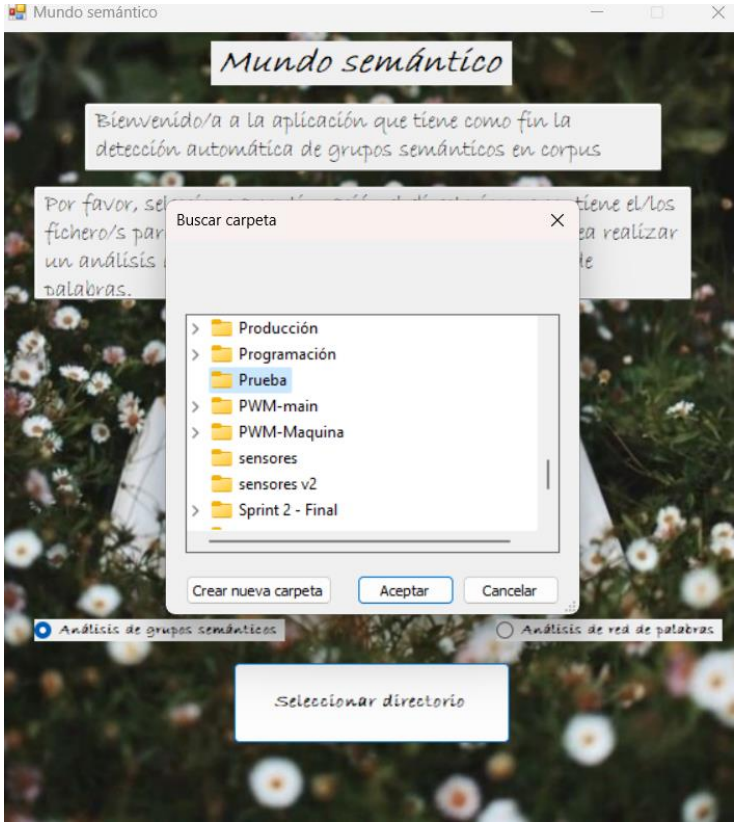

Figura 27. Elección de directorio

Seguidamente, se clicará en la opción “Aceptar” para continuar con el proceso. Luego, se pedirá al usuario que confirme la carpeta elegida.

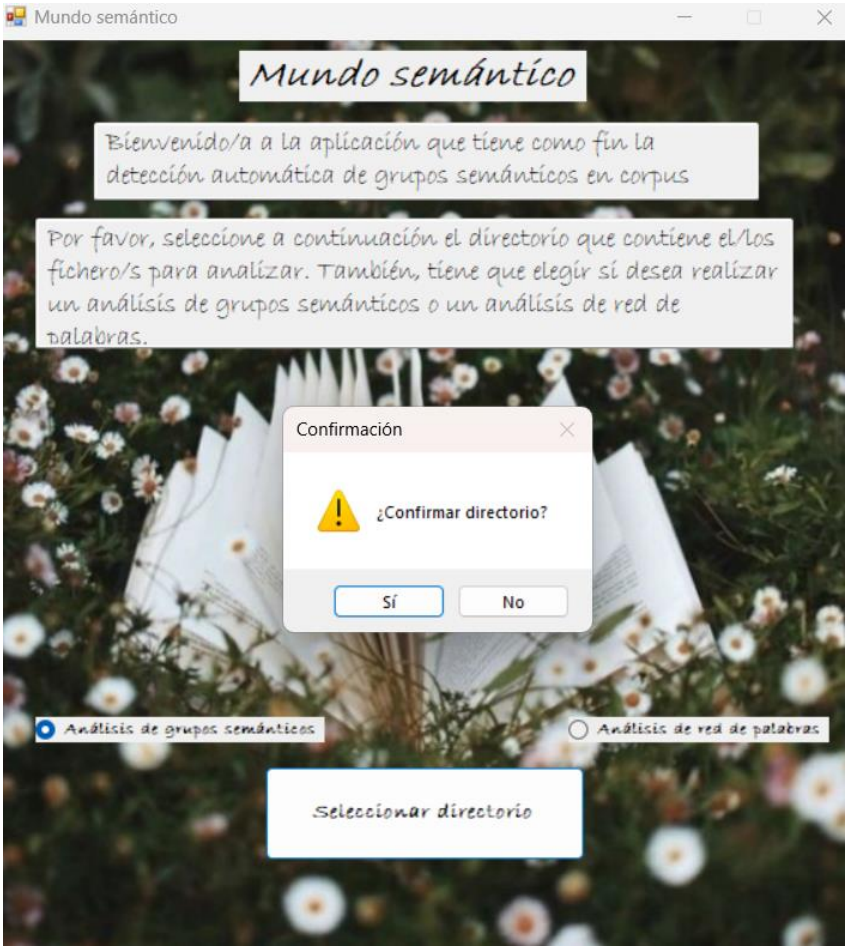

Figura 28. Confirmar directorio elegido

Una vez que se confirme el directorio clicando en la opción "Sí", se mostrará la segunda ventana con la barra de progreso. El valor de la barra de progreso se irá actualizando a medida que se ejecuten las diferentes tareas.

Figura 29. Visualización del estado de la barra de progreso

Una vez terminado los procesos, se podrán visualizar los resultados en la última ventana.

Se puede comprobar que entre mayor sea el número del botón mayores resultados hay debido al número de recursos utilizados y el grado de precisión siendo el cuarto botón el más abstracto. Este fichero tenía 85 palabras y ha tardado en ejecutarse con este análisis aproximadamente 1 minuto.

Figura 30. Resultados obtenidos botón 1

Figura 31. Resultados obtenidos botón 2

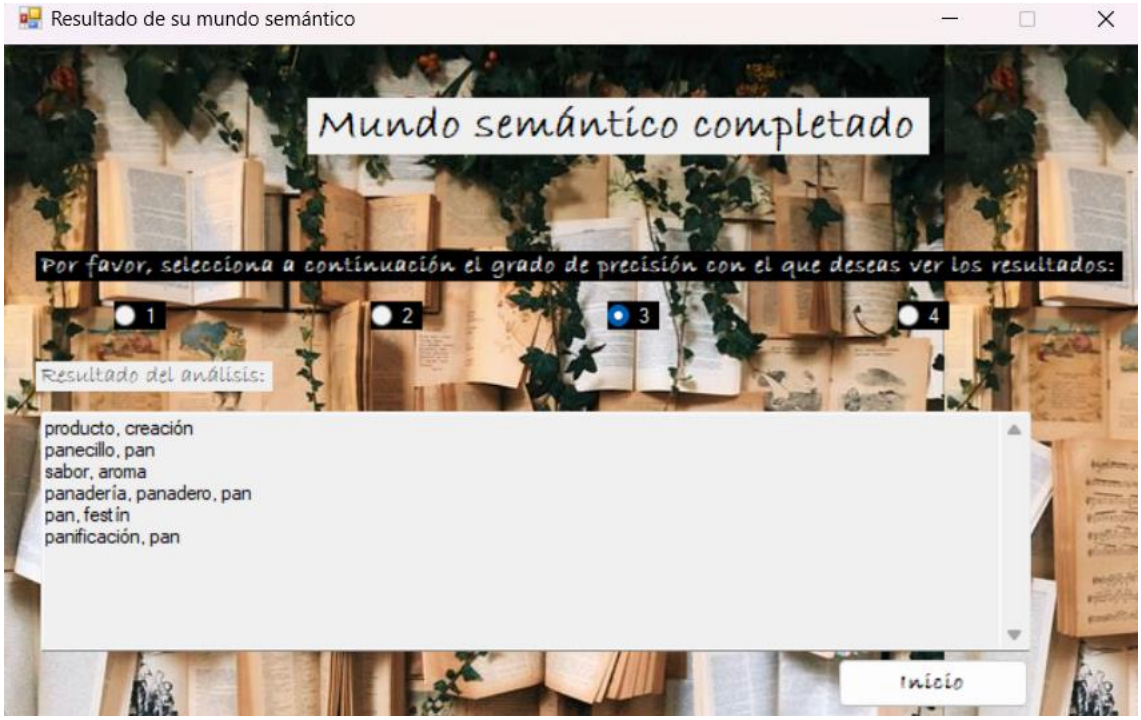

Figura 32. Resultados obtenidos botón 3

Figura 33. Resultados obtenidos botón 4

También se puede comprobar que el fichero con los resultados obtenidos se ha creado en la carpeta “Prueba”, que fue la que se eligió para este ejemplo, dentro del directorio nuevo, “Salida”, creado por el programa.

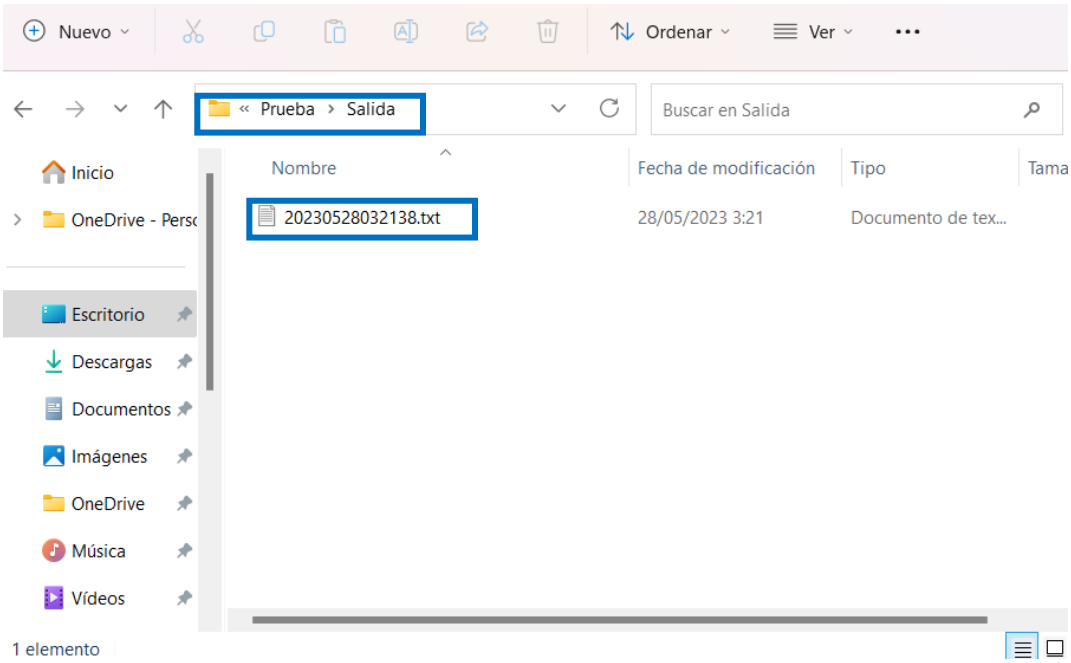

Figura 34. Creación del fichero de salida.

Además, se observan los resultados obtenidos escritos correctamente en el mismo.

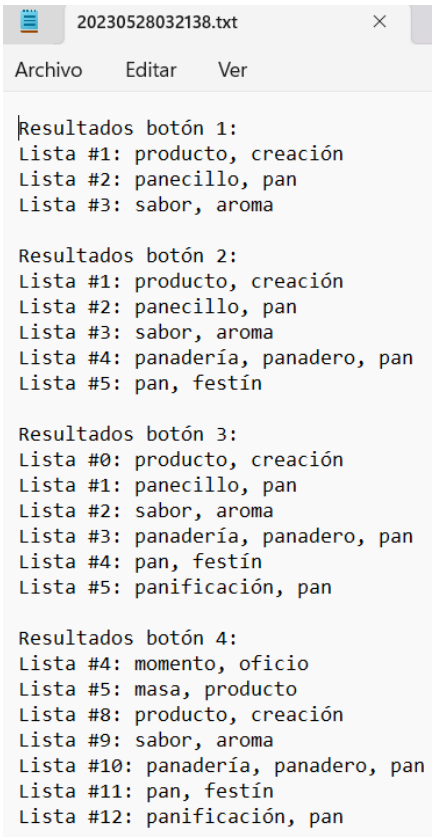

Figura 35. Visualización de los resultados escritos en el fichero

4.1.1 Prueba 1.2

En esta prueba, se va a seleccionar un tipo de análisis de red de palabras con el mismo fichero utilizado anteriormente en el mismo directorio. Los pasos para seguir son los mismos que los realizados en la prueba anterior. En la siguiente imagen se puede confirmar lo dicho y observar la elección de análisis elegida.

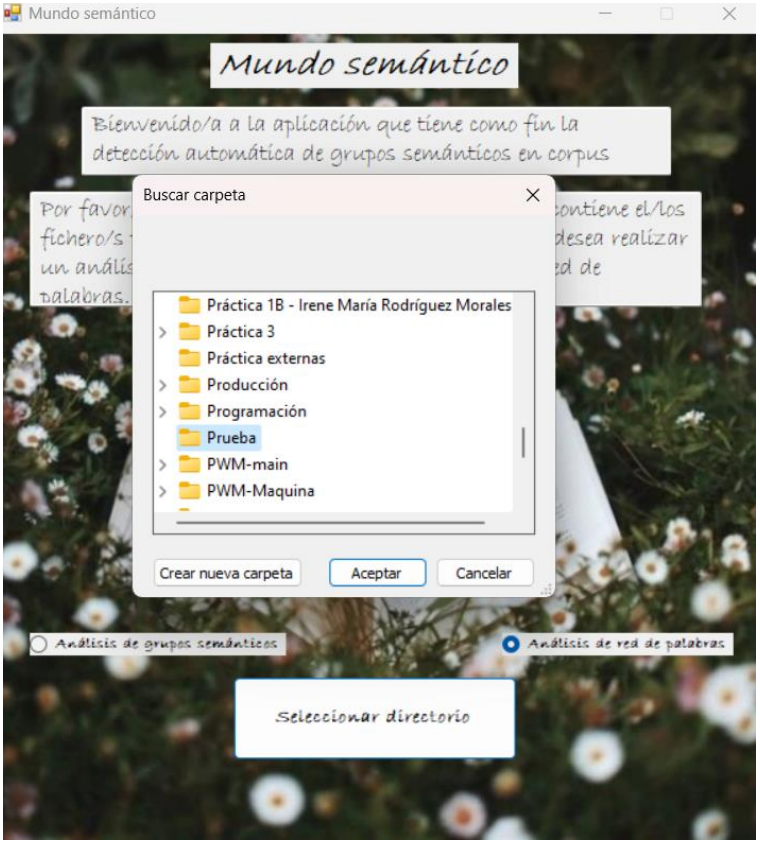

Figura 36. Elección de directorio

Para este análisis se tendrán en cuenta las tres categorías gramaticales; sustantivos, adjetivos y verbos. En la prueba de antes solo se tuvo en cuenta los sustantivos. Este análisis tarda un poco más en ejecutarse que el otro debido a que se analizan más palabras.

Una vez que se termine el proceso, se pueden comprobar los diferentes resultados obtenidos como se muestran en las siguientes imágenes. Para este mismo fichero de 85 palabras el análisis ha durado 1 minuto 20 segundos.

Figura 37. Resultados botón 1

Figura 38. Resultados botón 2

Figura 39. Resultados botón 3

Figura 40. Resto resultados botón 3

Figura 41. Resultados botón 4

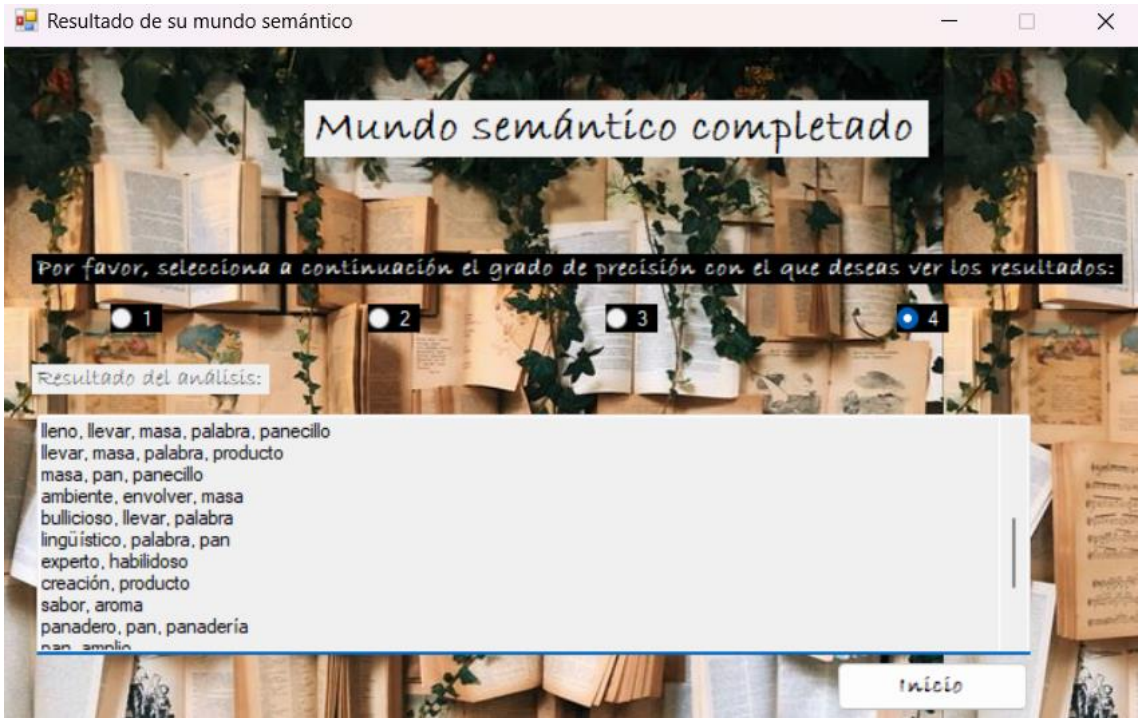

Figura 42. Resto de resultados botón 4

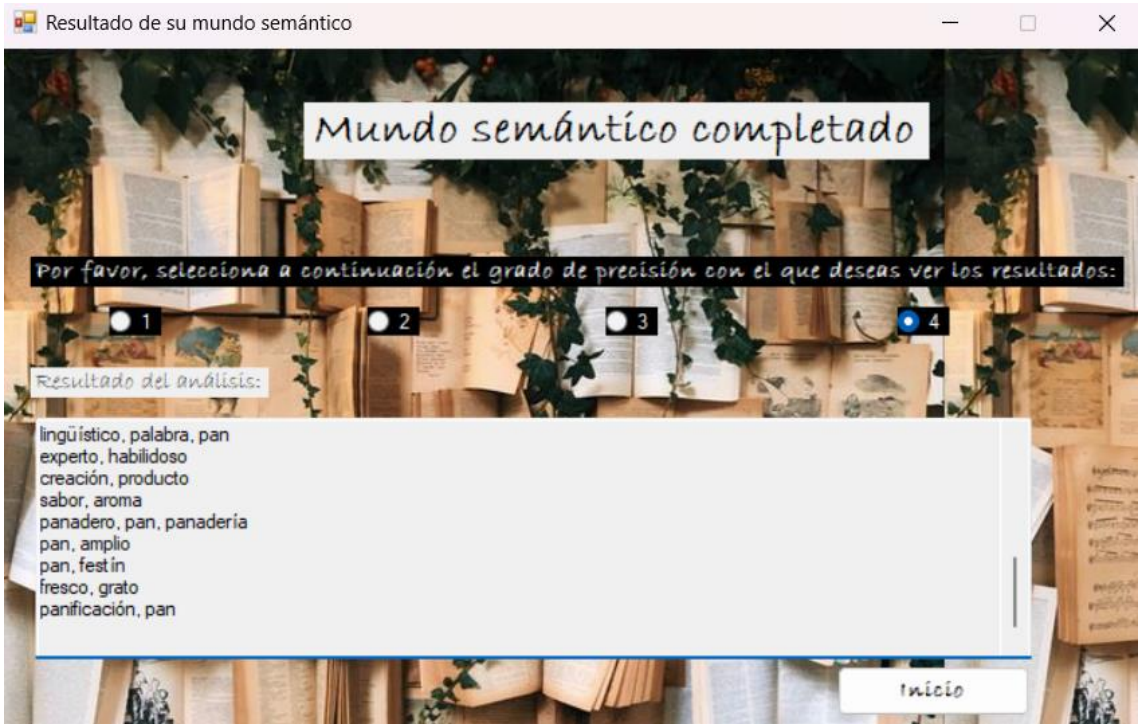

Figura 43. Resto de resultados botón 4

Se puede verificar que se obtienen muchísimas más relaciones de palabras e, incluso, se tiene una scrollbar en caso de que no se puedan visualizar todas porque son muchas. Igualmente, como en la otra prueba, los resultados se guardan en un fichero.

Figura 44. Resultados escritos en el fichero

4.2 Prueba 2

4.2.1 Prueba 2.1

A continuación, se analizará mediante un análisis de campos semánticos puro el siguiente texto para tener otro ejemplo de que el algoritmo funciona correctamente: “Los ángeles son seres celestiales que se han convertido en un símbolo de pureza y bondad en muchas culturas y religiones. Se dice que estas criaturas aladas son mensajeros divinos, guardianes y protectores de la humanidad. Sus figuras angelicales han inspirado a artistas a lo largo de los siglos y su presencia se ha sentido en todo, desde música hasta literatura. Muchas personas tienen fe en que los ángeles están a su lado en momentos de necesidad, trayendo paz y consuelo. La creencia en los ángeles puede brindar un sentido de esperanza y seguridad a aquellos que buscan guía espiritual y apoyo.”.

Este texto es más largo que el anterior, y, en este caso, los resultados son muy buenos. Se mostrarán los resultados obtenidos escritos en el fichero para no ocupar muchas imágenes en la memoria.

Figura 45. Resultados escritos en el fichero

```
Lista #11: ser, ángel
Lista #12: ser, música
Lista #13: ser, guardián
Lista #14: paz, bondad
Lista #15: guía, sentido
Lista #16: música, figura

Resultados botón 4:
Lista #3: ángel, persona, ser
Lista #4: creencia, persona, ser
Lista #10: esperanza, persona, ser
Lista #11: figura, persona, seguridad, ser
Lista #12: guía, persona, ser
Lista #13: apoyo, persona, ser
Lista #16: persona, siglo
Lista #17: persona, presencia, ser
Lista #18: artista, figura, música, persona, símbolo
Lista #19: literatura, persona, ser
Lista #22: persona, pureza, ser
Lista #23: bondad, figura, persona, ser
Lista #24: cultura, figura, persona, ser
Lista #25: momento, persona, religión, ser
Lista #26: criatura, persona, ser
Lista #27: persona, protector, ser
Lista #28: humanidad, persona, ser
Lista #29: bondad, humanidad
Lista #30: guardián, protector
Lista #31: creencia, presencia, religión
Lista #32: fe, creencia, seguridad, esperanza
Lista #33: consuelo, apoyo
Lista #34: ángel, mensajero, religión
Lista #35: presencia, figura
Lista #36: figura, creencia
Lista #37: ser, música
Lista #38: ser, guardián
Lista #39: paz, bondad
Lista #40: guía, sentido
```

Figura 46. Resto de resultados escritos en el fichero

Con estos resultados se pueden observar las distintas relaciones que forman campos semánticos encontradas. Se puede comprobar que entre más recursos de búsquedas se utilicen, más campos semánticos se encuentran siendo el cuarto botón más abstracto. Se recuerda que por defecto el parámetro para encontrar similitudes entre palabras en el diccionario de definiciones está a 15.

4.2.1 Prueba 2.2

Ahora se realizará un análisis de red de palabras para el mismo texto y ver sus resultados. Se recuerda que para este análisis se incluyen los sustantivos, adjetivos y verbos. Los campos semánticos encontrados son los siguientes:

```
Resultados botón 1:

Lista #1: bondad, humanidad

Lista #2: religión, fe, creencia, presencia

Lista #3: guardián, protector

Lista #4: criatura, persona

Lista #5: esperanza, persona

Lista #6: apoyo, consuelo

Lista #7: ser, criatura

Lista #8: ángel, mensajero

Lista #9: celestial, divino

Lista #10: esperanza, seguridad, creencia

Lista #11: figura, símbolo

Lista #12: persona, ser

Lista #13: presencia, figura

Lista #14: seguridad, fe


Resultados botón 2:

Lista #1: bondad, humanidad

Lista #2: religión, fe, creencia, presencia

Lista #3: guardián, protector

Lista #4: criatura, persona

Lista #5: esperanza, persona

Lista #6: apoyo, consuelo

Lista #7: ser, criatura, traer

Lista #8: ángel, mensajero

Lista #9: celestial, divino

Lista #10: esperanza, seguridad, creencia

Lista #11: figura, símbolo

Lista #12: persona, ser

Lista #13: presencia, figura

Lista #14: seguridad, fe

Lista #15: ser, angelical, espiritual

Lista #16: ser, ángel

Lista #17: ser, música

Lista #18: ser, guardián

Lista #19: figura, creencia
```

Resultados botón 3:

Lista #1: religión, fe, creencia, presencia, esperanza

Lista #2: guardián, protector

Lista #3: criatura, persona

Lista #4: esperanza, persona

Lista #5: apoyo, consuelo

Lista #6: ser, criatura, traer

Lista #7: ángel, mensajero, celestial, alado, angelical, religión

Lista #8: celestial, divino

Lista #9: esperanza, seguridad, creencia

Lista #10: figura, símbolo

Lista #11: persona, ser

Lista #12: presencia, figura

Lista #13: seguridad, fe

Lista #14: ser, angelical, espiritual

Lista #15: ser, ángel

Lista #16: ser, música

Lista #17: ser, guardián

Lista #18: figura, creencia

Lista #19: guía, sentido

Lista #20: paz, bondad

Lista #21: música, largo, figura

Resultados botón 4:

Lista #1: buscar, creencia, inspirar, persona, sentido, sentir, ser

Lista #2: ángel, figura, persona, ser

Lista #3: brindar, persona, ser, traer, criatura

Lista #5: buscar, esperanza, inspirar, persona, sentir, ser, traer

Lista #8: buscar, figura, guía, persona, sentir, ser, símbolo, traer

Lista #9: buscar, espiritual, inspirar, persona, sentir, ser, angelical

Lista #13: convertir, momento, persona, ser

Lista #15: buscar, largo, persona, pureza, sentir, ser, traer

Lista #24: criatura, persona, sentido, ser

Lista #27: consuelo, persona, ser

Lista #30: divino, humanidad, inspirar, literatura, momento, persona, sentido, sentir, ser

Lista #31: persona, protector, ser

Lista #33: buscar, figura, inspirar, persona, sentir, ser, traer

Lista #34: artista, creencia, figura, inspirar, persona, presencia, sentir, ser, símbolo

Lista #37: persona, presencia, sentido, ser, traer

Lista #39: artista, figura, humanidad, literatura, música, persona, sentido, sentir, ser, símbolo

Lista #40: literatura, persona, ser, traer

Lista #42: guardián, protector

```
Lista #43: apoyo, consuelo

Lista #44: celestial, divino

Lista #45: ser, guardián

Lista #46: guía, sentido

Lista #47: paz, bondad
```

4.3 Razonamiento de los resultados

Como se dijo anteriormente, se puede comprobar que entre más recursos de búsquedas se utilicen, más campos semánticos se encuentran. El cuarto botón es el más abstracto. Por defecto el parámetro para encontrar similitudes entre palabras en el diccionario de definiciones está a 15 pero siempre se puede cambiar.

Gracias a la utilización de todos los recursos que se tenían disponibles se ha sido capaz de utilizarlos de la mejor manera posible para encontrar grupos semánticos. Los resultados en su mayoría son coherentes y lógicos. Es posible que en ocasiones parezca que una palabra parezca que no pertenece a cierto campo semántico, pero eso depende de la abstracción del recurso utilizado y de la manera en la que este algoritmo trabaja.

Por lo general, el algoritmo da relaciones de campos semánticos afines, coherentes y entendibles. Siempre este algoritmo se podrá mejorar utilizando más recursos sobre este o refinando ciertos parámetros para hacer una búsqueda de relaciones más precisa. Sin embargo, de la manera en la que se ha hecho este algoritmo se obtienen resultados muy exitosos y valiosos.

Asimismo, al hacer un análisis de cualquier tipo el usuario sin leer el fichero podrá conocer la temática de la que trata su fichero, lo cual es un tiempo que ahorra si tiene prisa en saber de qué va un texto o qué relaciones de palabras se puede encontrar en él.

5. Conclusiones

Con este proyecto se han aprendido diversos conocimientos que no se habían dado en el plan de estudios educativo. Se han asimilado nuevas tecnologías, nuevos lenguajes y nuevas maneras de programar, en lo que en el ámbito de informática se refiere. De la misma manera, se han repasado conceptos de lengua española que son muy importantes en el día a día para la creación de inteligencias artificiales que utilizan procesamiento de lenguaje natural. Se han puesto en práctica los contenidos dados en múltiples asignaturas relacionadas con el software a la hora de estructurar el código haciéndolo escalable, legible y modular. Se ha repasado la programación multihilos que se dio en asignaturas relacionadas con sistemas operativos lo que ha supuesto una mayor eficiencia en el rendimiento del programa mejorando su velocidad de ejecución.

Sin ninguna duda, ha supuesto un reto ya que no se tenía ningún proyecto anterior de referencia y se ha tenido que hacer una exhausta investigación al principio lo que ha generado a que se hayan adquiridos nuevas maneras de razonar, organizar, estructurar ideas y buscar posibles soluciones a los problemas que iban surgiendo.

También, se han conocido muchos recursos que han sido útiles para el desarrollo de este trabajo. Uno de ellos se ejecuta en lenguaje Python, wordnet, por lo que se ha aprendido a integrar código Python en lenguaje C# usando el entorno Visual Studio preparado para este último lenguaje. El desarrollo de este proyecto ha sido muy educativo porque ha brindado muchos nuevos conocimientos y se ha realizado una larga investigación de cómo resolver este problema de encontrar grupos semánticos buscando recursos, creándolos y/o utilizando recursos ya creados previamente.

6. Posibles usos del algoritmo

Uno de ellos, es que este software se puede utilizar como herramienta educativa en los centros escolares cuando se esté dando el temario de campos semánticos en la asignatura de lengua española. Por ejemplo, si el alumno/a está estudiando en su casa y quiere comprobar que ha identificado correctamente los campos semánticos presentes en un texto, puede utilizar esta herramienta como método de corrección para comprobar que sus respuestas estaban bien o complementarlas.

Asimismo, este algoritmo puede utilizarse para crear un modelo que se entrene para usar redes neuronales. Es decir, puede servir para programar este algoritmo, pero utilizando redes neuronales.

Además, puede utilizarse como herramienta de investigación policial. Se puede crear un subconjunto de palabras “peligrosas” y si se imprimen alguna de ellas significa que algún acto malo se está hablando en una página web, chat y/o otros portales de internet.

También, sirve como herramienta para traductores e intérpretes para detectar campos semánticos y analizarlos. Igualmente, para estudiantes de lengua española que quieran comprobar los resultados.

7. Líneas futuras

Se pueden añadir nuevos recursos que garanticen mejores resultados como "*word embeddings*" en Python para ser más precisos en las búsquedas de grupos semánticos. Asimismo, se pueden buscar más recursos existentes en línea.

De la misma forma, se puede crear a partir de este algoritmo un modelo que sirva de entrenamiento para redes neuronales.

Además, en este algoritmo se llama externamente a los scripts de Python. Se puede buscar la manera de ejecutar estos scripts en el propio entorno.

8. Referencias

[1] "Real Academia Española", "Semántica léxica", 2022. [En línea]. Disponible en: https://dle.rae.es/sem%C3%A1ntico#XVRDns5

[2] "Real Academia Española", "Sinonimia", 2022. [En línea]. Disponible en: https://dle.rae.es/sinonimia?m=form

[3] "Real Academia Española", "Antónimo", 2022. [En línea]. Disponible en: https://dle.rae.es/ant%C3%B3nimo?m=form

[4] "Stanford University", "The Stanford NLP Group". [En línea]. Disponible en: https://nlp.stanford.edu/

[5] "Princeton University", "WordNet". [En línea]. Disponible en: https://wordnet.princeton.edu/

[6] "Real Academia Española", "Polisemia", 2022. [En línea]. Disponible en: https://dle.rae.es/polisemia?m=form

[7] "Universidad Autónoma de Barcelona", "WordNet", 2022. [En línea]. Disponible en: https://www.researchgate.net/publication/265535119_The_Spanish_Version_of_WordNet_30

[8] "Python", "NLTK". [En línea]. Disponible en: https://www.nltk.org/

[9] "Python", "Python.NET". [En línea]. Disponible en: http://pythonnet.github.io/

[10] "Python", "IronPythom". [En línea]. Disponible en: https://ironpython.net/

[11] "Microsoft", "Process Class". [En línea]. Disponible en: https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.process?view=net-7.0

[12] "Medium", "NLP" [En línea]. Disponible en: https://theolivenbaum.medium.com/nlp-in-c-made-easy-with-spacy-catalyst-acc93e005f3d

[13] "FreeLing". [En línea]. Disponible en: https://nlp.lsi.upc.edu/freeling/node/1

[14] "Microsoft", "Text Translation". [En línea]. Disponible en: https://www.microsoft.com/en-us/translator/business/translator-api/

[15] “Desconocido”. [En línea]. Disponible en: https://github.com/thewickedaxe/WordNetLib/

[16] “Kinsta”. [En línea]. Disponible en: https://kinsta.com/es/base-de-conocimiento/que-es-web-scraping/´

[17] “Microsoft”, “Desarrollo de aplicaciones .NET” [En línea]. Disponible en: https://visualstudio.microsoft.com/es/vs/features/net-development/

[18] “Microsoft”, “What Is Windows Communication Foundation” [En línea]. Disponible en: https://learn.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf

[19] “Microsoft”, “Using threads and threading” [En línea]. Disponible en: https://learn.microsoft.com/en-us/dotnet/standard/threading/using-threads-and-threading

[20] “Microsoft”, “Task.WaitAll Método” [En línea]. Disponible en: https://learn.microsoft.com/es-es/dotnet/api/system.threading.tasks.task.waitall?view=net-7.0

[21] “Microsoft”, “Task Clase” [En línea]. Disponible en: https://learn.microsoft.com/es-es/dotnet/api/system.threading.tasks.task?view=net-8.0

[22] “Microsoft”, “Procedimiento Escribir un bucle Parallel.ForEach sencillo” [En línea]. Disponible en: https://learn.microsoft.com/es-es/dotnet/standard/parallel-programming/how-to-write-a-simple-parallel-foreach-loop

[23] “Desarrollo web”, “Qué es MVC”. [En línea]. Disponible en: https://desarrolloweb.com/articulos/que-es-mvc.html

[24] “Geeks for geeks”, “MVC Design Pattern”. [En línea]. Disponible en: https://www.geeksforgeeks.org/mvc-design-pattern/

[25] “Core NLP”. [En línea]. Disponible en: https://stanfordnlp.github.io/CoreNLP/

[26] “Python”, “Natural Language Processing With Python's NLTK Package”. [En línea]. Disponible en: https://realpython.com/nltk-nlp-python/

[27] “Python”, “Natural Language Toolkit”. [En línea]. Disponible en: https://sites.google.com/view/programacion-en-python/home/3-natural-language-toolkit?pli=1

[28] “Microsoft”, “Usar Visual C# para leer y escribir en un archivo de texto” [En línea]. Disponible en: “Microsoft”, “Task Clase” [En línea]. Disponible en: https://learn.microsoft.com/es-es/troubleshoot/developer/visualstudio/csharp/language-compilers/read-write-text-file

[29] “Microsoft”, “Usar Visual C# para leer y escribir en un archivo de texto” [En línea]. Disponible en: “Microsoft”, “File.Create Método” [En línea]. Disponible en: https://learn.microsoft.com/es-es/dotnet/api/system.io.file.create?view=net-8.0

[30] “IATEXT”, “Familias TIP”. [En línea]. Disponible en: https://tip.iatext.ulpgc.es/familia-de-palabras/