uc3m | Universidad Carlos III de Madrid

Doble Grado en Ingeniería Informática y Administración de Empresas
2019-2020

Trabajo Fin de Grado

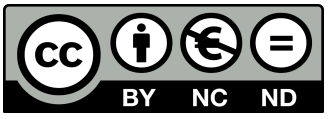
"Resolución de anáforas en función de su contexto semántico"

Santiago Fuentes Cruz

Tutor: Valentín Moreno Pelayo

FUENTES CRUZ SANTIAGO - 06026504K

Firmado digitalmente por FUENTES CRUZ SANTIAGO - 06026504K
Fecha: 2020.09.03 09:12:01 +02'00'



RESUMEN

En el campo de la Inteligencia Artificial, el Procesamiento del Lenguaje Natural es una rama que busca conseguir que los computadores interactúen con las personas usando lenguaje natural. Para lograrlo, uno de los problemas que lleva décadas investigándose es cómo resolver determinados fenómenos lingüísticos como las elipsis, anáforas, catáforas y otras estructuras lingüísticas que generan ambigüedad.

Este trabajo consiste en una investigación previa sobre los enfoques usados hasta nuestros días para resolver el fenómeno de la anáfora pronominal. El trabajo continúa con la construcción de dos soluciones. La primera consiste en una solución computacional que se integra con una herramienta comercial de procesamiento de lenguaje natural. La segunda se basa en la creación de un modelo y el uso de una herramienta basados en Inteligencia Artificial. Ambas soluciones se han realizado orientándose hacia el español, aunque son perfectamente aplicables para otros idiomas.

Después se analizan y comparan los resultados de ambas soluciones, que han mostrado ser bastante eficaces, obteniendo resultados de un 83% y un 97% de éxito respectivamente.

Por último se muestra la planificación que se ha llevado durante el trabajo, y las herramientas usadas.

AGRADECIMIENTOS

Muchas personas me han ayudado a ser quien soy hoy. Enumerarlas sin dejarme a nadie sería arduo para mí e insufrible para el lector. Por tanto, de forma escueta, me gustaría agradecer:

- A toda mi familia, y en especial a mis padres.
- A Carrillo y a Cabanas, sin vosotros la universidad no habría sido lo mismo. A David y a Miguel, que crecimos juntos y somos como hermanos. A Rubén, Mateo y Harmo, por los años disfrutando en el Mundo de los Doce. Y a todos mis amigos.
- A mi tutor Valentín, por su acertada guía, buena disposición y profesionalidad. Al resto de profesores de la UC3M, por compartir sus valores y conocimientos.

¡Mente fría!. Pensemos.
Máximo Angulo

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1 Motivación

El Procesamiento del Lenguaje Natural (en adelante PLN), es una parte fundamental de las Ciencias de la Computación, concretamente de la Inteligencia Artificial (en adelante IA). Un futuro en el que las máquinas interactúen con los humanos a través del lenguaje natural queda todavía algo lejos, debido a que los lenguajes tienen fenómenos lingüísticos que crean ambigüedad, siendo esto una barrera complicada de superar para las máquinas.

Uno de estos fenómenos es la anáfora, enmarcada dentro de la ambigüedad referencial. Esta relación que se establece entre una expresión lingüística y aquello a lo que alude, es fácilmente resuelto por los humanos, mientras que para las máquinas es una tarea ardua.

La importancia que tienen la IA y el PLN en nuestras vidas es cada día mayor, tal y como se discute en el apartado 6 "Marco socieconómico", por lo cual tenemos que continuar investigando y aprendiendo cómo mejorar en la desambiguación de textos.

Una de las tareas pendientes que tenemos es investigar y desarrollar algoritmos para resolver anáforas en español, así como construir *corpus* en español con anáforas resueltas. En otros idiomas como el inglés, en pocos minutos se encuentran investigaciones de todo tipo y herramientas para resolver anáforas, mientras que el español destaca por la ausencia de soluciones, especialmente de IA que resuelvan esta problemática.

Todo esto ha motivado el desarrollo del presente trabajo.

1.2 Hipótesis

Usando tanto algoritmos computacionales como IA, podemos resolver anáforas pronominales, especialmente si clusterizamos los sustantivos por categorías.

1.3 Objeto

Proponer un algoritmo computacional para resolver anáforas pronominales, así como un modelo para resolverlas usando IA. Ambas tareas hacen hincapié en la necesidad de contar con un *corpus* anotado de anáforas así como de una clusterización extensa.

La principal aplicación será para desambiguar textos de cualquier tipo. Por poner algunos ejemplos:

- Puede ser útil para bufetes de abogados que quieran analizar grandes cantidades de sentencias y predecir si sus clientes ganarán o perderán un juicio.

- A la hora de elaborar la elicitación de requisitos de un proyecto software, para desambiguar los mismos y facilitar de esta manera su comprensibilidad.

1.4 Objetivos

Esta propuesta plantea resolver las anáforas pronominales de un texto dado. Para ello se definen estos objetivos:

1. Obtener los candidatos a resolver la anáfora que genera un pronombre.
2. Estimar en función de parámetros morfológicos y sintácticos el mejor candidato. Estos parámetros son la categoría gramatical, el género, número así como la distancia o el contexto semántico.

1.5 Metodología

La metodología se compone de las siguientes fases:

1. Estudio de las anáforas desde un punto de vista gramatical. Tipos.
2. Estudio de los diferentes algoritmos computacionales existentes para resolver anáforas.
3. Construcción de una base de términos muy completa en español y clusterización de sustantivos.
4. Diseño e implementación de una algoritmia computacional para estimar el mejor candidato que resuelva la anáfora.
5. Diseño e implementación de un corpus anotado de anáforas resueltas.
6. Entrenamiento de una IA para resolver anáforas.
7. Análisis de los resultados, tanto de la algoritmia computacional como del enfoque con IA.

2. ESTADO DEL ARTE

2.1 Procesamiento de Lenguaje Natural

Los lenguajes naturales son aquellos que usamos los seres humanos para comunicarnos, de forma escrita o hablada (Kumar, 2010).

El procesamiento de lenguaje natural es un conjunto de técnicas de las ciencias de la computación y de la lingüística. Permiten a los computadores analizar y representar el lenguaje natural. De esta manera pueden interpretar e interactuar de un modo similar al que poseen las personas (Liddy, 2001).

2.1.2 Breve historia del Procesamiento de Lenguaje Natural

Década de los 50

A nivel histórico, uno de los primeros que investigó sobre este tema fue Alan Turing. Publicó en 1950 una serie de estudios para medir la inteligencia mostrada por un computador mientras conversa con un humano usando lenguaje natural (test de Turing). Los trabajos de Turing en la década de los 50, condujeron primero al desarrollo del modelo neuronal de McCulloch-Pitts, y posteriormente a los trabajos de Kleene sobre autómatas finitos. Posteriormente Shannon aplicó los modelos de Markov para automatizar el procesamiento de lenguaje natural. Chomsky en 1957 publica "*Estructuras sintácticas"* proponiendo la gramática generativa, un conjunto de reglas gramaticales que permiten combinar símbolos para predecir la estructura de las oraciones.

Década de los 60

En este período se desarrollan multitud de algoritmos de *parsing* así como trabajos con IA. Uno de los logros más llamativos fue la finalización del primer corpus online de inglés americano (Brown *corpus*) entre 1963 y 1964.

Década de los 70

Surgen abundantes algoritmos de lógica de predicados, así como reconocimiento y representación de entidades semánticas.

1993 - Hoy en día

Estandarización de modelos probabilísticos. Algoritmos de etiquetado, resolución de correferencias y reconocimiento avanzado de entidades. Innovaciones como la autocorrección ortográfica y gramatical. Explosión de las aplicaciones web, que han llevado a la nube todos estos avances, algunos muy conocidos como los servicios de traducción online (Google Translate). Todo esto habría sido mucho más difícil sin los avances en la capacidad

de procesamiento que se ha conseguido en los computadores, que cada dos años prácticamente se duplica (Ley de Moore).

2.2 Análisis morfológico de textos con PLN

2.2.1 Conceptos fundamentales

La morfología es una rama de la lingüística que nos permite estudiar las características de las palabras, tales como su categoría gramatical, sus variantes y cómo se forman.

El lexema de una palabra es la parte que le da significado como concepto. También se la conoce como raíz. Este concepto, en inglés, se traduce como *stem*. Cualquier derivación que se agregue después de la raíz se conoce como sufijo.

Por otro lado, a cada palabra o unidad con significado semántico se le llama *token*, y al proceso de dividir las oraciones en estos términos se le conoce como *tokenization*. Usualmente los *tokens* suelen ser palabras individuales, es decir, se encuentran entre espacios en blanco. En algunos idiomas hay excepciones, por ejemplo en inglés, que es habitual encontrar contracciones y palabras compuestas que parecen ser un único término, pero que es necesario dividir correctamente (ejemplo: *aren't* que se compone de *are* y *not*).

Al conjunto de documentos objeto de análisis, estudio, u otras tareas, se le llama *corpus*. Si además ha sido previamente procesado, analizado y enriquecido con etiquetas (por ejemplo morfológicas o sintácticas), se le llama *corpus* anotado.

La representación formal de las reglas, conocimientos, conceptos, restricciones y relaciones que rigen los lenguajes se conocen como ontologías (Gruber, 1993).

2.2.2 Tokenización

Suele ser el primer proceso de un análisis morfológico con PLN. Las oraciones del *corpus* se tokenizan, obteniendo palabras que posteriormente serán analizadas. En este proceso se obtienen de las oraciones las palabras por separado. También se retiran por ejemplo los puntos de las siglas (D.N.I a DNI) o se corrigen las tildes que falten en alguna palabra, o se sustituyen las abreviaturas por su palabra completa (pza. a plaza).

2.2.2 Normalización

Con la normalización aplicamos normas (reglas) a los *tokens* para estandarizar su análisis. Por ejemplo sustituir las palabras en femenino por su equivalente en masculino, los verbos a su forma en infinitivo, y las palabras en plural a singular.

2.2.3 Desambiguación y etiquetado

En esta fase cada una de las palabras son lematizadas para obtener por un lado la raíz y por otro lado el sufijo, si es que tuviera alguno. Este etiquetado se puede realizar tanto a mano como automáticamente con herramientas diseñadas para esta tarea.

Normalmente suele haber más de una etiqueta candidata para cada palabra, y es necesario contar con un proceso de desambiguación basado en reglas (esto es, una base de conocimiento u ontología) que asigne correctamente las etiquetas.

2.2.4 Algoritmia

Una vez que el texto está etiquetado, se puede realizar la actividad que se desee, desde construir motores de búsqueda hasta desambiguadores semánticos, entre otros.

2.3 Soluciones existentes para resolver anáforas

Las soluciones que se han ido proponiendo desde los 50 para resolver las anáforas, se pueden dividir en dos grandes vertientes:

1. Enfoque basado en algoritmia computacional: se elabora un algoritmo que contiene una serie de pasos que permiten resolver la anáfora, debiendo el programador volcar toda la lógica de resolución en el algoritmo. Aquellos casos no contemplados en el mismo, fallarán.

2. Enfoque basado en IA: por ejemplo se elabora un modelo que contiene anáforas resueltas, y se le da a la IA para que entrene. Al contrario que en el caso anterior, no es necesario elaborar ningún algoritmo que describa la resolución del problema, sino que es la IA la que estudia el modelo, elabora sus propias reglas y va aprendiendo cómo resolver el problema.

2.3.1 Enfoque con algoritmia computacional

En este apartado se resumen los principales enfoques computacionales.

El problema de la anáfora no es nada nuevo. Wingrad ya propuso en 1972 el primer procedimiento computacional para localizar antecedentes. Posteriormente Rieger en 1974 desarrolló algoritmos para localizar antecedentes en función de las propiedades conocidas de la entidad que genera la anáfora. Ambos se toparon con un problema importante: ¿qué hacer si hay más de un candidato válido entre los antecedentes?

Para intentar resolverlo, Wilks en 1974 propone una solución parcial para decidir entre esos candidatos, intentando obtener información sobre las oraciones para determinar al mejor antecedente, usando para ello una base de datos semántica.

Algoritmo *naive* de Hobbs

Uno de los trabajos más exitosos usando el enfoque computacional lo lleva a cabo Jerry R. Hobbs. Este profesor del departamento de computación del City University de New York explica en su estudio *Pronoun Resolution* de 1976, que usando un análisis puramente sintáctico se obtienen resultados aceptables. Llamó a su algoritmo *naive* (ingenuo). Esto es debido a que cuando el algoritmo falla en elegir el antecedente correcto, no es consciente de que ha fallado.

Hobbs propuso estructurar el texto en forma de árbol de análisis (*parse tree*). De esta manera el árbol replica la estructura de cada oración: sujeto, verbo y complementos, entre otros. Cada una de las palabras es un nodo terminal u hoja en dicho árbol. Además, Hobbs asume que ciertos grupos de palabras conforman nodos jerárquicamente superiores en el árbol. Concretamente los sintagmas nominales (*noun phrase* o NP), los sintagmas verbales (*verb phrase* o VP), oraciones completas (*sentence* o S) y demás estructuras que propone Chomsky en su teoría lingüística de 1970. Después se aplica el siguiente algoritmo:

1. Empezar desde el NP que contiene al pronombre
2. Ir hacia arriba en el árbol hasta encontrar un NP o un S. Llamar a ese nodo X, y al camino recorrido, P.
3. Recorrer todas las ramas que nazcan por debajo y a la izquierda de X, de izquierda a derecha, y proponer como antecedente cualquier NP que se encuentre.
4. Si el nodo X es la raíz de esa oración, viajar a las oraciones anteriores, analizando primero las más cercanas. Si el nodo X no es la raíz, pasar a 5.
5. Desde X, viajar hacia arriba hasta encontrar un NP o un S. Llamar a ese nodo X, y al camino recorrido, P.
6. Si X es un NP y el camino P hasta X no ha pasado por un NP padre de X, proponer a X como antecedente.
7. Buscar antecedentes abajo y a la izquierda de X.
8. Si X es un S viajar por las ramas de la derecha de X, buscando antecedentes, sin ir nunca por debajo de los NP o S que se encuentren.
9. Ir al paso 4.

Un ejemplo de resolución sería el siguiente. Para la oración: *The castle in Camelot remained the residence of the king until 536 when he moved it to London.* (La residencia del rey se mantuvo en Camelot hasta 536, cuando él la trasladó a Londres.)

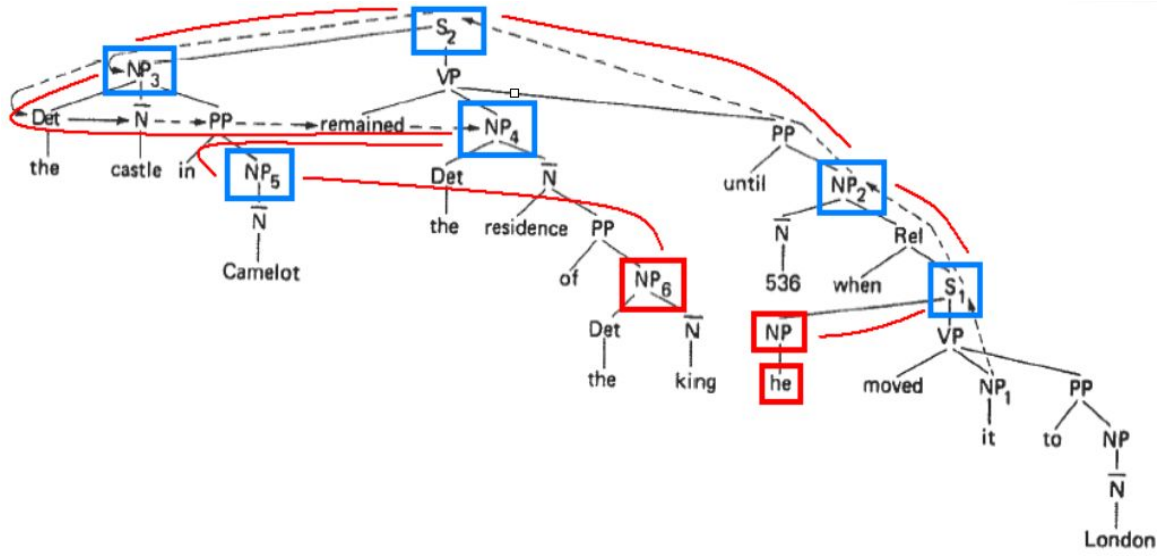


Figura 1: Ejemplo de resolución por algoritmo de Hobbs.
Fuente: Universidad de Montreal

El algoritmo *naive* consiguió un 72,7% de aciertos que si bien no es un éxito rotundo, sí que sentó las bases de posteriores algoritmos.

Algoritmo de Lappin and Leass

Una de las grandes mejoras de este algoritmo de 1994 es proponer que los candidatos a antecedentes se clasifiquen según ciertos parámetros como la distancia, el género, el número, o la concordancia con entidades semánticas, entre otros.

Los pasos a seguir son:

1. Obtener los candidatos a antecedentes de las 4 oraciones anteriores.
2. Eliminar los que no coincidan semánticamente.
3. Eliminar los que no coincidan sintácticamente.
4. Obtener el *salience*, es decir, una puntuación basada en la prominencia o importancia de ese antecedente, en base a parámetros como la distancia hasta el pronombre, o las veces que se repite en el texto.
5. Escoger al candidato con el *salience* más alto.

Este algoritmo obtuvo en torno a un 87% de eficacia resolviendo anáforas.

Algoritmo del enfoque del discurso

Los textos suelen tener una serie de temáticas centrales (usualmente sustantivos) en torno a las cuales gira todo el texto. Si se consigue detectarlos, se puede afirmar que muchos de los pronombres que aparezcan tenderán a resolverse en esos sustantivos. Usando esta hipótesis se elaboraron diferentes algoritmos, pero ninguno obtuvo más de un 80% de éxito.

Otros algoritmos

Respecto a otros algoritmos, me gustaría destacar los estudios de PLN basados en conversaciones. Si bien son más complejos de anotar, usan algoritmos de resolución muy similares a los algoritmos clásicos. El que más me gustó fue la tesis "*Resolución computacional de la anáfora en diálogos*", de Patricio Martínez Barco. Usa un conjunto de listas en las que va clasificando según distancias los antecedentes, y luego tiene una serie de penalizaciones según concuerden o no sintácticamente.

2.3.2 Enfoque con IA

Este enfoque es más complejo, usan diferentes técnicas y quisiera destacar estos estudios, los cuales son relativamente recientes:

- **Redes neuronales recurrentes**: la información de salida es usada como entrada. Son aptos para tratar largas secuencias de datos elemento a elemento. Son costosos en memoria y tiempo de procesado. (Wiseman, Rush y Shieber, 2016).
- **Modelos de pares de menciones**: cambian el enfoque del problema de resolución de correferencia como una tarea de clasificación. Se entrena a un clasificador para que decida si un par de entidades nominales forman una correferencia. Después se construyen cadenas de correferencia completas agrupando estas decisiones por pares. (Wiseman, Rush y Shieber, 2015). Los modelos de pares de menciones no son nuevos, algunos datan del 2001, solo que ahora tenemos más datos, más capacidad de almacenamiento y de cómputo, y todo puede hacerse en la nube.
- **Modelos de menciones de entidades**: se basa en el uso de clusterización sobre representaciones distribuidas de menciones anafóricas (Clark y Manning, 2016)

2.4 Conceptos básicos sobre las anáforas

Una anáfora es un fenómeno lingüístico que se produce cuando un elemento del discurso hace referencia a algo ya mencionado.

Los tipos más conocidos de anáfora son:

- **Anáfora pronominal**: es una de las más abundantes en cualquier idioma. El elemento que genera la correferencia es un pronombre. Estos pronombres pueden ser de muchos tipos: personales, demostrativos, relativos, interrogativos, posesivos, y reflexivos. Dependiendo del pronombre, diferenciamos tipos de anáforas pronominales. Las que más abundan en todos los idiomas son los siguientes:
    - Anáfora pronominal de sujeto: usan pronombres personales (yo, tú, él...) en función de sujeto.
    - Anáfora pronominal de objeto: usan pronombres personales (me, te, se...) en función de complemento directo o indirecto.

    - Anáfora pronominal reflexiva: usan pronombres personales actuando reflexivamente.
    - Anáfora pronominal demostrativa: usan pronombres demostrativos (este, esa, aquel...).
    - Anáfora pronominal posesiva: usan pronombres posesivos (mío, tuyo, suyo...).

- **Anáfora adjetiva**: la anáfora la genera un adjetivo, y se requiere un contexto previo que contenga un sintagma nominal para entenderlo. Por ejemplo: “Tráeme la **chaqueta** que está en el armario. La **verde**.”

- **Anáfora adverbial**: la anáfora la genera un adverbio, y se requiere un contexto previo que contenga un sintagma nominal para entenderlo. Por ejemplo: “El tren sale para **Barcelona** en 10 minutos. Suele llegar **allí** en 2 horas.”

3. SOLUCIÓN PROPUESTA

El primer paso que se tomó previo al diseño de una solución, fue estudiar la anáfora como concepto lingüístico, así como los diferentes tipos de anáfora. Al final me decanté por un estudio de la anáfora pronominal, puesto que es una de las más abundantes en cualquier idioma.

Una vez que hube entendido en detalle cómo se generaban las anáforas pronominales, fue momento de elegir centrarme en un idioma concreto. Si bien en un primer momento pensé en escoger el inglés, acabé eligiendo el español por varios motivos:

1. Me sentía más cómodo trabajando en español, ya que conocía mejor las estructuras gramaticales de las oraciones.
2. Hay una notable falta de *corpus* y algoritmos relativos a anáforas que se dediquen al español, al menos comparándolo con idiomas como el inglés. Un Trabajo de Fin de Grado, en mi opinión, debe aportar algo sobre lo que no se haya investigado previamente o al menos aportar enfoques novedosos o sobre los que no se haya profundizado mucho. Es decir, no ser una mera recopilación de conocimientos existentes. Me motivaba más realizar mis diseños orientándome al español.

3.1 Construcción de la base de datos

Elegido el idioma, restaba encontrar o construir una base de datos con términos en español. Como mencioné en la Introducción, el grupo Reuse me aportó un entorno de trabajo consistente en una aplicación llamada *Knowledge Manager* (en adelante *KM*), capaz de analizar la morfología de las palabras de un texto, clusterizarlas y muchas más tareas. También un código inicial escrito en *C#* que construía una sesión de trabajo conectándose a la base de datos del *KM*. El problema de este entorno, es que me dieron una base de datos RQA (orientada al análisis de requisitos de calidad). Contenía solo unos 8.000 términos de los cuales la inmensa mayoría (unos 6.000) eran verbos, mientras que tan solo 446 eran sustantivos, y 909 adjetivos.

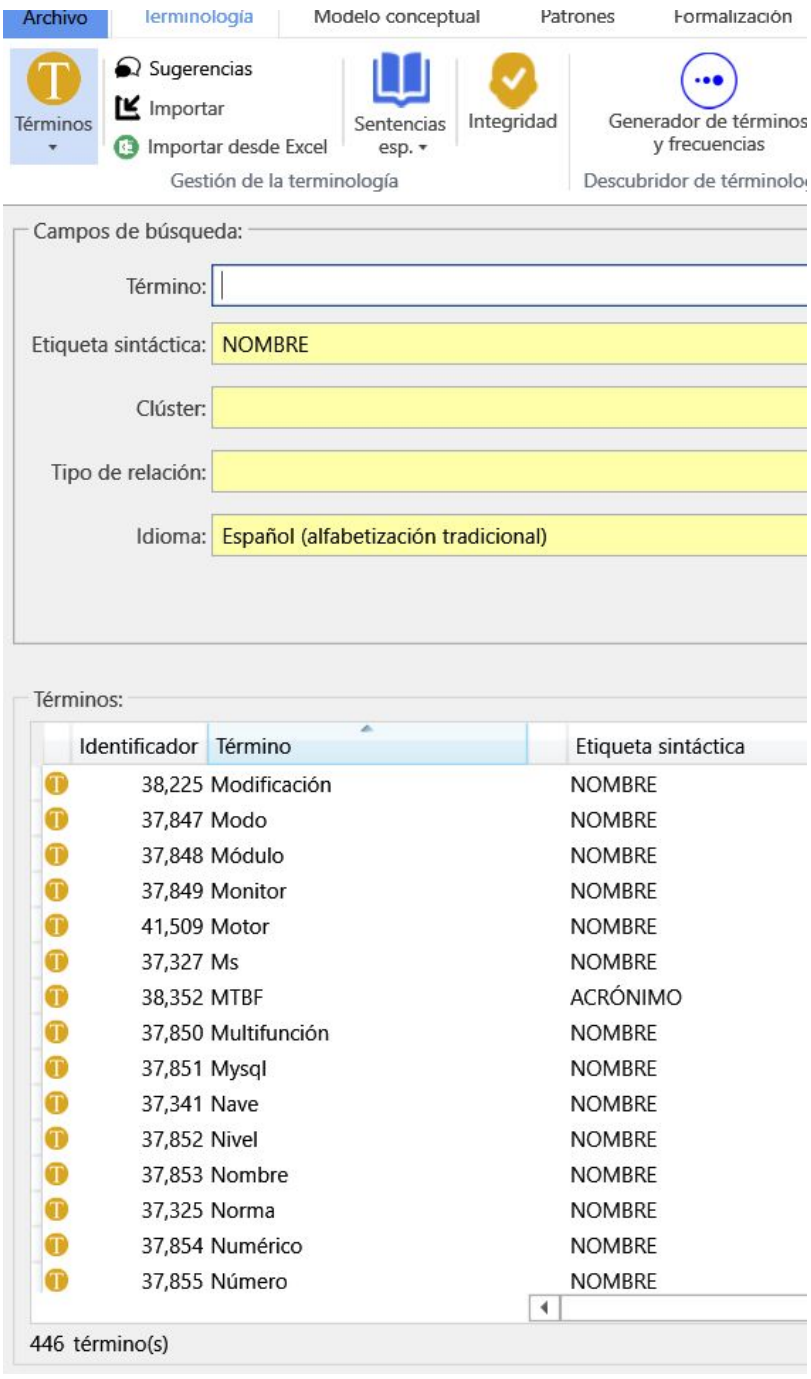


Figura 2. Sustantivos iniciales en la base de datos de KM.

Este primer obstáculo requirió bastante tiempo hasta que encontré una solución. Necesitaba una base de datos lo más completa posible de verbos, adjetivos y sustantivos si quería analizar cualquier oración, por ejemplo de una novela o de un periódico. Sí que es verdad que la herramienta *KM* es capaz de saber con bastante precisión la etiqueta gramatical de una palabra, aunque no la tenga en la base de datos, pero no es capaz de asignar un género y número, que para mí era imprescindible, tal y como expondré en el diseño de la solución.

Un ejemplo de qué es lo que ocurre cuando analizas una oración cuyas palabras no se encuentran en la base de datos lo tenemos aquí:

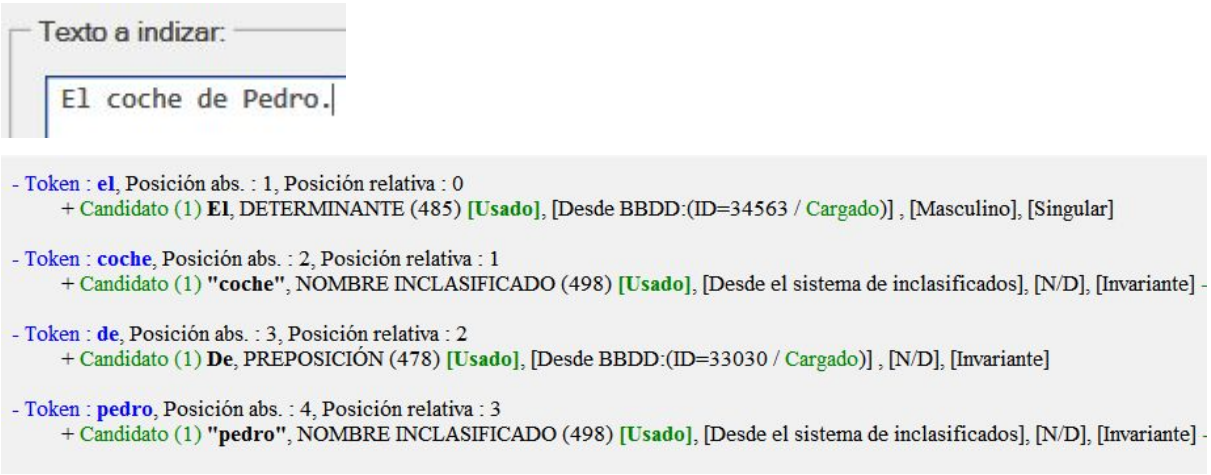


Figura 3. Análisis de palabras que no existen en la base de datos del KM.

Como se puede observar, no se asigna ni género ni número a los nombres inclasificados.

Parece que a primera vista es sencillo encontrar bases de datos con términos en español por Internet, pero no lo es. Para idiomas como el inglés sí. Además tenía que tener en cuenta las limitaciones de formato, puesto que la herramienta *KM* de Reuse solo acepta principalmente diccionarios de entrada en formato .txt (fichero de texto) o .xlsx (Excel). Encontré una base de datos que contenía 144.000 palabras en español, entre sustantivos, verbos, adjetivos etc. El problema es que venían todos mezclados, así que tocaba manualmente separarlos o construir un algoritmo que lo hiciera. Como las palabras no venían etiquetadas, no parecía una opción eficiente y la descarté.

Afortunadamente, mientras buscaba otras bases de datos online con términos en español, me encontré con Wiktionary. Esta web reúne aproximadamente 6,5 millones de palabras en más de 4.000 idiomas. Visitando la subpágina para el español, me encontré que era posible consultar adjetivos, sustantivos, verbos...todo lo que uno se imagine relativo al español.

Me puse manos a la obra y comencé por los sustantivos, los cuales se encontraban aquí.
Una imagen de cómo se divide la información es esta:

Figura 4: Sustantivos en Wiktionary

Como se puede ver, Wiktionary ofrece sustantivos categorizados, además de un índice alfabético desde la A hasta la Z. Pronto me encontré con un inconveniente, y es que no se ofrecen las palabras para descargarlas, así que tenía que obtenerlas de otra manera.

Pensé en copiarlas a mano, pero no aparecían todas en la misma página, solo aparecen 200:

Figura 5. Límite de palabras por página en Wiktionary.

Si necesitas las siguientes 200, había que pulsar en *Next page* para obtenerlas.
No era viable copiar a mano miles de palabras y tener que estar cambiando de página para copiar las siguientes 200, ya que no iba a acabar nunca. Pero Wiktionary me gustó y decidí buscar otro camino. Preguntando en StackOverflow me dieron una solución. Resulta que Wiktionary tenía una *API* fantástica sobre la cual se pueden enviar *queries* para obtener la información que necesites. Cada *query* tiene un límite máximo de 500 términos devueltos, pero programando un método recursivo que lo ejecute hasta que llegue a la Z, pude obtener todos. Un ejemplo de *query* sería:
https://en.wiktionary.org/w/api.php?action=query&list=categorymembers&cmtitle=Category:Spanish_nouns&cmprop=title&cmlimit=max

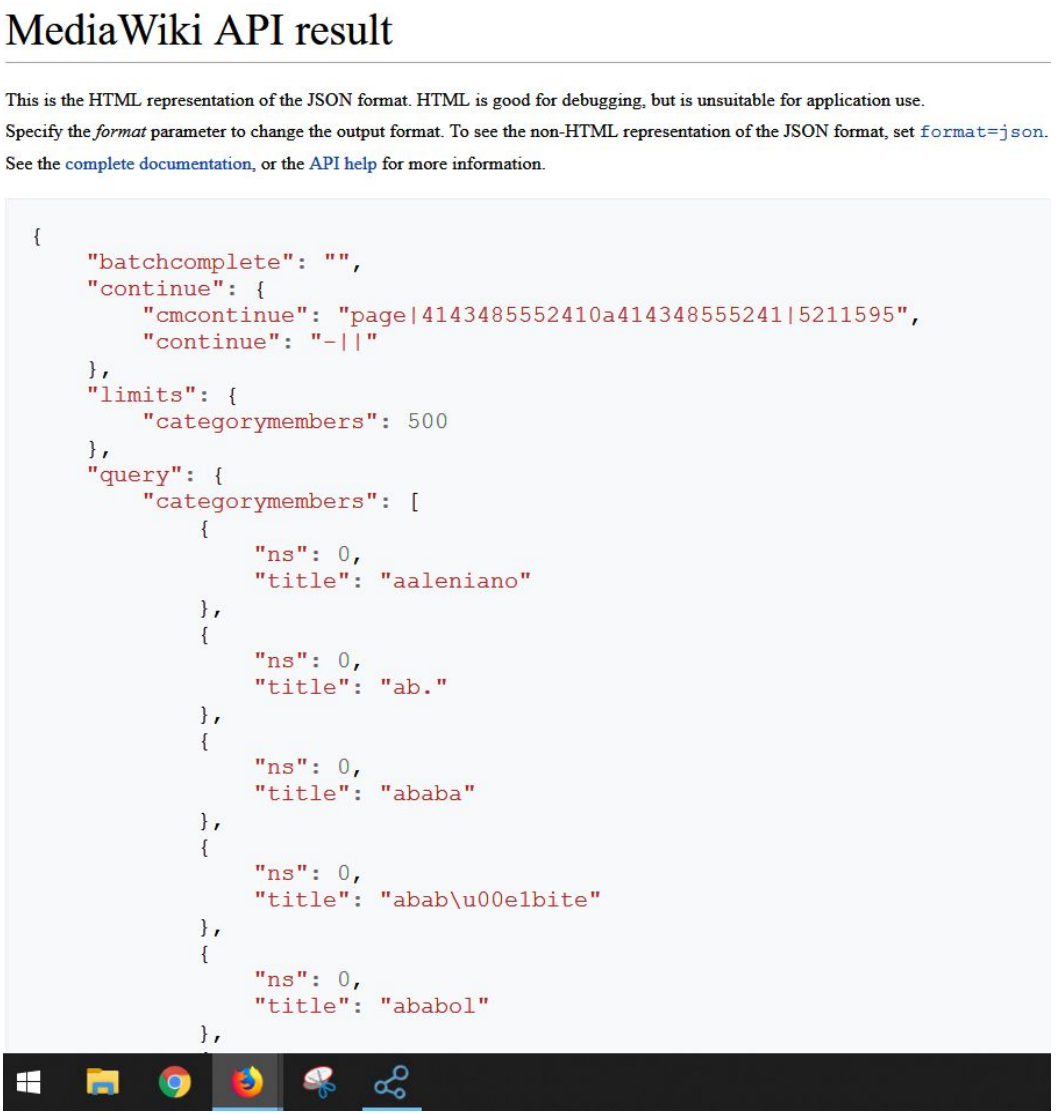


Figura 6: Resultados de query en la API de Wiktionary

Una vez que obtienes los 500 primeros términos, que es lo máximo que la *API* da, puedes obtener los siguientes usando un parámetro extra llamado *cmcontinue*, que podemos verlo en la figura anterior. Es como un puntero para saber en qué palabra te quedaste y poder obtener las siguientes. La *query* usando este parámetro devolvería las siguientes 500 palabras.

https://en.wiktionary.org/w/api.php?action=query&list=categorymembers&cmtitle=Category:Spanish_nouns&cmprop=title&cmlimit=max&cmcontinue=page|41434943414c414455524 10a41434943414c4144555241|4536720

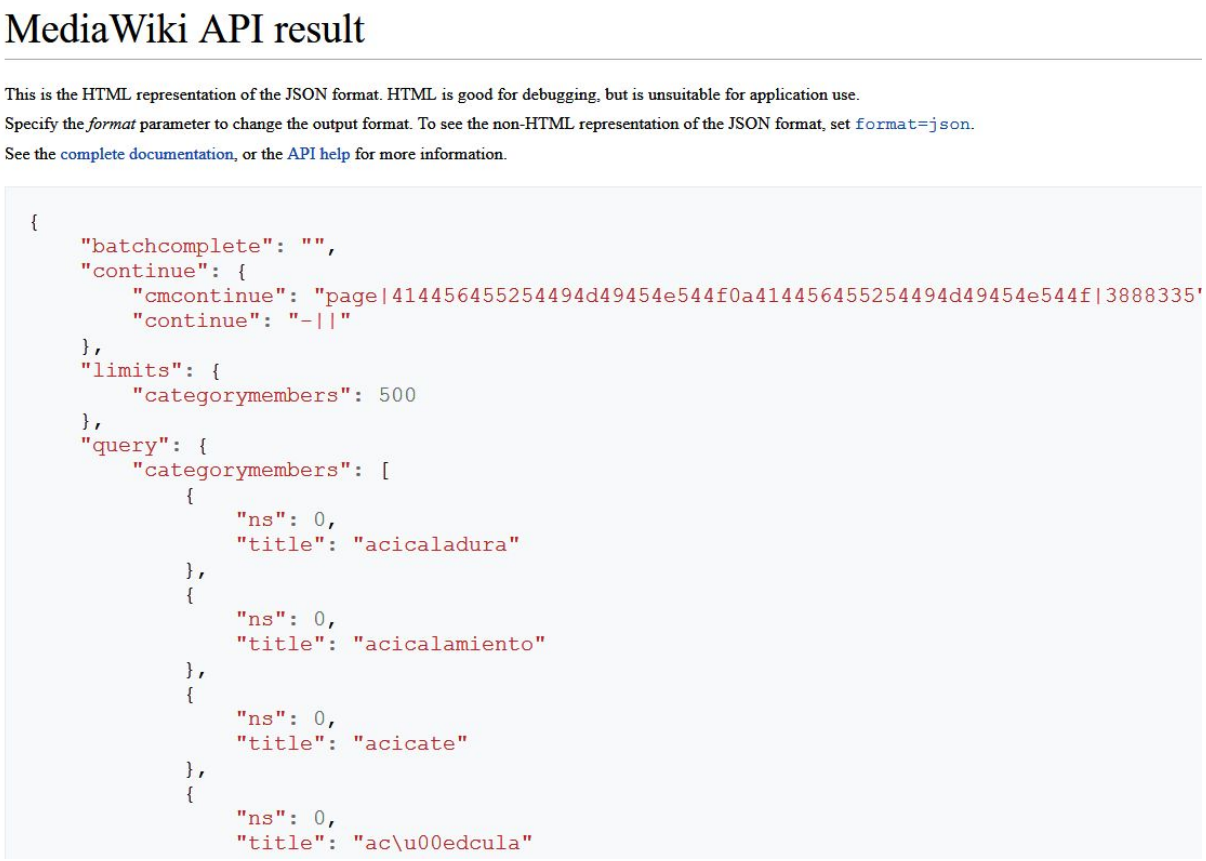


Figura 7: Resultados de query con cmcontinue usando Wiktionary

Programando en el lenguaje que uno prefiera, en pocas líneas te creas un método recursivo que vaya lanzando peticiones HTTP con estas *queries* y almacenando todas las palabras. Después se conservan únicamente los nombres, que vienen en el campo *title* y ya tienes un diccionario hecho a medida. Por si a alguien le es de utilidad este dato, quisiera mencionar que si además de la palabra se necesita su definición u otros parámetros, hay dos formas principales de hacerlo:

1. Descargando un *dump* de los términos en un lenguaje concreto, que Wiktionary va actualizando cada cierto tiempo. Estos descargables están en formatos *XML* y habría que limpiarlos.
2. Realizar un mini *crawler* que se vaya metiendo en los *links* asociados a cada palabra y busque la etiqueta de la definición.

Repitiendo este proceso con la lista de verbos y adjetivos, logré tener una serie de diccionarios bastante completa. Adicionalmente la completé con nombres propios españoles, tanto de mujer como de hombre, así como nombres de comunidades autónomas de España, así como países de todo el mundo, los cuales también encontré en Wiktionary.

Resumiendo: una gran herramienta esta *API*, y que me solucionó el que probablemente fue el mayor obstáculo que encontré en este proyecto.

Una vez que tenía los diccionarios base era hora de probarlos. Los introduje en el *KM* y empecé a experimentar. La sintaxis que se usa para trabajar con *KM* no es complicada, cada palabra se denomina *token* y tiene una serie de propiedades y métodos asociados.

Cuando indexas texto te devuelve una estructura que contiene todas las oraciones, a las que puedes acceder de una en una, así como a sus palabras (*tokens*).

Estos *tokens* tienen decenas de propiedades, de las cuales es preciso concretar algunas antes de seguir:

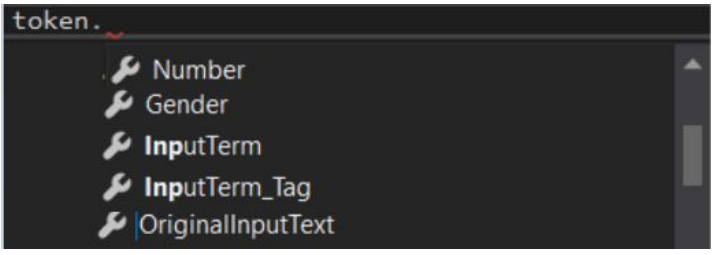


Figura 8: Propiedades importantes de los tokens

Tal y como dice el nombre, *Number* y *Gender* permiten saber el número y género de la palabra. *InputTerm* devuelve el término tokenizado, lematizado y normalizado. *InputTerm_Tag* aporta la etiqueta gramatical que se ha asociado a la palabra, mientras que *OriginalInputText* devuelve la palabra sin lematizar ni normalizar.

En mi caso me encontré con que la herramienta *KM* tenía unas reglas de normalización que hacían que los sustantivos con género femenino cambiaran a su equivalente en masculino, y que los sustantivos en plural cambiaran a singular. Esto fue un problema debido a que *KM* me alteraba el contenido original de los *inputs* que le metía. Por ejemplo:

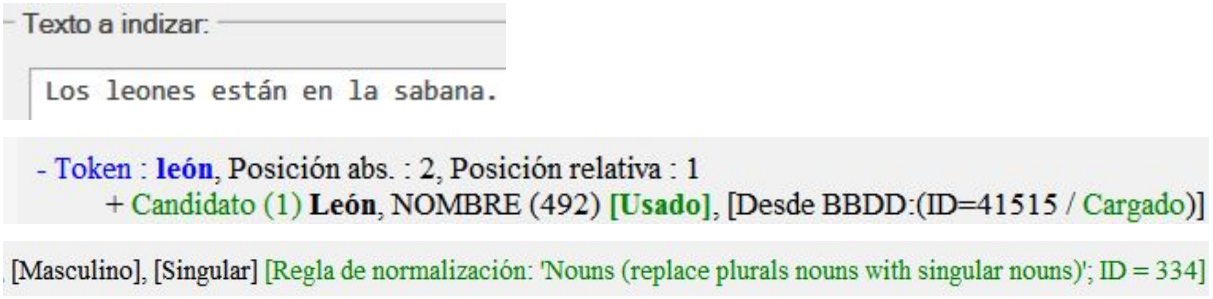


Figura 9: Normalización de plurales en KM.

Y la misma oración pero poniendo leonas en vez de leones:

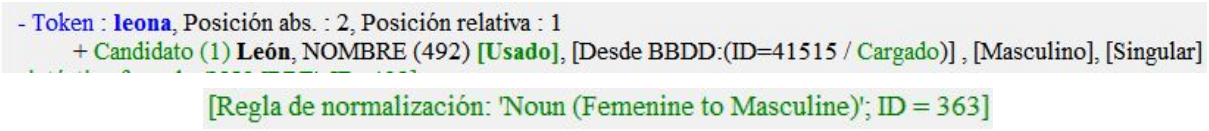


Figura 10: Normalización de femeninos en KM.

Este fenómeno traía varios problemas asociados:

- Al no existir la palabra en plural, o la palabra en femenino en la base de datos, el *InputTerm* de ese token salía nulo. Las propiedades más útiles de los tokens que necesitaba, como el *Gender* y el *Number* solo eran accesibles a través de *InputTerm.Gender* e *InputTerm.Number*.

- El término original, accesible mediante *OriginalInputText* o *InputText*, no tenía propiedades *Gender* ni *Number*, solo *Length*.

Figura 11: Propiedades de InputText y OriginalInputText

Las posibles soluciones eran las siguientes:

1. Crear un *InputTerm* y asignarle el texto de *OriginalInputText*. Cambiarle el género y el número a los opuestos de lo que marcara la regla de normalización. El género y el número se representan mediante valores enteros. Por ejemplo el número puede ser 0 para no definido, 1 para masculino, 2 para femenino y 3 para invariable.

```
InputInfo i = new InputInfo();
i = token;
i.InputTerm.NormalizedName = token.OriginalInputText;
i.InputTerm.AllowChangeGender = true;
i.InputTerm.AllowChangeNumber = true;
i.InputTerm.Gender = 2;
i.InputTerm.Number = 2;
```

Figura 12: Posible solución al problema de la normalización.

Si bien esta solución era válida, no la llegué ni a probar. Esto es debido a que para cada palabra en femenino y/o plural, se iba a crear un *token* nuevo, además de revisar qué regla había sido disparada, y actuar en consecuencia. Si el texto a analizar contuviese 5.000 palabras en femenino, se iban a crear en *runtime* 5.000 nuevos tokens, y 5.000 asignaciones de género. A nivel de velocidad de ejecución y de consumo de memoria, no me parecía una solución eficiente.

2. Desactivar las reglas que producían ese comportamiento. No era viable debido a que las palabras en femenino y en plural no existían en la base de datos, y *KM* les asignaba género y número no definidos.

3. Desactivar las reglas de normalización de sustantivos y meter en la base de datos los sustantivos en plural y en femenino. Esta solución es por la que opté.

    - **Inconvenientes**:
        - Las reglas de normalización existen, entre otras cosas, para evitar meter en la base de datos palabras cuyo lexema es igual y solo varían en género y número. Esto ahorra espacio. Esta solución iba a introducir una carga de datos "repetidos" en la base de datos que no es eficiente. El espacio que ocupa la base de datos aumenta en varias decenas de MB.
    - **Ventajas**:
        - Crear mis propios diccionarios distinguiendo entre masculino y femenino, singular y plural, iba a ahorrar tiempo para futuros proyectos en los que solo se requiera trabajar con una morfología concreta. Dicho de otra manera, se ha realizado pensando tanto en este proyecto como en otros futuros.
        - El tiempo de búsqueda en la base de datos no cambia casi nada, debido a que los términos en masculino-femenino, singular-plural, se almacenan de forma contigua (puesto que alfabéticamente son casi iguales), y el buscador tiene índices.
        - Me evito generar en *runtime* una gran cantidad de *tokens* nuevos, detección de reglas y demás, ahorrando tiempo de ejecución y memoria. Al existir las palabras en la base de datos, se puede acceder directamente a las propiedades *Gender* y *Number* y van a tener los datos correctos, sin necesidad de hacer nada más.

Tomada la decisión, programé un algoritmo que generara a partir de mis sustantivos en masculino, su equivalente en femenino. Y una vez hecho lo anterior, otro algoritmo que creara las palabras en plural. Hay que tener en cuenta que hubo que retirar a mano las excepciones. Por ejemplo palabras que cambian como "toro" a "vaca" o palabras como "día" que acaba en 'a' pero no es femenino.

Por último dividí el .txt en varios separados, para facilitar su inserción en la base de datos, y para ahorrar tiempo a la gente que quiera usar solo alguna letra en concreto en proyectos futuros.

Spanish Feminine Nouns Plural	14/08/2020 13:16	Carpeta de archivos
Spanish Adjetives	30/07/2020 11:02	Carpeta de archivos
Spanish Feminine Nouns	24/06/2020 19:01	Carpeta de archivos
Spanish Locations	24/06/2020 19:01	Carpeta de archivos
Spanish Masculine Nouns	25/06/2020 22:56	Carpeta de archivos
Spanish Masculine Nouns Plural	14/08/2020 13:16	Carpeta de archivos
Spanish Proper Nouns	24/06/2020 19:01	Carpeta de archivos

Diccionarios Español › Spanish Masculine Nouns

Nombre	Fecha de modificación	Tipo	Tamaño
Nouns Spanish A Masculine.txt	24/06/2020 19:27	Documento de tex...	40 KB
Nouns Spanish B Masculine.txt	21/06/2020 12:43	Documento de tex...	15 KB
Nouns Spanish C Masculine.txt	24/06/2020 19:54	Documento de tex...	48 KB
Nouns Spanish D Masculine.txt	21/06/2020 12:54	Documento de tex...	21 KB
Nouns Spanish E Masculine.txt	21/06/2020 12:55	Documento de tex...	27 KB
Nouns Spanish F Masculine.txt	21/06/2020 12:56	Documento de tex...	12 KB
Nouns Spanish G Masculine.txt	21/06/2020 12:57	Documento de tex...	11 KB
Nouns Spanish H Masculine.txt	24/06/2020 20:18	Documento de tex...	11 KB
Nouns Spanish I Masculine.txt	21/06/2020 13:00	Documento de tex...	11 KB
Nouns Spanish J Masculine.txt	21/06/2020 13:01	Documento de tex...	3 KB
Nouns Spanish K Masculine.txt	21/06/2020 13:02	Documento de tex...	2 KB
Nouns Spanish L Masculine.txt	21/06/2020 13:02	Documento de tex...	9 KB
Nouns Spanish M Masculine.txt	21/06/2020 13:03	Documento de tex...	25 KB
Nouns Spanish N Masculine.txt	21/06/2020 13:06	Documento de tex...	8 KB
Nouns Spanish Ñ Masculine.txt	21/06/2020 13:07	Documento de tex...	1 KB
Nouns Spanish O Masculine.txt	21/06/2020 13:08	Documento de tex...	7 KB
Nouns Spanish P Masculine.txt	21/06/2020 13:09	Documento de tex...	35 KB
Nouns Spanish Q Masculine.txt	21/06/2020 13:10	Documento de tex...	2 KB
Nouns Spanish R Masculine.txt	21/06/2020 13:12	Documento de tex...	19 KB
Nouns Spanish S Masculine.txt	21/06/2020 13:13	Documento de tex...	23 KB
Nouns Spanish T Masculine.txt	21/06/2020 13:14	Documento de tex...	20 KB
Nouns Spanish U Masculine.txt	21/06/2020 13:14	Documento de tex...	2 KB
Nouns Spanish V Masculine.txt	21/06/2020 13:15	Documento de tex...	9 KB
Nouns Spanish W Masculine.txt	21/06/2020 13:15	Documento de tex...	1 KB
Nouns Spanish X Masculine.txt	21/06/2020 13:16	Documento de tex...	1 KB
Nouns Spanish Y Masculine.txt	21/06/2020 13:17	Documento de tex...	1 KB
Nouns Spanish Z Masculine.txt	21/06/2020 13:18	Documento de tex...	2 KB

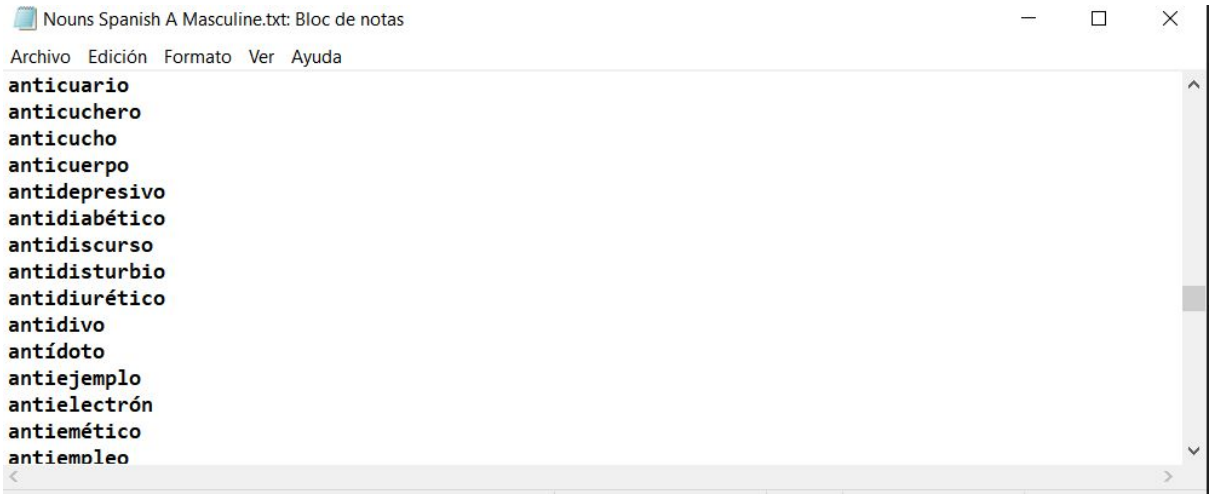


Figura 13: Diccionarios creados. Sustantivos masculinos en singular. Contenido de la letra A.

De esta manera acabé con un total de 92.333 sustantivos en la base de datos.

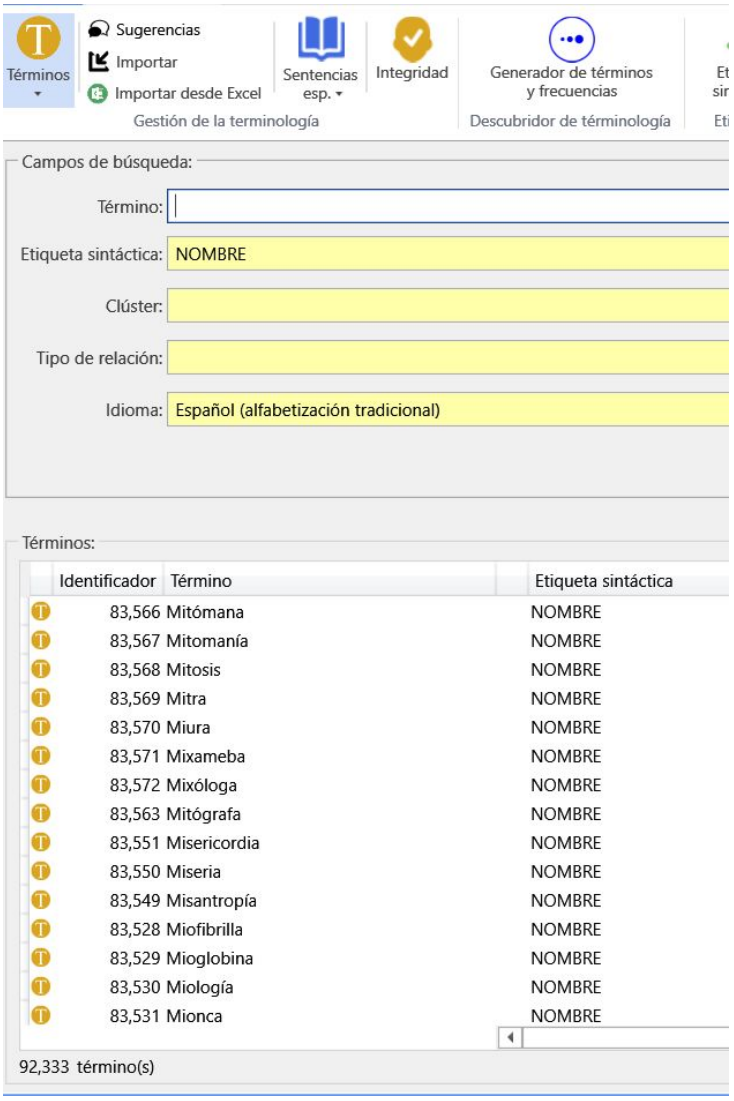


Figura 14: Sustantivos.

Y realizando un proceso similar, aumenté el número de adjetivos hasta un total de 18.152 términos.

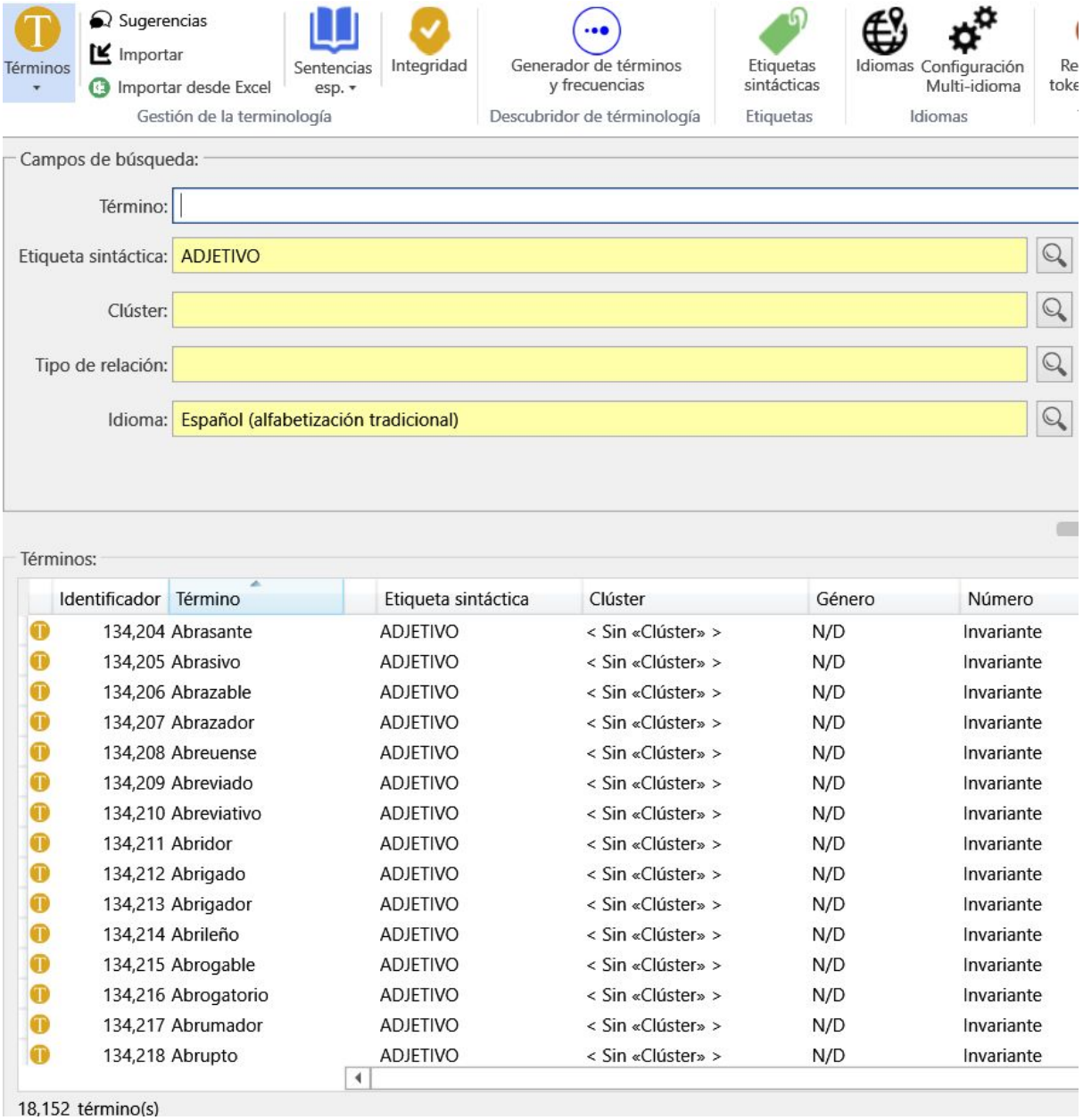
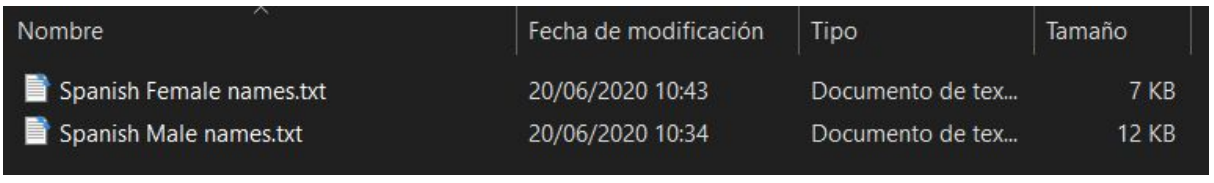


Figura 15: Adjetivos.

Era momento de clasificar en la medida de lo posible los sustantivos obtenidos. Todos aquellos nombres que fueran nombres de persona, los etiqueté como nombres propios de persona, obteniendo un total de 2.165 términos. Afortunadamente no hubo que hacerlo a mano, sino que obtuve los nombres de Wiktionary, como ya comenté.



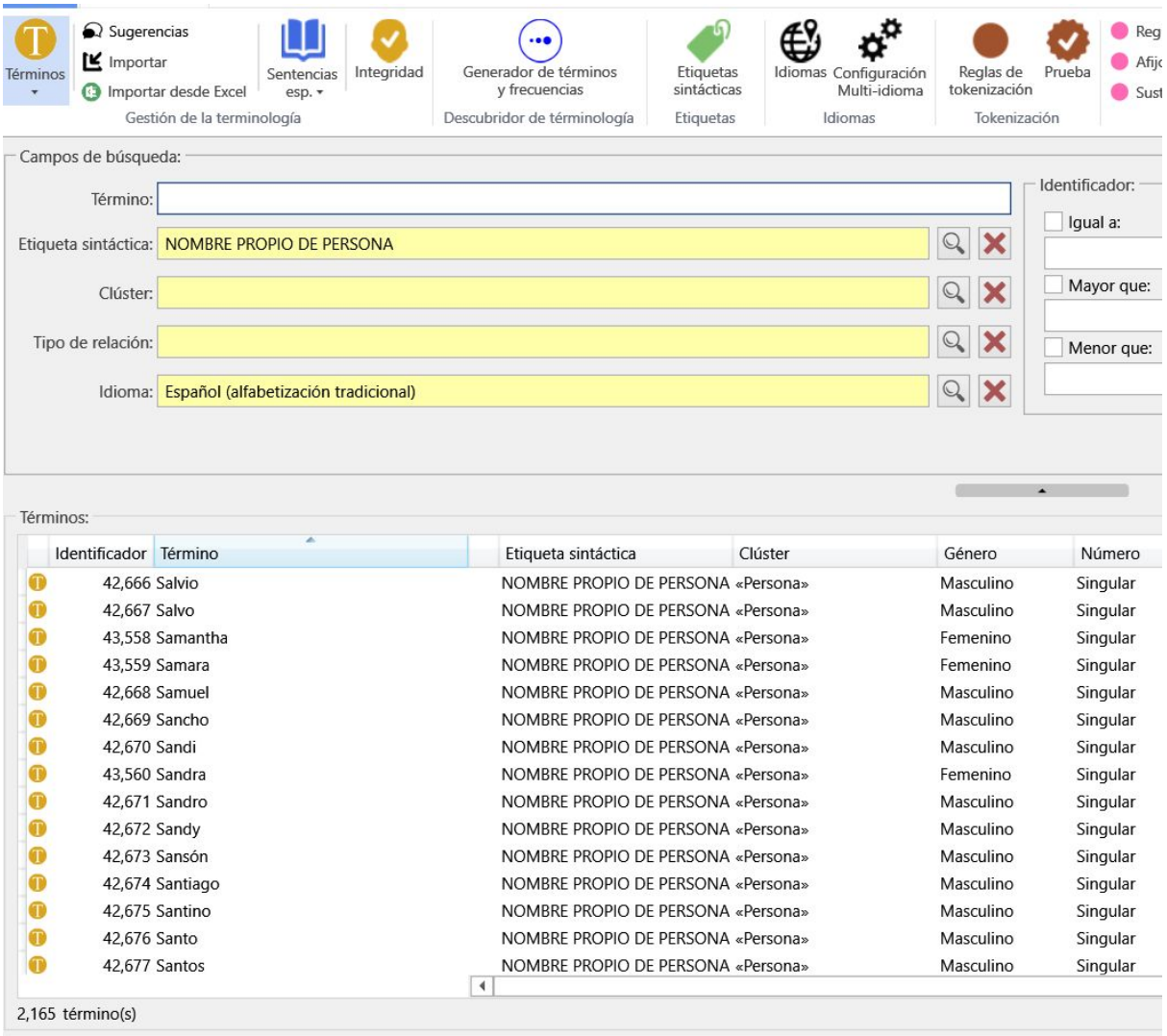
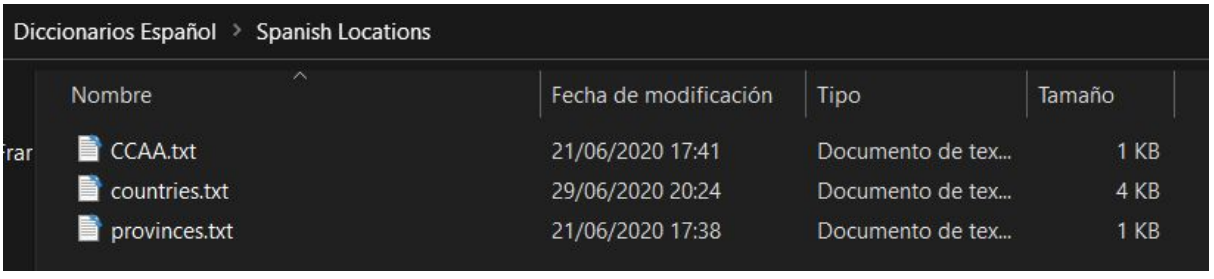


Figura 16: Nombres propios.

Con aquellos sustantivos que fueran nombres de lugares hice lo mismo. Obtuve una lista enorme de países, y luego introduje las provincias y comunidades autónomas de España, con un total de 291 términos.



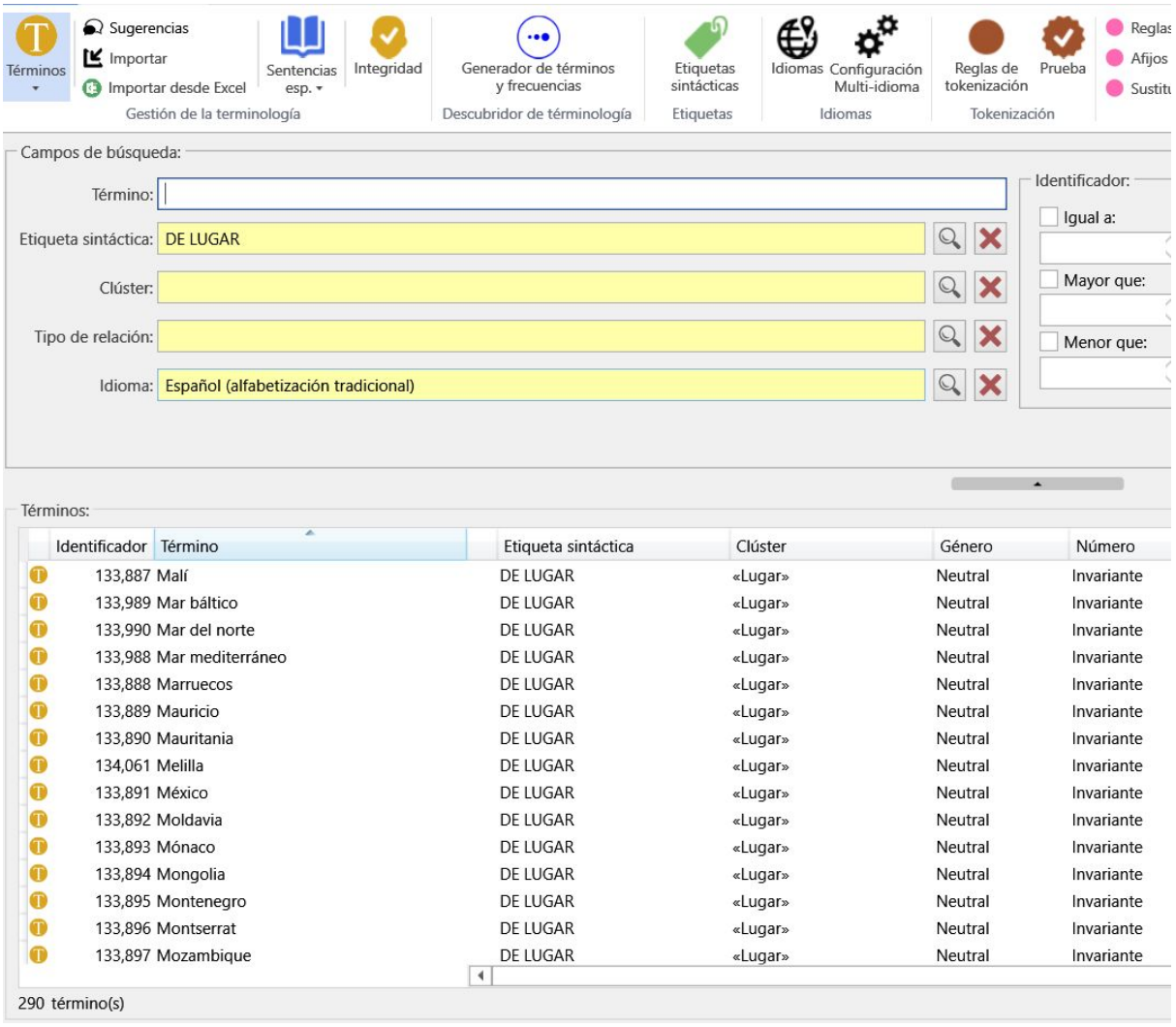


Figura 17: Lugares.

La última tarea que realicé en la base de datos fue clusterizar los sustantivos. Concretamente:

Clúster Lugar: referido a todo aquello relativo a países, ciudades, y espacios que se puedan definir como un lugar (una cocina, un colegio, un restaurante...). Tiene un total de 462 términos.

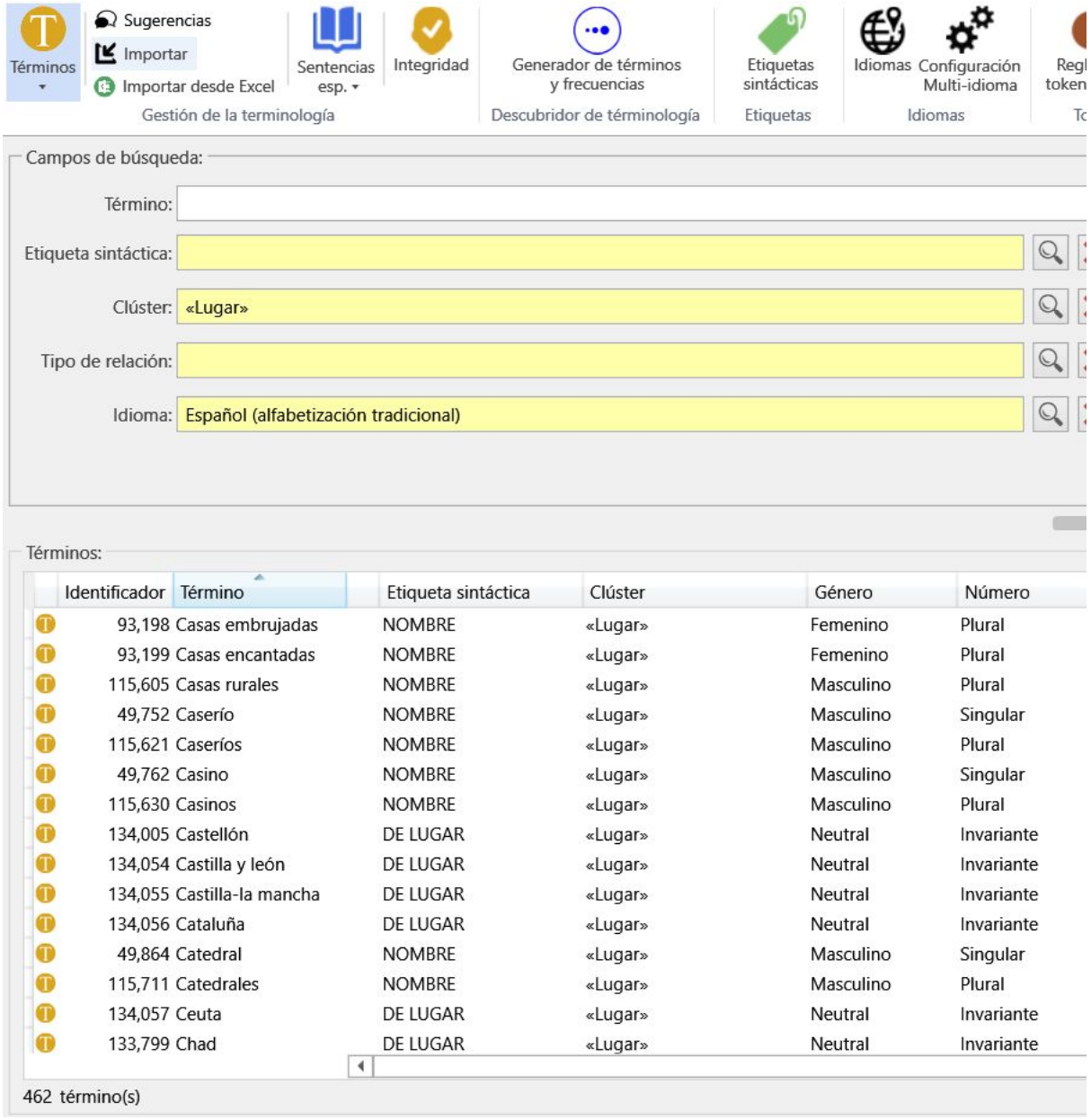


Figura 18: Clúster lugar.

Clúster Persona

Todo aquello que pueda referirse a una persona. Esto incluye oficios, nombres propios de persona, agrupaciones de gente (manifestación) o entidades que se refieran a personas (el pueblo). Para los oficios, como la mayoría acaban en "-ista" fue fácil obtenerlos. Les puse un género invariante (neutral) puesto que se puede decir tanto "el electricista" como "la electricista". Salieron un total de 6.395 términos.

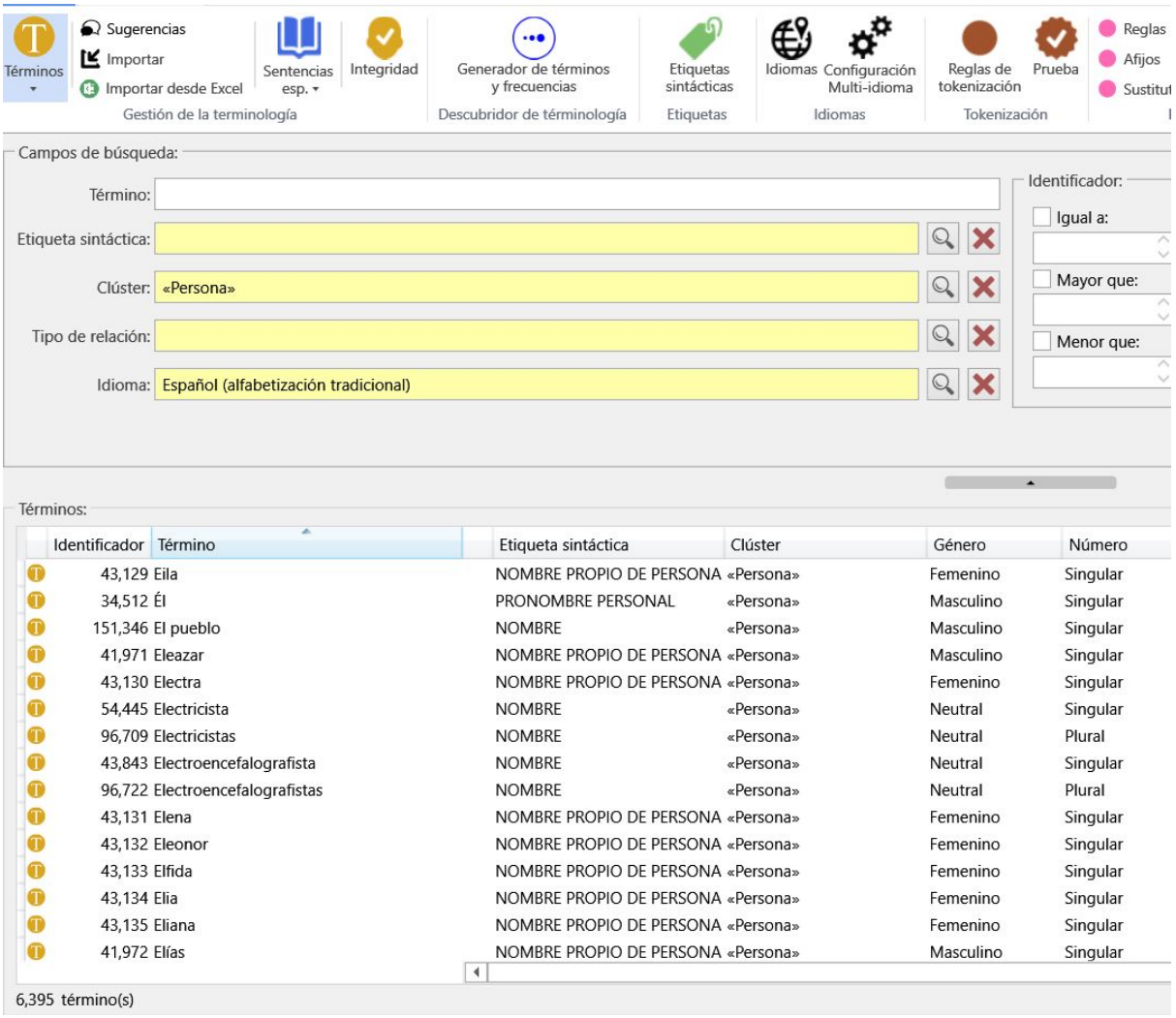


Figura 19: Clúster persona.

Clúster Momento Temporal.

Referido a todo aquello relativo a fechas o acontecimientos históricos. Por ejemplo: Navidad, miércoles de ceniza, enero o trimestre.

Tiene un total de 64 términos.

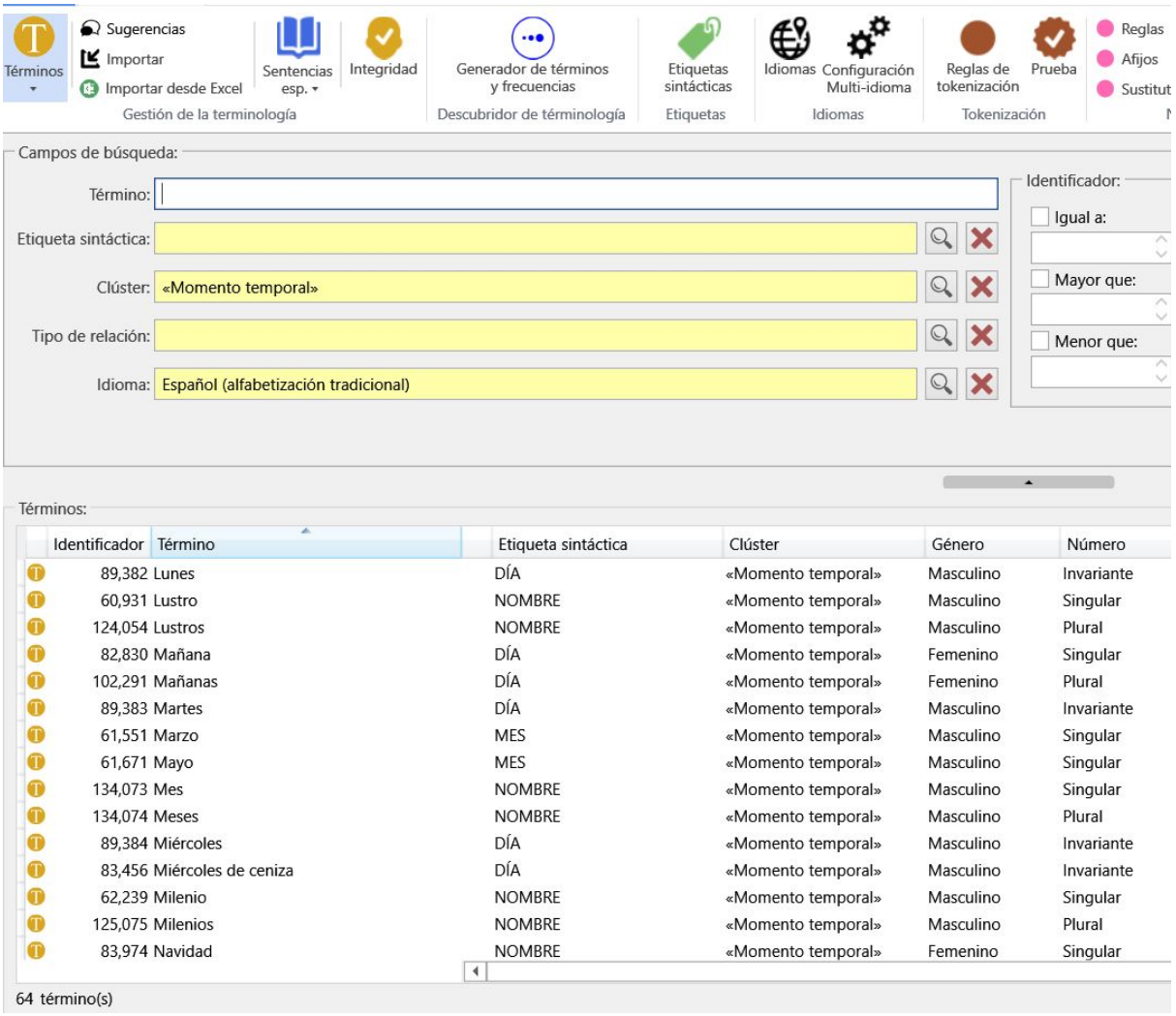


Figura 20: Clúster momento temporal

Aquellos momentos temporales más específicos como los días de la semana, les puse una etiqueta “Día” y a los meses “Mes”.

Con esto ya dispuse de una base de datos bastante completa para comenzar el Trabajo Fin de Grado (en adelante TFG).

3.2 Tecnologías utilizadas

A continuación se detallan las herramientas que se han usado para elaborar las soluciones.

- **Hardware**: Acer Aspire E 15
  - Intel Core i3-7100U (2,4 GHz)
  - Intel HD Graphics 620
  - 12 GB RAM (DDR3)
  - SSD 128 GB
  - Windows 10 Home, build 2004 (Agosto de 2020)

- **Software**: todas las herramientas software, tanto web como de escritorio, son gratuitas.
    - **Visual Studio Community 2019**: es un *IDE* (entorno de desarrollo integrado) compatible con múltiples lenguajes de programación. He usado este porque el entorno de trabajo que me aportó Reuse estaba ya hecho con Visual Studio, usando lenguaje *C#*, y personalmente me gusta este *IDE*.
    - **Windows 10**: es el sistema operativo usado.
    - **SonarQube**: herramienta de análisis de código fuente, se integra con Visual Studio de forma manual o a través de *plug-ins* como **SonarLint**. Los análisis y resultados se pueden realizar de forma local, o en la nube, con plataformas como **SonarCloud**, a la cual estoy integrado con mi cuenta de GitHub.
    - **GitHub**: es una plataforma que permite almacenar proyectos software, sirviendo como sistema de control de versiones. Una vez que consigo un avance importante lo guardo todo allí. Se integra con Visual Studio a través de **GitHub Extension for Visual Studio**.
    - **Trello**: aplicación web que permite administrar proyectos software, registrando las actividades con el sistema *kanban* (tareas en formato tarjeta). Me es muy útil para planificarme.
    - **Tom's Planner**: me permite realizar el diagrama Gantt de actividades, marcando fechas, recursos y demás información útil. Está disponible mediante aplicación web.
    - **Google Docs y Google Drive**: me proporcionan un conjunto de software ofimático (documentos, hojas de cálculo, presentaciones...) y un espacio de almacenamiento en la nube, respectivamente.
    - **Postman**: Es un software que funciona como cliente de una *API*. Permite mandar peticiones de todo tipo, desde *HTTP* plano hasta *REST* o *SOAP*. Usado inicialmente para trabajar con la *API* de Wiktionary, y entender el formato de los resultados.
    - **Notepad++**: editor de texto para trabajar con diferentes archivos de texto, como los diccionarios de la base de datos, o el modelo de *Weka*.
    - **Weka:** *Waikato Environment for Knowledge Analysis* es una plataforma software de aprendizaje automático y minería de datos, desarrollado por la Universidad de Waikato (Nueva Zelanda) y programada en Java.

3.3 Diseño de la solución

Las dos soluciones que se presentan en los apartados siguientes tienen rasgos comunes. Se trabaja con la categoría gramatical de las palabras, así como con el género, el número, y otros conceptos como clusterización, preferencia de entidades y distancia. Se definen como sigue:

- **Categoría gramatical**: se obtiene del proceso de etiquetado con *KM*, se refiere al tipo de palabra (sustantivo, verbo, adjetivo...).
- **Género y número**: propiedades de cada palabra que se obtienen con *KM*. Las categorías son:

    - No definido, singular, plural e invariante para el número.

    - No definido, masculino, femenino e invariante para el género.

  La diferencia entre un género no definido y un género invariante, radica en que en el primer caso tenemos una ausencia de información, mientras que en el segundo, se refiere a que puede formar oraciones tanto con entidades masculinas como femeninas. Quizás un ejemplo ayude a definirlo:

    - Una palabra no contenida en la base de datos, recibirá un género no definido, debido a la ausencia de información.

    - Una palabra contenida en la base de datos, como el pronombre personal "le" tendría género invariante, puesto que nos sirve para acompañarlo o referirnos a entidades tanto masculinas como femeninas. Una parte minoritaria de hispanohablantes encuentra cierta controversia con el uso del leísmo para ambos géneros, y defienden el uso del laísmo para todas las entidades femeninas. No puedo dictaminar cuál ha de ser el uso correcto puesto que ni entiendo en profundidad el tema, ni es el objeto de este trabajo. En cualquier caso, después de cierta investigación, parece conveniente justificar esta decisión por tres motivos principales:

        1. *Laísmo**: es el uso impropio de "la(s)" en función de complemento indirecto femenino, en lugar de "le(s)", que es la forma a la que corresponde etimológicamente ejercer esa función**. El laísmo [...] comienza a fraguarse en la Castilla primitiva durante la Edad Media, pero no consiguió extenderse a la variedad del castellano andaluz, por lo que no se trasladó al español atlántico (Canarias e Hispanoamérica)*. (RAE, 2005).
        2. La gran mayoría de hispanohablantes usan el leísmo frente al laísmo.
        3. Prefiero programar la solución siguiendo las normas de escritura y habla que usa la mayoría de la población.

- **Clusterización**: es una técnica de identificación de patrones, usada para aprendizaje no supervisado automático, y que agrupa los datos en función de características comunes.

- **Preferencia de entidades**: este concepto es muy intuitivo. Se basa en la hipótesis de que dependiendo del tipo de pronombre que genere la anáfora, tendrá más probabilidad de resolverse en un tipo de palabra concreto. Por ejemplo, los pronombres personales tienden a resolverse hacia entidades del tipo persona (sustantivos relativos a profesiones, nombres propios...). Otro ejemplo son los pronombres relativos, como el caso de "donde" que tiende a tener más acierto con entidades que se refieren a lugares (ciudades, países, continentes, lugares cotidianos como las estancias de una casa...etc).

- **Distancia:** este concepto hace referencia al número de palabras que hay entre el elemento que genera la anáfora, y el elemento que la resuelve, contando desde la anáfora hacia atrás. Pongo un ejemplo:

Figura 21: Concepto de distancia.

En el ejemplo anterior, comenzamos a contar desde la anáfora hasta encontrar la solución. Nótese que se ha contado como un salto extra al signo de puntuación. La herramienta *KM* clasifica como *tokens* a las comas, puntos y demás signos de puntuación. Por ese motivo, el total de saltos en el ejemplo anterior asciende a 5.

Aclarados esos conceptos, paso a detallar las soluciones individuales propuestas.

3.3.1 Solución con algoritmia computacional

A la hora de diseñar una solución computacional, recordemos que cuando se aplica a PLN, las características principales suelen ser:

- El programador vuelca toda la lógica que debe seguir el programa en el algoritmo.
- Ante casos no contemplados en dicha lógica, el programa probablemente tomará una decisión errónea, y no será consciente de que ha fallado.
- Suelen ser algoritmos que pertenecen al grupo de métodos de conocimiento limitado: aproximan una solución usando la información morfológica y sintáctica de la que se dispone. Los algoritmos clásicos expuestos en el apartado "Estado del arte" pertenecen a esta categoría.

Antes de nada, debía familiarizarme con la sintaxis de los métodos que usa el entorno de trabajo de *KM*. Afortunadamente, desde Reuse me dieron tanto un manual de uso como ejemplos de código de tareas básicas.

Entendidas las bases de funcionamiento del entorno de trabajo, restaba diseñar la solución. Desde la empresa no me dieron ningún requisito de usuario, ya que no tenían ningún componente previo que resolviera anáforas, y este trabajo era más una investigación en esa línea. Tuve plena libertad para hacer el sistema a mi gusto. Por lo tanto, intenté realizar algo que fuera simple.
Antes de explicar en detalle el algoritmo, quisiera presentar la solución mediante diagramas.

Diagrama de clases

Usamos el diagrama de clases para modelar la estructura estática de un sistema, mediante la descripción de sus elementos y de las relaciones entre ellos. Estos no cambian a lo largo del tiempo. (Seidl, 2015). Es sin duda uno de los diagramas UML más usados, y se aplica en diversas fases del desarrollo software.

En cuanto al formato, se ha seguido lo estipulado en el estándar UML 2.5:

Una clase representa o caracteriza un conjunto de entidades similares, por ejemplo personas, cosas o eventos. Un objeto es una representación concreta de una clase, es decir, su instancia. Las características más relevantes de estas instancias se describen a través de los atributos y los comportamientos a través de las operaciones

En un diagrama de clases, cada clase se representa mediante un rectángulo que puede dividirse en varios compartimentos. El de arriba del todo suele contener el nombre de la clase. El nombre de esta, por convención, suele ir en singular y en negrita. El segundo compartimento suele contener los atributos de la clase, y el tercero los métodos u operaciones.

El nombre del método va seguido de una lista de parámetros entre paréntesis. La lista en sí puede estar vacía. Un parámetro se representa de manera similar a un atributo. La única información obligatoria es el nombre del parámetro. La adición de un tipo, una multiplicidad, un valor predeterminado y otras propiedades, ya sea ordenadas, únicas o sus contrapartes negadas, es opcional. El valor de retorno de una operación es opcional, y se especifica con el tipo de retorno.

Mi solución computacional cuenta con 4 clases.

1. Una clase Core, que contiene el método *main()*, que inicia todo. No contiene la lógica del sistema, sino que únicamente manda ejecutar tareas al resto de clases.
2. Una clase CakeSession que crea una sesión de trabajo conectándose con la herramienta *KM* y desde la cual podemos usar todos los métodos del motor de *KM*. Esta clase me venía ya programada.
3. Una clase AnaphoraSolver, que contiene métodos que van resolviendo las anáforas.

4. Una clase EntitiesManager, que crea y consulta las preferencias de cada pronombre. Esto se explica más adelante.

A la hora de representarlas en el diagrama, existe cierta controversia sobre si se debe o no incluir la clase que contiene el método *main()* en el diagrama.

Argumentos en contra de representar el *main()* en un diagrama de clases UML:

1. No es una clase que se pueda instanciar.
2. Solo se usa como punto de entrada para la aplicación, no forma parte de la estructura.
3. Al no tener atributos ni operaciones, no es interesante modelarlo.

Argumentos a favor de representar el *main()* en un diagrama de clases UML:

1. Es una clase, y por tanto debe estar en el diagrama de clases.
2. Si ayuda a entender mejor cómo funciona el sistema, debe estar representado.

En mi caso he decidido optar por representarlo.

El diagrama de clases quedaría así:

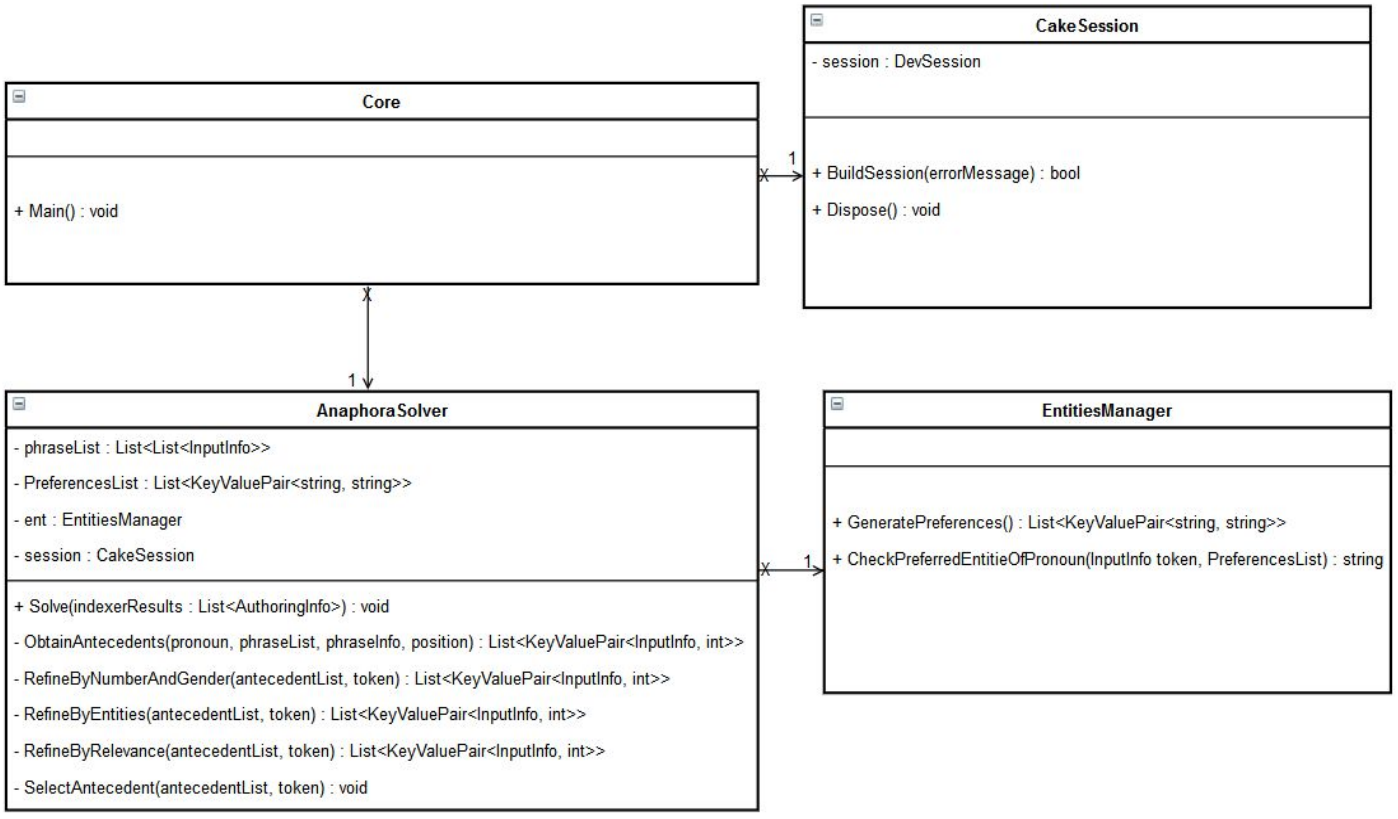


Figura 22: Diagrama de clases.

Por otro lado, delante del nombre del método se incluye un símbolo. En mi diagrama aparece el símbolo '+' que significa público, y '-' que significa privado.

Diagrama de secuencia

Usamos el diagrama de secuencia para modelar las interacciones que suceden entre los objetos de un sistema, de forma secuencial. Con ello podemos observar el flujo de mensajes del mismo. Estos diagramas describen una situación específica, pudiendo perfectamente existir ejecuciones válidas diferentes (Seidl, 2015). El formato usado para representar las diferentes partes sigue el estándar UML 2.5

Las entidades que pueden interaccionar se representan como líneas de vida, formadas por una línea vertical, generalmente discontinua, y arriba de la cual, en un rectángulo, se pone una expresión del tipo "rol : clase". Si solo se especifica la clase, deberá estar precedida por el símbolo ":".

El eje vertical del diagrama modela el orden cronológico de los sucesos, y el eje horizontal modela cada una de las interacciones.

En un diagrama de secuencia, los mensajes se representan mediante una flecha que nace en el emisor y finaliza en el receptor. Según el tipo de flecha, estamos ante un tipo de mensaje u otro.

- **Mensajes síncronos**: el emisor del mensaje espera hasta recibir una respuesta. Se representa mediante una flecha continua con cabeza triangular opaca.

Figura 23: Mensaje síncrono. Fuente: UML @ Classroom

- **Mensajes asíncronos**: el emisor del mensaje no tiene que esperar hasta recibir una respuesta, sino que continúa ejecutándose. Se representa mediante una flecha discontinua con cabeza triangular abierta.

Figura 24: Mensaje asíncrono. Fuente: UML @ Classroom

- **Mensajes de respuesta**: el emisor del mensaje no tiene que esperar hasta recibir una respuesta, sino que continúa ejecutándose. Se representa mediante una flecha discontinua con cabeza triangular abierta.

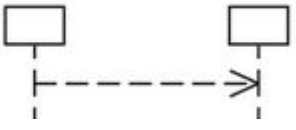


Figura 25: Mensaje de respuesta. Fuente: UML @ Classroom

El diagrama quedaría como sigue:

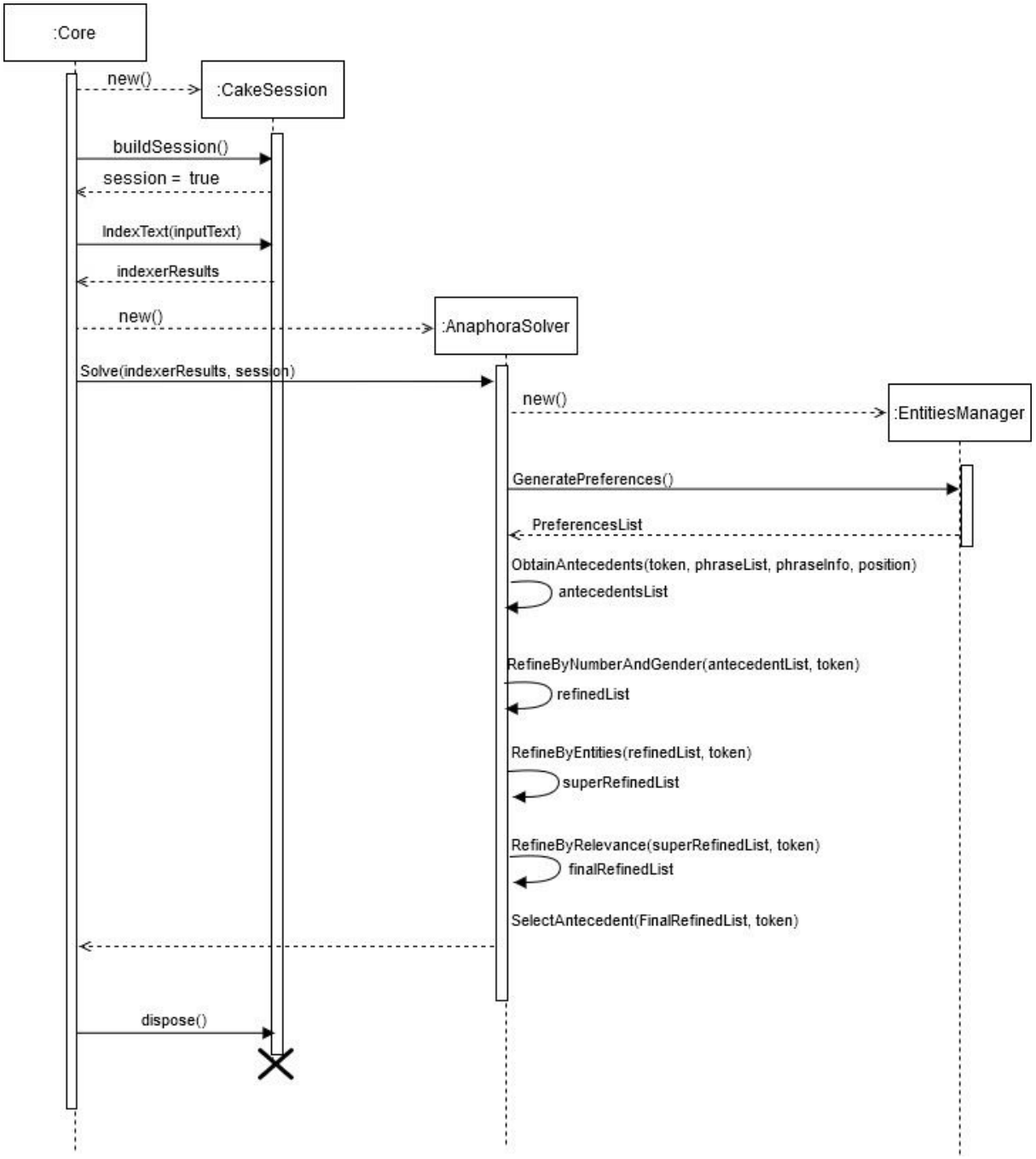


Figura 26: Diagrama de secuencia.

Descripción del algoritmo

Core

Es la clase que contiene el *main()*. Lo primero es indexar el texto. Para ello se crea una sesión de trabajo y se manda el texto a analizar usando la herramienta *KM*. El texto puede obtenerse directamente en un *string* o leyendo desde un archivo de texto. Este proceso devuelve una lista con cada una de las oraciones, ya etiquetadas, lematizadas, normalizadas etc. Esta lista es la que se le manda a AnaphoraSolver para que trabaje con ella.

Una vez resueltas las anáforas cerramos la sesión de trabajo y finaliza el programa.

Todo lo anterior en código es como sigue:

```
9   public class Core
10  {
11      [STAThread]
        0 referencias
12      public static void Main()
13      {
14          CakeSession session = new CakeSession();
15          string errorMessage = string.Empty;
16          bool sessionOK = session.BuildSession(out errorMessage);
17
18          if (!sessionOK && !string.IsNullOrWhiteSpace(errorMessage))
19          {
20              System.Windows.Forms.MessageBox.Show("Error = " + errorMessage);
21          }
22
23          string inputText = "Texto a analizar.";
24
25          //Instead of directly using a string, you can read text from a file. Use the following.
26          //StreamReader sr = File.OpenText(@"C:\Users\santi\Desktop\text.txt");
27          //string inputText = sr.ReadToEnd();
28
29          // Index text
30          List<AuthoringInfo> indexerResults = session.Session.Indexer.IndexText(inputText);
31          Console.WriteLine(inputText);
32
33          //Build an anaphoraSolver and solve the indexed text
34          AnaphoraSolver AnaphoraManager = new AnaphoraSolver();
35          AnaphoraManager.Solve(indexerResults, session);
36
37          // Disconnecting (releases license)
38          session.Dispose();
39      }
40  }
41  }
```

AnaphoraSolver

AnaphoraSolver.Solve()

La primera tarea del método Solve() es generar las preferencias de los pronombres. Esto se explicará más adelante. Después AnaphoraSolver irá leyendo las oraciones de la lista que le llega como parámetro una por una, buscando pronombres.

He optado por generar una ventana deslizante que guarde las *n* oraciones previas a la del pronombre encontrado. Esta ventana se va llenando a medida que se itera sobre la lista de oraciones, y cuando llega al límite programado por el usuario (por ejemplo 4 oraciones), elimina una, siguiendo el algoritmo *FIFO* (*First in, first out*). El límite de oraciones en las que buscar queda definido por una constante.

Una vez que el algoritmo encuentra un pronombre, introduce las oraciones que están en la ventana deslizante, así como la oración en la que se ha encontrado el pronombre en una lista, y le aplica 5 métodos encadenados que irán poco a poco refinando dicha lista.

Todo lo anterior en código es como sigue:

```
#region Constants

private const int PHRASES_LOOK_BACK = 3;

#endregion
```

```
#region Variables

private List<List<InputInfo>> phraseList;
private List<KeyValuePair<string, string>> PreferencesList;
private EntitiesManager ent;
private CakeSession cakeInstance;

#endregion
```

```
public void Solve(List<AuthoringInfo> indexerResults, CakeSession session)
{
    cakeInstance = session;
    ent = new EntitiesManager();
    PreferencesList = ent.GeneratePreferences();

    //List for storing each phrase, up to PHRASES_LOOK_BACK phrases, in MRU order
    phraseList = new List<List<InputInfo>>();

    if (indexerResults != null && indexerResults.Count > 0)
    {
        // Each AuthoringInfo holds the results for each phrase of the input text
        foreach (AuthoringInfo phraseInfo in indexerResults)
        {
            //this integer is used to distinguished between 2 pronouns that occur twice in the same sentence
            //so my algorithm can select the correct antecedents
            int position = 0;
            //We check if the phrase has pronominal pronouns
            foreach (InputInfo token in phraseInfo.InputPhrase)
            {
                // If the token matches a term in the vocabulary, it's stored in the InputInfo properties
                if (token.InputTerm != null)
                {
                    if (token.InputTerm.Kind.IsTerm_TagWithinMyAncestors(cakeInstance.Session.Engine.PronounType))
                    {
                        List<KeyValuePair<InputInfo, int>> antecedentList = ObtainAntecedents(token, phraseList, phraseInfo, position);
                        List<KeyValuePair<InputInfo, int>> RefinedList = RefineByNumberAndGender(antecedentList, token);
                        List<KeyValuePair<InputInfo, int>> SuperRefinedList = RefineByEntities(RefinedList, token);
                        List<KeyValuePair<InputInfo, int>> FinalRefinedList = RefineByRelevance(SuperRefinedList, token);
                        SelectAntecedent(FinalRefinedList, token);
                    }
                    position+=1;
                }
            }
            //We insert the phrase at the first position on phraseList
            phraseList.Insert(0, phraseInfo.InputPhrase);

            //We check if there are more phrases than PHRASES_LOOK_BACK allows, and remove the last if necessary
            if (phraseList.Count > PHRASES_LOOK_BACK)
            {
                phraseList.RemoveAt(PHRASES_LOOK_BACK);
            }
        }
    }
}
```

AnaphoraSolver.ObtainAntecedents()

Este método se encarga de obtener todos los sustantivos que se encuentren en la lista de oraciones que le llega por parámetro. Esos sustantivos serán antecedentes candidatos a resolver la anáfora que generó el pronombre. La búsqueda se realiza en dos zonas:

1. En la propia oración donde está el pronombre, pero únicamente en las palabras precedentes al mismo.
2. En las *n* oraciones anteriores.

La búsqueda tiene por objeto también asignar la distancia, en número de palabras, a las que están los antecedentes del pronombre. Por tanto para asignarlas de forma correcta, la búsqueda se realiza invirtiendo el orden natural de la oración. Dicho de otra manera, se recorre en sentido inverso las oraciones y sus palabras.

A cada antecedente (sustantivo) encontrado, se le asigna la distancia correspondiente. La información se guarda en pares clave-valor, siendo la clave el antecedente, y la distancia el valor.

Lógicamente no todos los antecedentes son válidos, pero eso se refina en métodos posteriores.

Todo lo anterior en código es como sigue:

```
75  private List<KeyValuePair<InputInfo, int>> ObtainAntecedents(InputInfo pronoun, List<List<InputInfo>> phraseList, AuthoringInfo phraseInfo, int position)
76  {
77      List<KeyValuePair<InputInfo, int>> antecedentList = new List<KeyValuePair<InputInfo, int>>();
78      int distance = 0;
79      //We search for antecedents from the phrase on which the pronoun was found (only before the pronoun)
80      //Sometimes the pronoun can be repeated twice or more on the same sentence, now is when int position
81      //prevents the algorithm from stopping prematurely.
82      List<InputInfo> currentPhrase = new List<InputInfo>();
83      int pos = 0;
84      foreach (InputInfo token in phraseInfo.InputPhrase)
85      {
86          if (token.InputText.Equals(pronoun.InputText) && pos == position)
87          {
88              break;
89          }
90          else
91          {
92              currentPhrase.Add(token);
93          }
94          pos+=1;
95      }
96
97      currentPhrase.Reverse();
98
99      if(currentPhrase.Any())
100     {
101         foreach (InputInfo token in currentPhrase)
102         {
103             if (token.InputTerm != null)
104             {
105                 distance += 1;
106                 if (token.InputTerm.Kind.IsTerm_TagWithinMyAncestors(cakeInstance.Session.Engine.NounType))
107                 {
108                     antecedentList.Add(new KeyValuePair<InputInfo, int>(token, distance));
```

```
                }
            }
        }
    }

    //We search for antecedents (any type nouns) on our phraseList
    if (phraseList.Any())
    {
        foreach (List<InputInfo> phrase in phraseList)
        {
            //We reverse the order of the words on that phrase, so the distance score is accurate
            phrase.Reverse();

            foreach (InputInfo token in phrase)
            {
                if (token.InputTerm != null)
                {
                    //We increment the distance value for subsequent salience values
                    distance += 1;
                    if (token.InputTerm.Kind.IsTerm_TagWithinMyAncestors(cakeInstance.Session.Engine.NounType))
                    {
                        antecedentList.Add(new KeyValuePair<InputInfo, int>(token, distance));
                    }
                }
            }
        //We restore the order of the words on that phrase so the next iteration wont mess up
        phrase.Reverse();
        }
    }
    return antecedentList;
}
```

AnaphoraSolver.RefineByNumberAndGender()

Este método elimina de la lista de antecedentes aquellos que no coincidan en género y número con el pronombre. Tanto la lista como el pronombre son parámetros de entrada.

Lo anterior en código es como sigue:

```
private List<KeyValuePair<InputInfo, int>> RefineByNumberAndGender(List<KeyValuePair<InputInfo, int>> antecedentList, InputInfo token)
{
    List<KeyValuePair<InputInfo, int>> RefinedAntecedentList = antecedentList;
    //We refine by number and gender, using a copy of the list to be thread-safe
    var copy = RefinedAntecedentList.ToArray();
    foreach (var item in copy)
    {
        if (token.Number.ToString().Equals("Without") || token.Gender.ToString().Equals("Neutral"))
        {
            continue;
        }
        else
        {
            if (token.Number.ToString().Equals("Singular"))
            {
                if (item.Key.Number.ToString().Equals("Plural"))
                {
                    RefinedAntecedentList.Remove(item);
                }
            }
            else
            {
                if (item.Key.Number.ToString().Equals("Singular"))
                {
                    RefinedAntecedentList.Remove(item);
                }
            }
        }
```

```
        if (token.Gender.ToString().Equals("Without") || token.Gender.ToString().Equals("Neutral"))
        {
            continue;
        }
        else
        {
            if (token.Gender.ToString().Equals("Masculine"))
            {
                if (item.Key.Gender.ToString().Equals("Feminine"))
                {
                    RefinedAntecedentList.Remove(item);
                }
            }
            else
            {
                if (item.Key.Gender.ToString().Equals("Masculine"))
                {
                    RefinedAntecedentList.Remove(item);
                }
            }
        }

    }
    return RefinedAntecedentList;
}
```

AnaphoraSolver.RefineByEntities()

Este método se encarga de eliminar aquellos antecedentes que pertenezcan a un clúster diferente al preferido por el pronombre. Recordemos que en AnaphoraSolver.Solve() guardábamos en memoria las preferencias de cada pronombre. Estas preferencias están definidas en la clase EntitiesManager, y es como sigue:

```
public List<KeyValuePair<string, string>> GeneratePreferences()
{

//Before handling phrases, generate entitie preferences for each pronoun
  List<KeyValuePair<string, string>> EntitiesPreferences = new List<KeyValuePair<string, string>>() {
     //For personal pronouns, they all resolve into the same cluster.
      new KeyValuePair<string, string>("yo", "Persona"),
      new KeyValuePair<string, string>("tú", "Persona"),
      new KeyValuePair<string, string>("ti", "Persona"),
      new KeyValuePair<string, string>("él", "Persona"),
      new KeyValuePair<string, string>("ella", "Persona"),
      new KeyValuePair<string, string>("ellas", "Persona"),
      new KeyValuePair<string, string>("ello", "Persona"),
      new KeyValuePair<string, string>("ellos", "Persona"),
      new KeyValuePair<string, string>("mí", "Persona"),
      new KeyValuePair<string, string>("nosotras", "Persona"),
      new KeyValuePair<string, string>("nosotros", "Persona"),
      new KeyValuePair<string, string>("vosotros", "Persona"),
      new KeyValuePair<string, string>("vosotras", "Persona"),
      new KeyValuePair<string, string>("usted", "Persona"),
      new KeyValuePair<string, string>("ustedes", "Persona"),
      new KeyValuePair<string, string>("me", "Persona"),

      //For other pronouns, even if they are the same pronoun type, they can belong to different clusters
      //(ex: cuando, donde) which refers to time or locations.

      //Relative pronouns
      new KeyValuePair<string, string>("cuando", "Momento temporal"),
      new KeyValuePair<string, string>("donde", "Lugar"),
      new KeyValuePair<string, string>("cuanta", "Cantidad"),
      new KeyValuePair<string, string>("cuanto", "Cantidad"),
      new KeyValuePair<string, string>("cuantas", "Cantidad"),
      new KeyValuePair<string, string>("cuantos", "Cantidad"),
      new KeyValuePair<string, string>("quien", "Persona"),
      new KeyValuePair<string, string>("quienes", "Persona"),

      //Interrogative pronouns
      new KeyValuePair<string, string>("quién", "Persona"),
      new KeyValuePair<string, string>("dónde", "Lugar"),
      new KeyValuePair<string, string>("cuándo", "Momento temporal")
  };
  return EntitiesPreferences;
```

Si recordamos del apartado 3.1 “Construcción de la base de datos”, el último paso que realicé fue clusterizar los sustantivos. De esta manera puedo ahora establecer preferencias para cada pronombre. Como vemos en el código anterior, los pronombres personales prefieren resolver sustantivos del clúster “Persona”, y otros pronombres relativos como “cuando” y “donde” resuelven hacia el clúster “Momento temporal” y “Lugar” respectivamente.

Lógicamente estas preferencias no son exhaustivas, al igual que la clusterización de sustantivos tampoco lo es. Ambas pueden ser ampliadas y mejoradas, y más clústeres y preferencias se pueden introducir. En cualquier caso, queda esa tarea para trabajos futuros.

Recapitulando, el método RefineByEntities eliminará de la lista de antecedentes aquellos que no coincidan con el clúster que prefiera el pronombre. Si el pronombre no tiene asignada una preferencia, no se hace nada.

Todo lo anterior en código es como sigue:

```
1 referencia
private List<KeyValuePair<InputInfo, int>> RefineByEntities(List<KeyValuePair<InputInfo, int>> antecedentList, InputInfo token)
{
    List<KeyValuePair<InputInfo, int>> SuperRefinedAntecedentList = antecedentList;
    string PreferredEntitie = ent.CheckPreferredEntitieOfPronoun(token, PreferencesList);

    if (PreferredEntitie != null && PreferredEntitie != "Not found")
    {
        var copy = SuperRefinedAntecedentList.ToArray();
        foreach (var item in copy)
        {
            if (!item.Key.InputTerm.Clusters.Any())
            {
                SuperRefinedAntecedentList.Remove(item);
            }
            else
            {
                foreach (SemanticItem s in item.Key.InputTerm.Clusters)
                {
                    if (!s.Name.Equals(PreferredEntitie))
                    {
                        SuperRefinedAntecedentList.Remove(item);
                    }
                }

            }

        }
    }
    return SuperRefinedAntecedentList;
}
```

Donde el método CheckPreferredEntitieOfPronoun pertenece a la clase EntitiesManager y funciona así:

```
1 referencia
public string CheckPreferredEntitieOfPronoun(InputInfo token, List<KeyValuePair<string, string>> PreferencesList)
{
    foreach (KeyValuePair<string, string> kvp in PreferencesList)
    {
        if (token.InputText.Equals(kvp.Key))
        {
            return kvp.Value;
        }
    }
    return "Not found";
}

}
}
```

AnaphoraSolver.RefineByRelevance()

Este método se encarga de eliminar de entre los antecedentes supervivientes, los menos relevantes. Este paso es bastante impreciso, por lo cual solo he introducido un filtro de relevancia, para dar un ejemplo de cómo funcionaría.

Digamos que en el paso anterior de RefineByEntities, el pronombre es personal, y por tanto prefiere resolver hacia sustantivos del tipo Persona. Deja sustantivos del clúster persona en la lista. Una vez que dicha lista llega a RefineByRelevance, si detecta que hay nombres propios de persona, les va a dar más relevancia.

El código es como sigue:

```
227     private List<KeyValuePair<InputInfo, int>> RefineByRelevance(List<KeyValuePair<InputInfo, int>> antecedentList, InputInfo token)
228     {
229         List<KeyValuePair<InputInfo, int>> FinalRefinedAntecedentList = antecedentList;
230         //If the antecedentList has nouns related to Persona cluster,
231         //usually proper nouns are more relevant (ex: Juan es cocinero. Él estudió en París.)
232         if (token.InputTerm_Tag.ToString().Equals("PRONOMBRE PERSONAL"))
233         {
234             int flag = 0;
235             var copy = FinalRefinedAntecedentList.ToArray();
236             foreach (var item in copy)
237             {
238                 if (item.Key.InputTerm_Tag.ToString().Equals("NOMBRE PROPIO DE PERSONA"))
239                 {
240                     flag = 1;
241                     break;
242                 }
243             }
244             //Since we detected proper nouns, lets get rid of all non-proper nouns
245             if (flag==1)
246             {
247                 foreach (var item in copy)
248                 {
249                     if (!item.Key.InputTerm_Tag.ToString().Equals("NOMBRE PROPIO DE PERSONA"))
250                     {
251                         FinalRefinedAntecedentList.Remove(item);
252                     }
253                 }
254             }
255         }
256         return FinalRefinedAntecedentList;
257     }
```

Un ejemplo de ejecución sería el siguiente:
Para la oración “Marta vive en una casa de Barcelona porque es una chica que le gusta vivir cerca del mar.”
Se van detectando y resolviendo los pronombres. Llega un momento en que se detecta el pronombre “le”.
Se escogen como antecedentes {chica, Barcelona, casa, Marta}. Como el pronombre es personal, su preferencia es resolver hacia sustantivos del clúster “Persona”, y por tanto en el paso AnaphoraSolver.RefineByEntities() la lista queda reducida a {chica, Marta}.

Aquí es donde tiene más importancia contar con un filtro de relevancia. Si no tuviéramos nada, la anáfora se resolvería a “chica” puesto que está más cerca del pronombre. Sin embargo, con RefineByRelevance se priorizan los nombres propios, y se elige a “Marta” aunque esté más lejos.

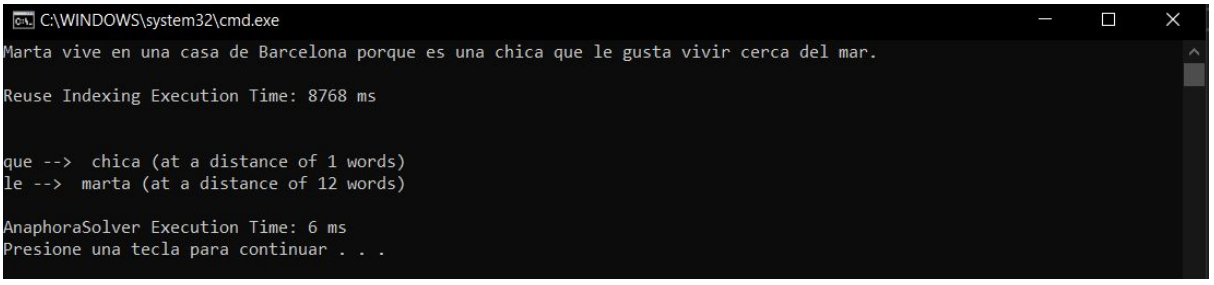


Figura 27: Ejecución con filtro de relevancia.

Lógicamente hay multitud de ejemplos donde estos filtros de relevancia no funcionan, por eso solo he programado el filtro de entidades persona como ejemplo de lo que se podría hacer. En cualquier caso es bastante experimental. Ejemplo que fallaría:

"Marta vive en una casa de Barcelona porque a su madre le gusta vivir cerca del mar."

En este caso "le" debería resolver a "madre", pero al tener activo el filtro de relevancia, se escoge a "Marta".

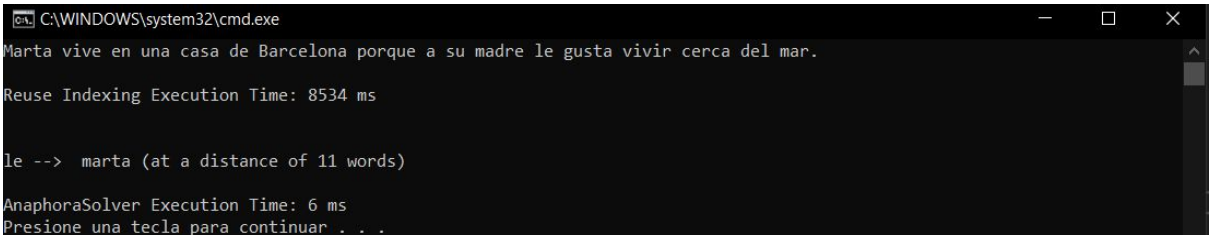


Figura 28: Ejecución incorrecta con filtro de relevancia.

Resumiendo: los filtros por relevancia están bien, pero es mejor realizarlos con IA para obtener mejores resultados. Si se intenta realizar con algoritmia computacional, los casos no contemplados van a fallar.

AnaphoraSolver.SelectAntecedent()

Este último método recibe la lista de antecedentes, y elige al que tenga asociada una distancia más pequeña. Estadísticamente el antecedente más cercano al pronombre suele ser el correcto. Como es evidente, todos podemos pensar en ejemplos que no cumplen esto, pero es la desventaja de los algoritmos computacionales. Es muy difícil programar algo que cubra todos los casos.

Lo anterior en código es como sigue:

```
259         private void SelectAntecedent(List<KeyValuePair<InputInfo, int>> antecedentList, InputInfo token)
260         {
261             KeyValuePair<InputInfo, int> result = new KeyValuePair<InputInfo, int>();
262             int minDistance = Int32.MaxValue;
263 
264             //Give salience score based on distance to the pronoun
265             foreach (KeyValuePair<InputInfo, int> antecedent in antecedentList)
266             {
267                 if (antecedent.Value < minDistance)
268                 {
269                     minDistance = antecedent.Value;
270                     result = antecedent;
271                 }
272             }
273             if (result.Key != null)
274             {
275                 Console.WriteLine(token.InputTerm.ToString() + " -->  " + result.Key.InputText + " (at a distance of " + result.Value+ " words)");
276             }
277             else
278             {
279                 Console.WriteLine(token.InputTerm.ToString() + " -->  " + "Not found");
280             }
281 
282         }
283 
284     }
285 }
```

EntitiesManager

Es la clase auxiliar que genera las preferencias de los pronombres y también se encarga de devolver la preferencia de un pronombre que le llegue como parámetro.

El código de ambos métodos, GeneratePreferences() y CheckPreferredEntitieOfPronoun() ya han aparecido en la explicación de AnaphoraSolver.

En definitiva, he intentado construir un algoritmo sencillo, que trabaja con una lista eficiente en memoria y tiempo.

El diseño del código podría haber cambiado un poco, particularmente se podría haber refactorizado la parte de AnaphoraSolver, que trabaja con varios métodos encadenados. Quizás separar cada método en clases separadas podría haber sido más correcto. Pero decidí dejarlos en la misma clase por varios motivos:

1. Los métodos de AnaphoraSolver son secuenciales. Cada uno trabaja con la salida del anterior (misma estructura de memoria).
2. Son métodos muy pequeños (de media unas 40 líneas).
3. Queda muy claro qué hace cada método, y para depurar el código es fácil saber dónde están los problemas.
4. Es escalable. En caso de que se quiera añadir o retirar funcionalidad, como cada método realiza tareas separadas muy distinguibles, es sencillo modificar la parte que se quiere (cambiar los filtros de género, meter filtros de relevancia...).
5. Si separamos los métodos en clases separadas vamos a necesitar instanciar todas ellas desde AnaphoraSolver. Una alternativa para no tener que hacer eso sería declarar las clases como estáticas, pero no soy partidario de tener muchas clases estáticas, puesto que dificultan la mantenibilidad del código, además de generar una cantidad de dependencias y acoplamientos innecesarios. Lógicamente es discutible a partir de qué cantidad de clases y/o métodos estáticos tenemos un problema, pero en general es una mala práctica. (No siempre se puede evitar, y no siempre afecta a la mantenibilidad o acoplamiento del diseño. Pongamos por ejemplo la clase *Math(),* que es enteramente estática y se usa muy a menudo en diversos programas).
6. Evaluando el código con métricas y herramientas que se detallan en el apartado 4 “Resultados”, el código está calificado como muy bueno, aunque se aconseja refactorizar algunos métodos. La complejidad cognitiva que se recomienda es 15, y los métodos que exceden esa cantidad, tienen 17, o 19, así que no es necesario refactorizarlo.

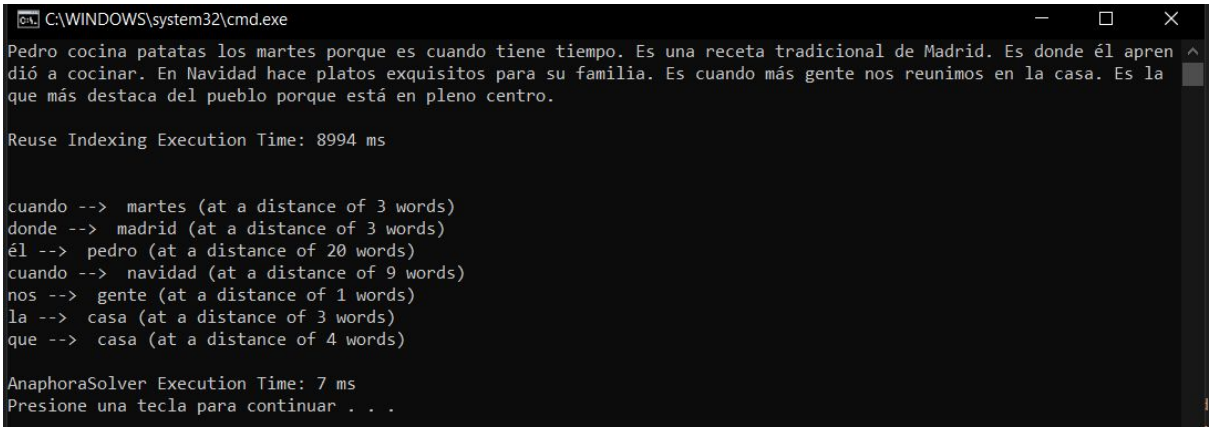
CakeSession

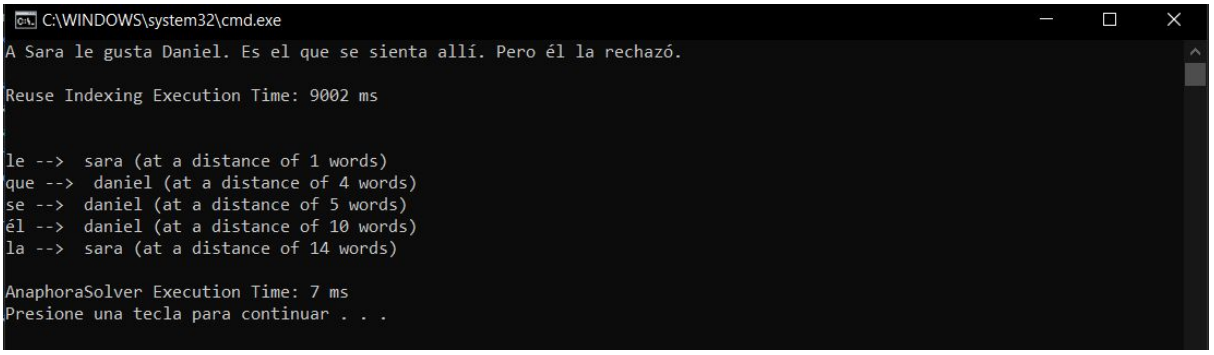
Esta clase genera una sesión de trabajo conectándose a las bases de datos de *KM* y otorga punteros para usar todas sus funcionalidades (*Engine*).

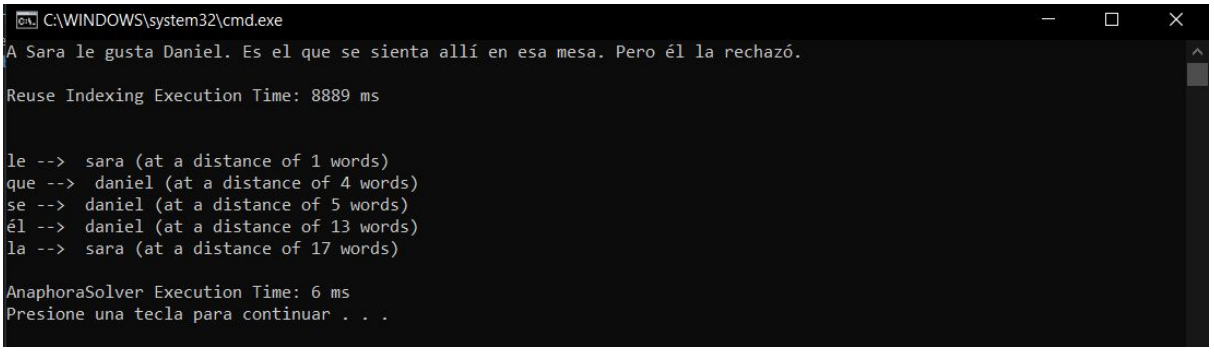
Esta clase me la dieron hecha, es la que venía en el entorno de trabajo que me facilitó Reuse.

Ejemplos de ejecución

Ejemplos de ejecución del algoritmo entero son:







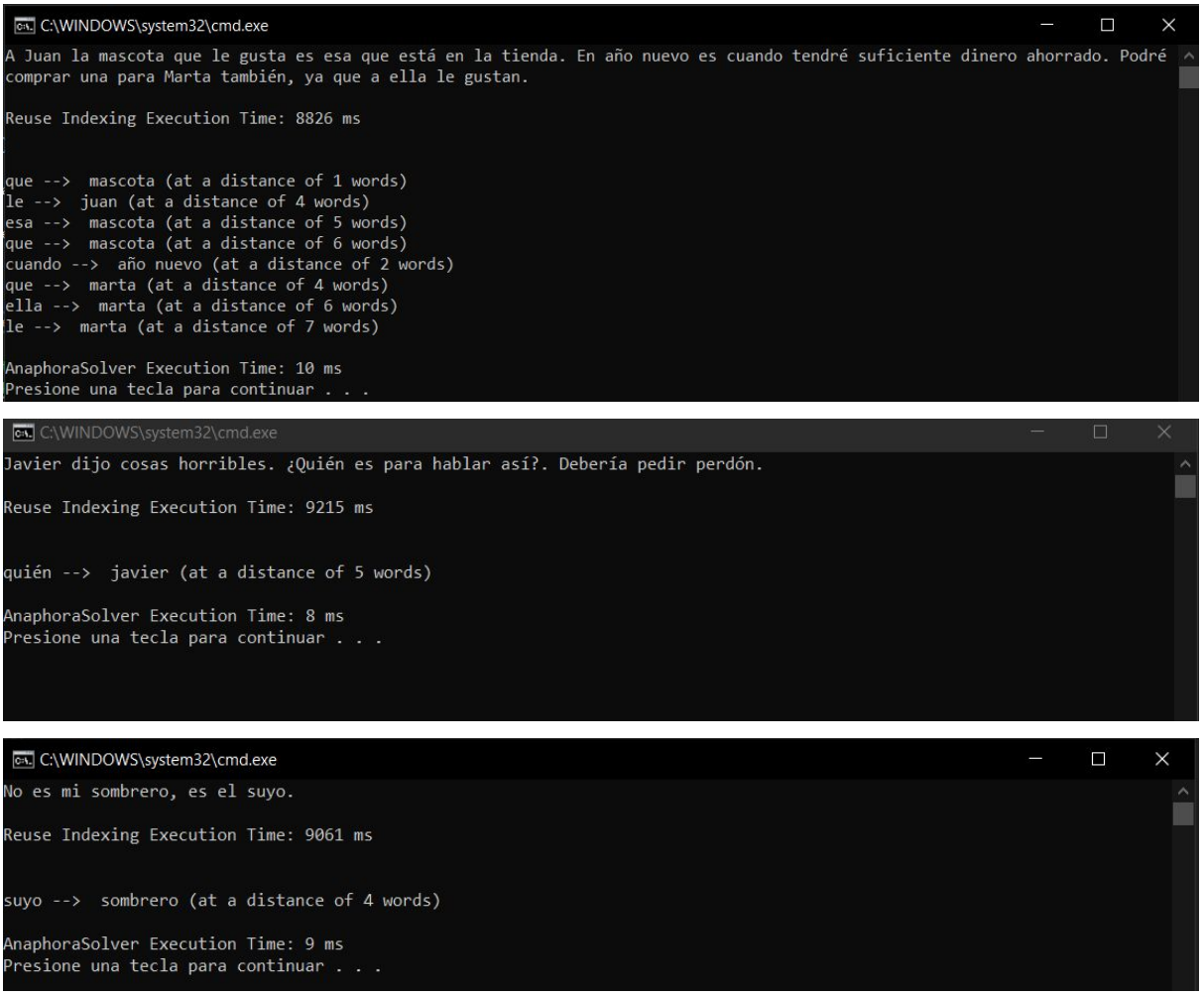


Figura 29: Ejemplos de ejecución del algoritmo.

Como se puede ver resuelve pronombres personales, relativos, demostrativos, interrogativos, posesivos y reflexivos. Eso sí, todo son anáforas pronominales (se resuelven a sustantivos). Otras anáforas como la adjetiva o la adverbial no están contempladas en esta solución por ser menos frecuentes, y quedan para trabajos futuros.

3.3.2 Solución con IA

En un principio pensé que el TFG iba a limitarse a soluciones computacionales, pero afortunadamente ha dado tiempo a desarrollar un enfoque alternativo. La idea que hay detrás de esta solución consiste en generar un modelo de entrenamiento para que una IA aprenda a resolver anáforas.

La herramienta usada ha sido *Weka* (*Waikato Environment for Knowledge Analysis*).

Breve introducción a Weka

Es una herramienta que contiene una colección de herramientas de visualización y algoritmos de análisis de datos y modelado predictivo. Contiene también una interfaz gráfica para acceder y configurar las herramientas.

Figura 30: Interfaz gráfica inicial de Weka.

La interfaz explorer es la única que he usado. Permite preprocesar los datos de entrada, aplicar clasificaciones, clusterizar, y realizar operaciones sobre los atributos del modelo.

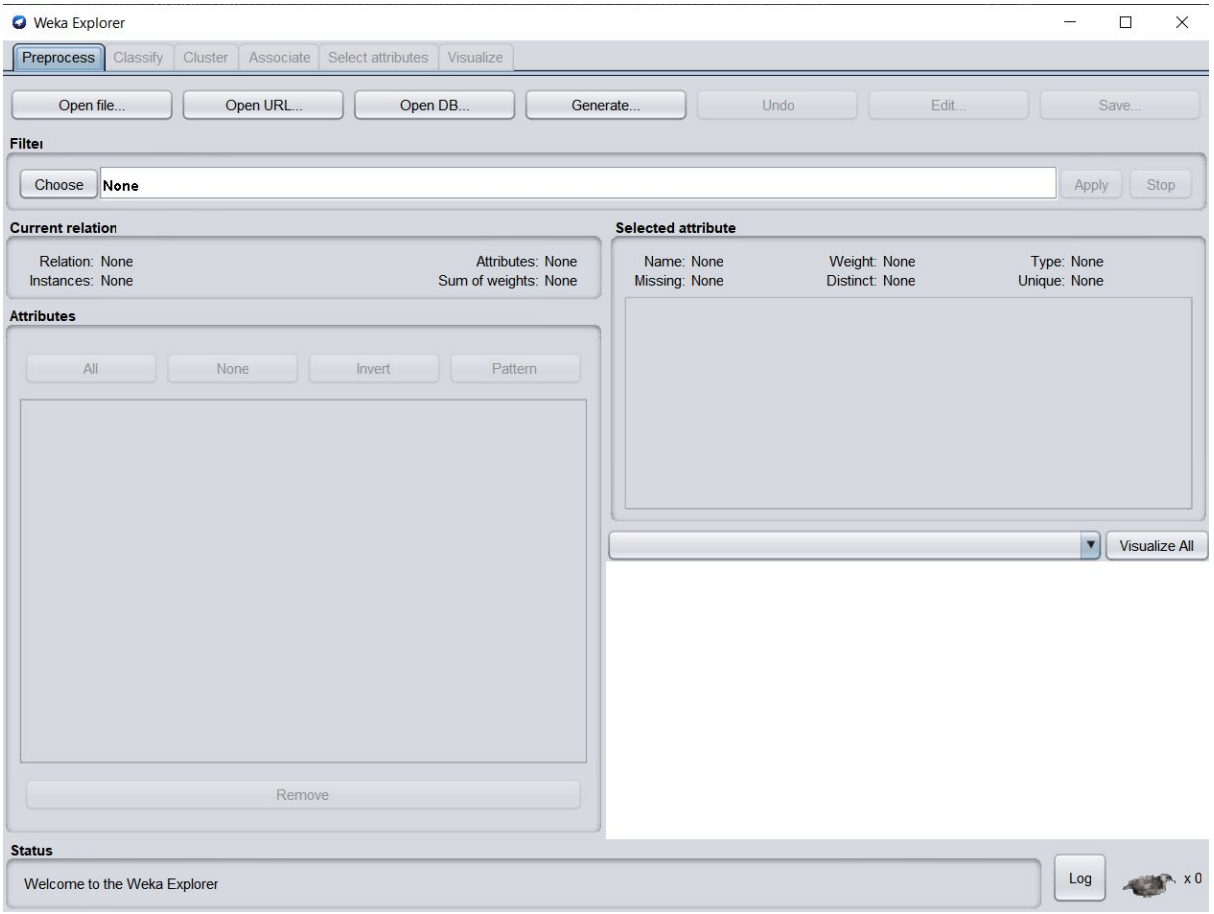


Figura 31: Interfaz del explorer Weka.

Presionando en *Open file* podemos seleccionar un archivo de entrada, en mi caso con la extensión *.arff*

En el botón *Choose* podemos elegir diferentes filtros que preprocesan los atributos del modelo de entrada.

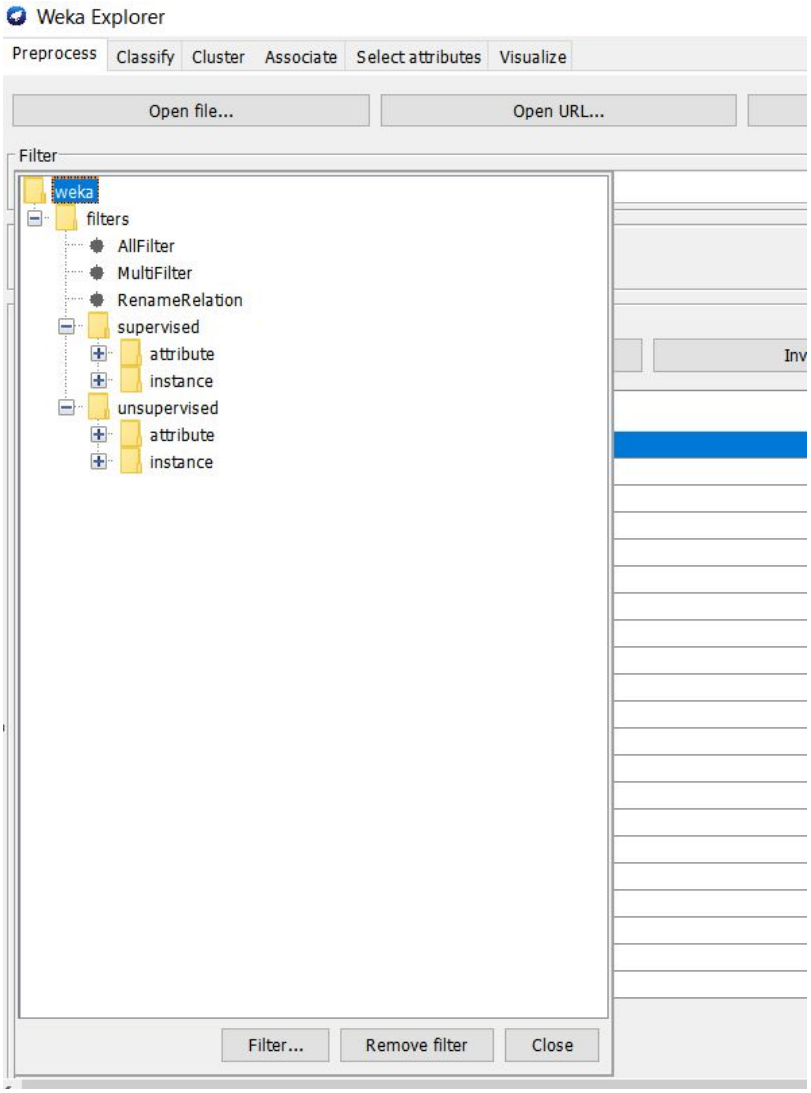


Figura 32: Filtros para preprocesado de datos.

Estos filtros realizan diferentes tareas. Algunos eliminan ciertos atributos, o los modifican, o crean otros nuevos en función de los originales.

En la pestaña *Classify* podemos elegir diferentes clasificadores, que generan reglas a partir del modelo. Después se obtiene el porcentaje de acierto que Weka ha conseguido. Los clasificadores son de varias familias:

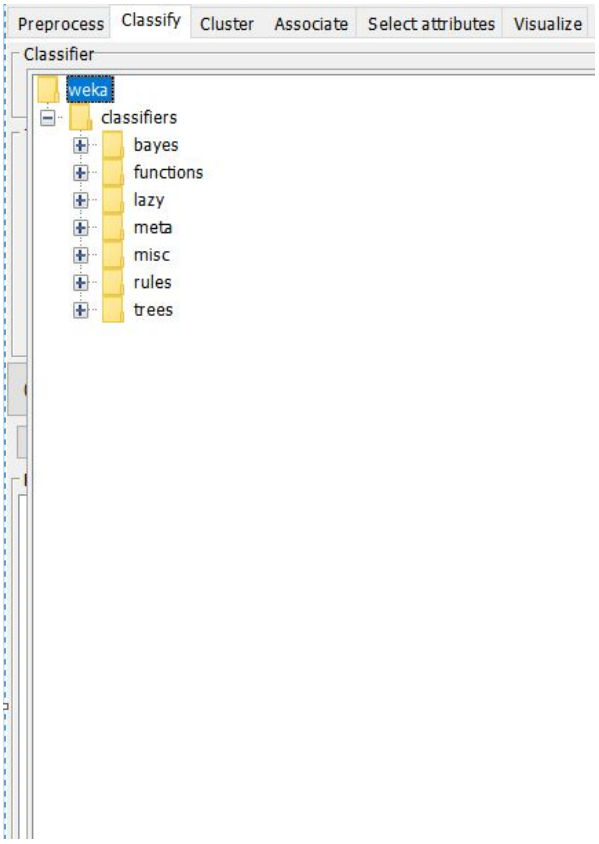


Figura 33: Familias de clasificadores en Weka.

Una vez seleccionado el clasificador, se inicia el entrenamiento.

El formato de los modelos para entrenar siguen una estructura propia de Weka, con un formato .arff o .xrff. En esta solución se ha usado un archivo .arff

En la documentación de Weka encontramos cómo se construyen estos ficheros.

Los ficheros ARFF tienen dos secciones principales. La primera es la cabecera, y la segunda la sección de datos.

La cabecera contiene el nombre de la relación, una lista de atributos y sus tipos. El orden será el que sigan las columnas de la sección de datos. Un ejemplo sería el siguiente:

```
@RELATION iris

@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}
```

Figura 34: Ejemplo de cabecera de fichero .arff

La relación se llama iris, un tipo de flor. Los atributos son características de la flor, como el ancho del sépalo o del pétalo. El último atributo es el de clase, y define o resuelve el caso de entrenamiento. En el ejemplo anterior, los posibles valores del atributo de clase son diferentes tipos de flores iris.

La sección de datos son valores separados por comas, que se corresponden con los atributos. Un salto de línea marca el final de la relación. Un ejemplo sería:

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
```

Figura 35: Ejemplo de sección de datos de fichero .arff

Traducido a palabras, sería algo como: "si la longitud del sépalo es 5.1, la anchura 3.5, la longitud del pétalo 1.4 y su anchura 0.2, el tipo de flor iris es setosa".

Creación del modelo

Al igual que con el algoritmo computacional, esta solución también está enfocada hacia el idioma español. En Internet he sido incapaz de encontrar un corpus anotado con anáforas resueltas en español. En inglés sí que había, pero he preferido hacer mi propio corpus en español porque puede ser de ayuda para trabajos futuros.

El primer paso fue decidir de qué forma modelar las oraciones, de tal manera que se supiera la etiqueta gramatical de la palabra, el género y el número. La idea consiste en sustituir cada palabra por el identificador numérico de su categoría gramatical. Este identificador lo asigna la herramienta *KM*. Por ejemplo los sustantivos tienen el 492.

Después agregamos al final de ese número una "F" si es femenino, una "M" si es masculino y una "W" si es invariante o no determinado. Para el número se agrega una "P" para plural o una "S" para singular.

Si en una oración hay un tipo de palabra que se repite, por ejemplo nombres de ciudades, pondremos el mismo código para ambas (el de nombre propio de lugar) pero a la segunda le añadiremos al final un 2, para distinguirla de la primera.

Cada oración de entrenamiento tiene una longitud de 20 posibles palabras y la solución de la anáfora al final.

Un ejemplo de modelado sería el siguiente:

Figura 36: Ejemplo de modelado Weka.

Como se puede ver en el ejemplo, cada palabra se sustituye por el identificador numérico, seguido del género y del número. Al final del todo se pone la solución de la anáfora generada por el pronombre. En este caso "Ella" marcado en rojo, se resuelve a "Marta" marcado en verde. Por eso se añade al final el identificador numérico de "ella" que es "548FS". Corresponde con "nombre propio de persona, femenino, singular".

Para el modelo he resuelto 100 anáforas pronominales de diferentes tipos: posesivos, relativos, personales…

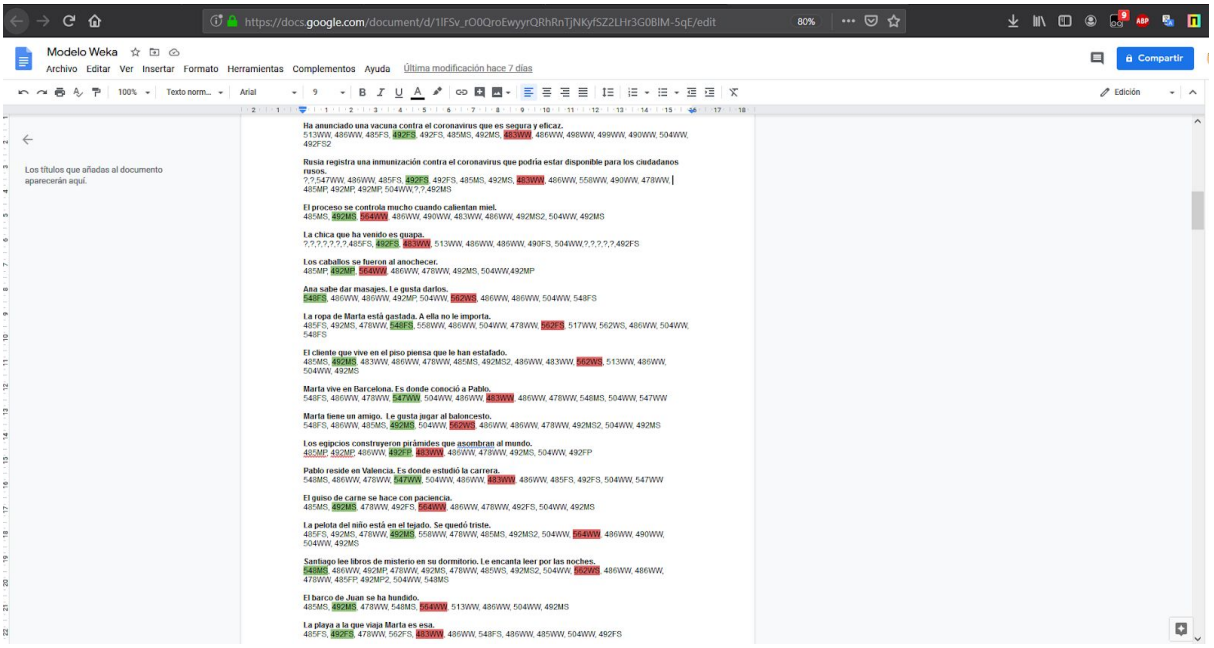


Figura 37: Ejemplos de oraciones resueltas para el modelo Weka.

Como las oraciones suelen tener menos de 20 palabras, a la hora de construir el modelo, se coloca una interrogación en las palabras que falten. La anáfora se coloca siempre en la misma posición. En este caso la posición 10.

El modelo queda así:

Y así hasta los 100 casos de entrenamiento:

Figura 38: Archivo .arff con el modelo de entrenamiento.

4. RESULTADOS

4.1 Resultados en la creación de la base de datos y diccionarios

He construido 7 diccionarios en formato txt:

- Sustantivos femeninos en singular.
- Sustantivos femeninos en plural.
- Sustantivos masculinos en singular.
- Sustantivos masculinos en plural.
- Nombres propios de persona.
- Nombres propios de sitios (países, ciudades, comunidades autónomas...)
- Adjetivos.

El total de términos en la base de datos se distribuye de la siguiente manera:

Tabla 1: Resultados de la inserción de términos en la base de datos. Elaboración propia.

	Términos que venían en la base de datos	**Términos nuevos introducidos**	**Total términos**
Sustantivos	446	91.888	92.334
Verbos	6.336	658	6.994
Adjetivos	909	17.238	18.147
Determinantes	129	0	129
Pronombres	87	13	100
TOTAL	8.264	109.797	118.063

Por otro lado los resultados de la clusterización de los sustantivos son los siguientes:

Tabla 2: Resultados de la clusterización de sustantivos en la base de datos. Elaboración propia.

	Cantidad de términos
Persona	6.396
Lugar	462
Momento temporal	64

4.2 Resultados de la solución computacional

El algoritmo logra resolver una gran cantidad de anáforas pronominales generados por los siguientes tipos de pronombres:

- Pronombre personal
- Pronombre interrogativo
- Pronombre posesivo
- Pronombre demostrativo
- Pronombre relativo
- Pronombre reflexivo

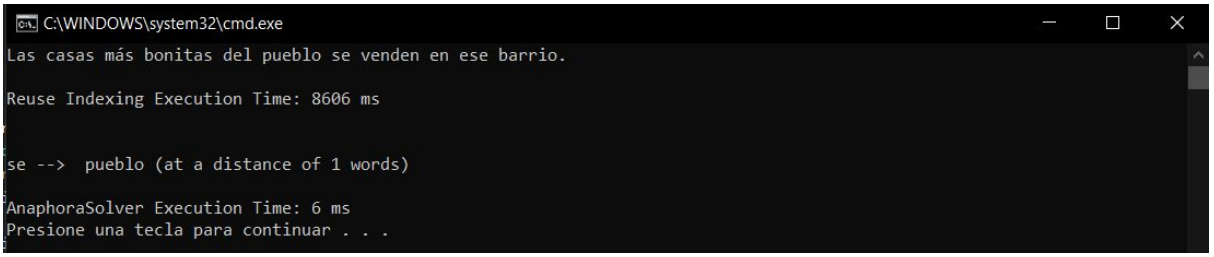
Para probarlo he usado las mismas 100 oraciones que en la solución con IA, para comparar el rendimiento. He marcado en verde las oraciones que el algoritmo resuelve bien y en rojo las que se resuelven mal. Un ejemplo de lo anterior:

Figura 39: Oraciones resueltas por el algoritmo computacional.

Así con las 100 oraciones. Como vemos los fallos ocurren cuando el filtro de relevancia se equivoca o se resuelve por defecto la anáfora al antecedente más cercano. Tomemos como ejemplo alguna de las oraciones de la figura anterior.

“Las casas más bonitas del pueblo se venden en ese barrio”.

El algoritmo arroja el siguiente resultado:



Como “se” es un pronombre reflexivo con género y número invariantes, los antecedentes candidatos son “casas” y “pueblo”. El comportamiento por defecto es resolver al más cercano al pronombre, y por tanto, falla.

“El gato de Pedro es el que me arañó.”

El algoritmo comete una doble equivocación. En primer lugar dice que Pedro es el que araña, en vez del gato. Igual que en el caso anterior, se resuelve mal por cercanía. El segundo error surge cuando se resuelve “me” a “Pedro” cuando en realidad no hay información sobre la entidad que está hablando en tercera persona, y el resultado debería ser “Not found”. Este tipo de oraciones casi siempre se van a resolver de forma errónea.

Tabla 3: Resultados del algoritmo computacional sobre las 100 oraciones. Elaboración propia.

	Oraciones bien resueltas	**Oraciones mal resueltas**
TOTAL	83	17

El porcentaje de acierto está en un 83%. Lógicamente este resultado depende mucho de las oraciones que se introduzcan en el programa. Si analizas 100 frases muy similares que sabes que van a ser resueltas bien, obtendrás un 100% de acierto, pero no es un resultado realista.

En mi caso he intentado que las oraciones sean muy variadas y diferentes unas de otras. En cualquier caso es difícil evitar repetir patrones. Teniendo en cuenta que algunas oraciones las he hecho yo, otras las he tomado de periódicos y otras de la novela "El conde de Montecristo", puedo decir que reflejan en cierta manera tanto el lenguaje coloquial cotidiano, como el lenguaje un poco más formal de la literatura.

No cabe duda de que para saber a ciencia cierta si el algoritmo realmente resuelve un 83% de anáforas, o más, o menos, serían necesarias muchas más pruebas. Un dato que parece darle fiabilidad a estos resultados es que la gran mayoría de algoritmos computacionales clásicos, en los cuales me he inspirado, dan un porcentaje de acierto muy similar, entre el 80% y el 85%.

En cuanto al código, he analizado el mismo usando varias herramientas. Por un lado tenemos las métricas de código automático de Visual Studio.

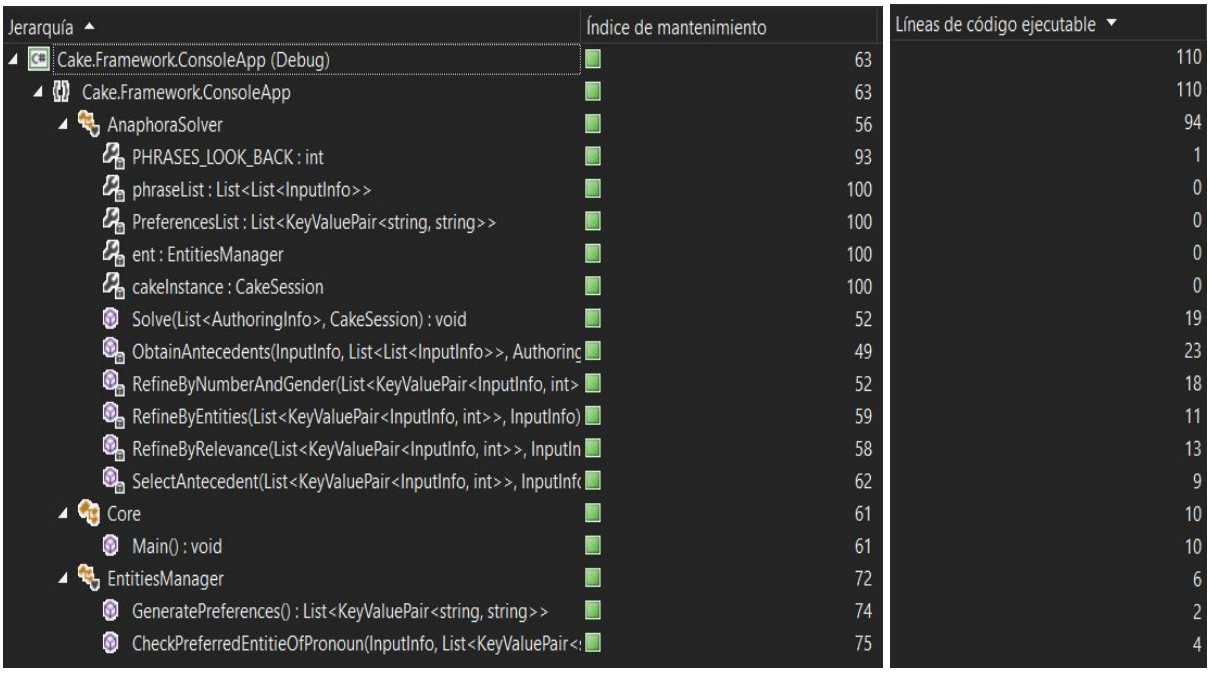


Figura 40: Resultados de las métricas del código con Visual Studio.

He obtenido un buen índice global de mantenimiento, con una media de 63 puntos, y todo el componente son solamente 110 líneas de código ejecutable. Si consultamos la documentación de Microsoft, encontramos lo siguiente:

Índice de mantenimiento : calcula un valor de índice entre 0 y 100 que representa la facilidad relativa de mantenimiento del código. Un valor alto significa un mejor mantenimiento. Las clasificaciones codificadas por colores se pueden usar para identificar rápidamente los puntos problemáticos en el código. Una clasificación verde está entre 20 y 100 e indica que el código tiene buena capacidad de mantenimiento. Una clasificación amarilla está comprendida entre 10 y 19 e indica que el código se mantiene de forma

moderada. Una clasificación de color rojo es una clasificación entre 0 y 9 y indica un mantenimiento bajo.
Por lo tanto es un código bastante eficaz, eficiente y mantenible.

Además de Visual Studio he trabajado con una herramienta externa. He usado SonarQube, que es la que suelo utilizar. Es software que analiza el código fuente de un proyecto y obtiene métricas. Tiene tanto un *plug-in* que se integra con Visual Studio, que es SonarLint, como un escáner local para Windows. Además de eso, tiene una plataforma *cloud* que recoge los resultados de los análisis de los proyectos. No hace falta que subas los mismos, puedes compartirlos mediante tu GitHub, y de esta manera en la web ves las métricas a tiempo real con cada cambio que haces en tu repositorio.

Normalmente subo los proyectos a SonarCloud, pero tiene la desventaja de que se hacen públicos, y no estaba seguro de hasta qué punto tenía permitido hacer eso (puesto que todo esto es funcionalidad software integrada con la herramienta comercial *KM*), así que realicé los análisis y luego lo borré todo:

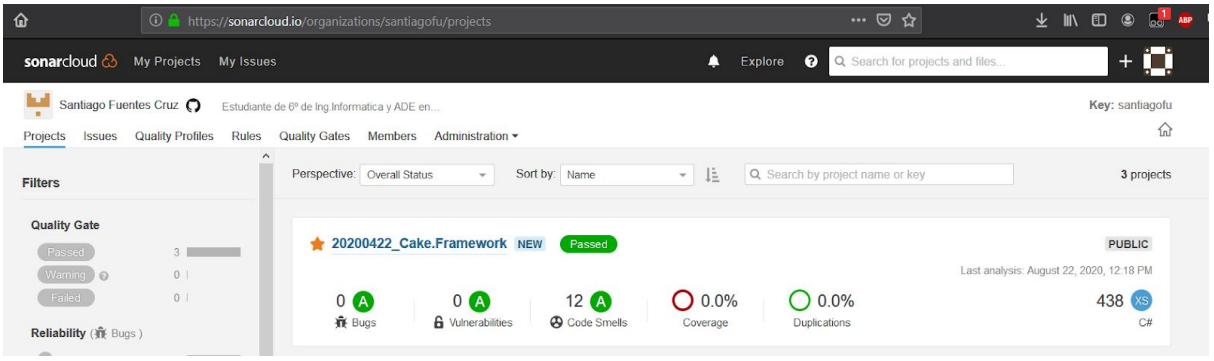


Figura 41: Vista general del análisis en SonarCloud.

La ventaja es que puedes ver exactamente qué partes (según sus métricas) deberías mejorar.

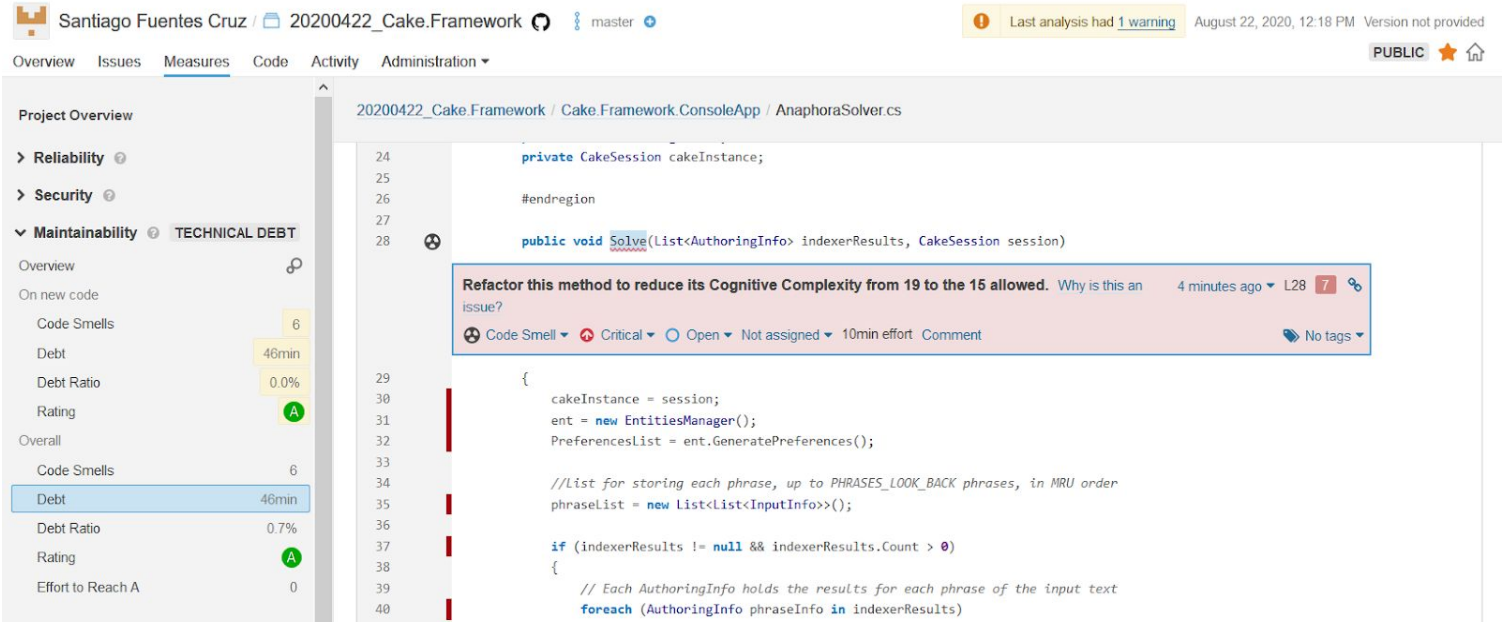


Figura 42: Ejemplo de posibles mejoras a realizar en el código.

En cualquier caso estoy bastante satisfecho porque obtengo una calificación de A en todos los aspectos, y mi deuda técnica asciende a aproximadamente 1 hora.

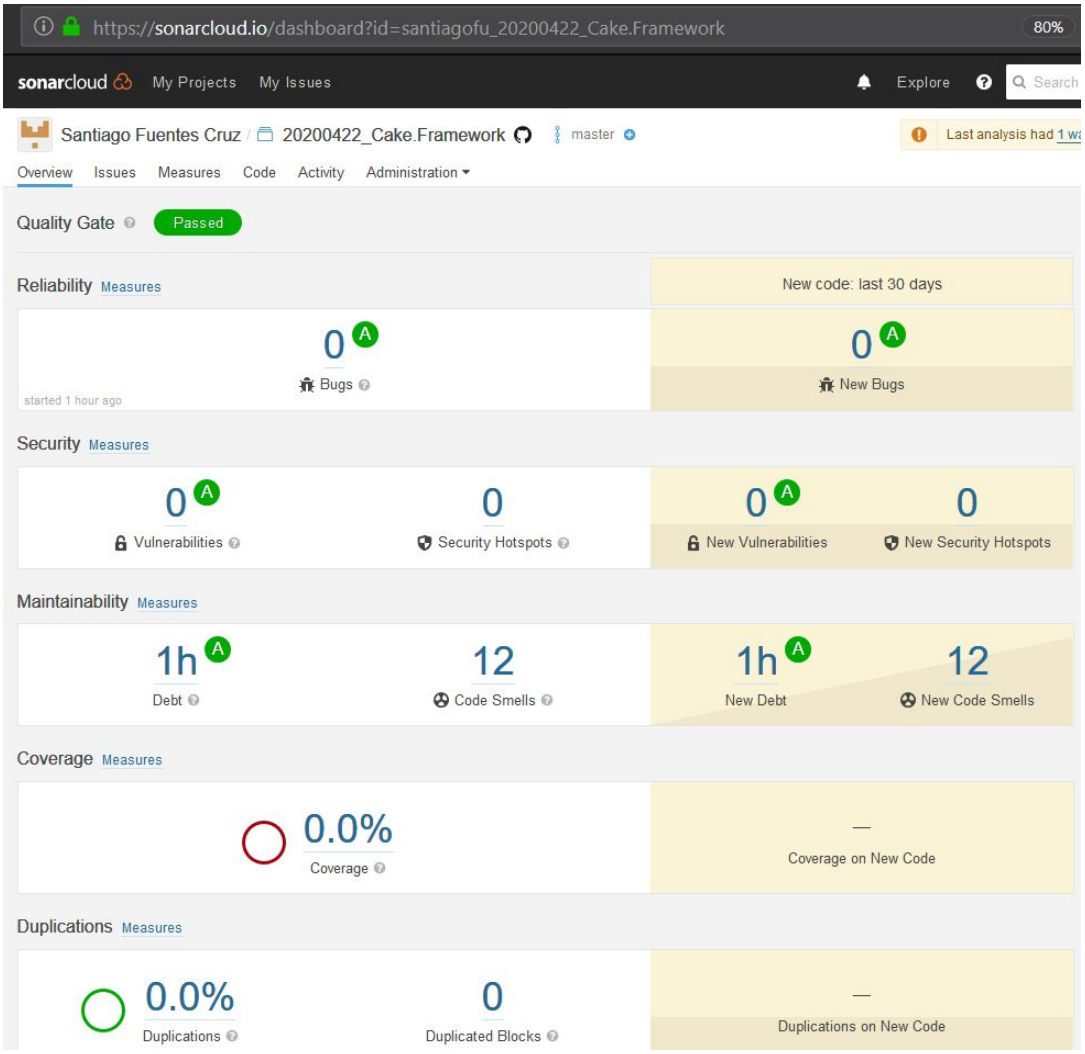


Figura 43: Desglose de los resultados de SonarQube.

En cuanto al ratio de deuda técnica, que se calcula como la deuda técnica dividido por el coste estimado de desarrollo, he obtenido muy buenos resultados:

Figura 44: Desglose del ratio de deuda técnica.

Todos los análisis anteriores se pueden hacer en local con SonarLint como ya comenté antes, si el programador no quiere estar esperando a analizar los resultados en la nube. Es tan simple como instalar el *plug-in*, conectar la cuenta de SonarCloud y de GitHub, ir al código y leer lo que te aconsejan. Por ejemplo:

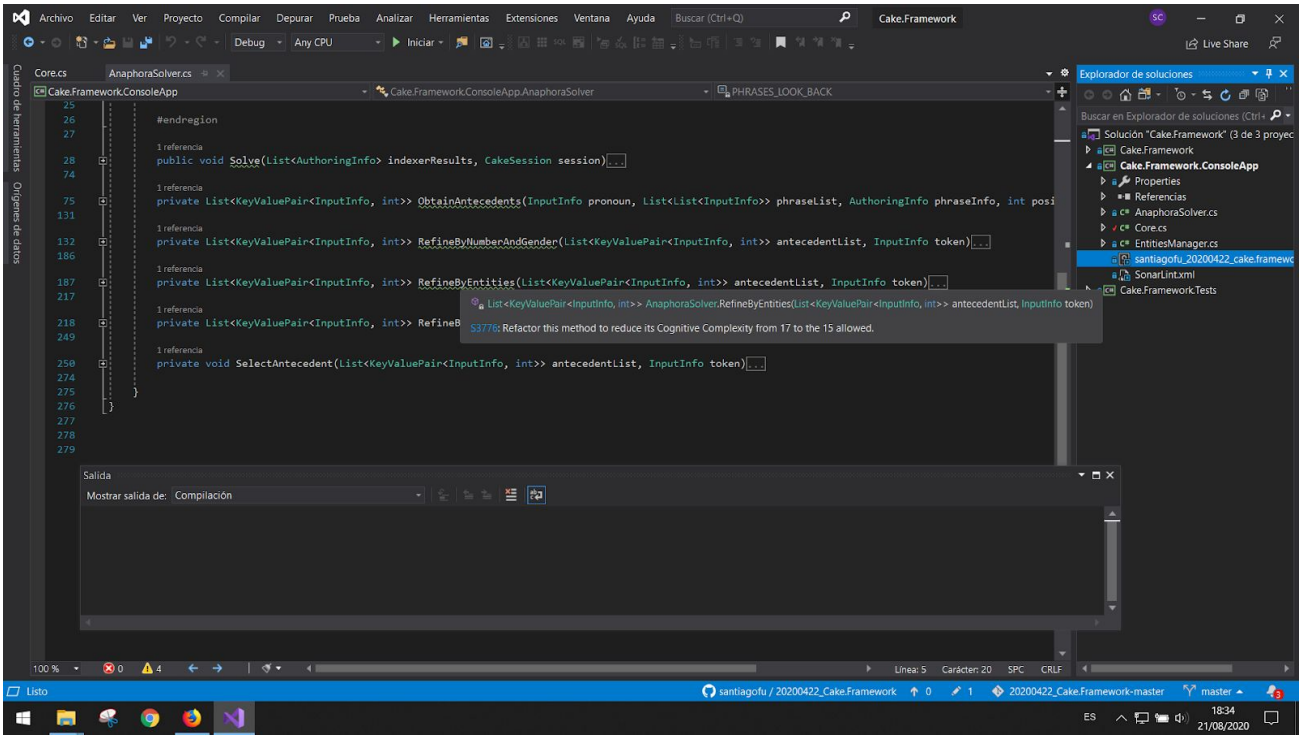


Figura 45: Ejemplo de integración de SonarQube en Visual Studio.

Como se puede ver, es el mismo tipo de análisis que vimos en SonarCloud, pero a tiempo real en tu entorno local. Este tipo de análisis no suelo usarlo mucho salvo en dos escenarios:

1. Si trabajo con estructuras de datos que no conozca bien, para eliminar redundancias y bloques de código mal construidos.
2. Si es un proyecto que necesita interactuar con un servidor (local, microservicios en la nube o lo que sea). SonarQube mantiene una lista actualizada de vulnerabilidades, tanto de OWASP, SANS y CWE. Por tanto prefiero usar SonarLint porque en cuanto escribes una línea de código que coincide con una posible vulnerabilidad, ya sea conectándote a un servidor, enviando consultas a una base de datos, peticiones http get/post o lo que sea, la alerta te sale instantánea. Es mucho mejor que esperarse a tener todo programado, hacer el análisis en *cloud*, y ver que tenías varios errores.

4.3 Resultados de la solución con IA

Los resultados de esta solución los aporta la propia herramienta *Weka*. Dependiendo de los filtros que se apliquen y del clasificador usado, nos arroja porcentajes de éxito diferentes.

Se ha usado validación cruzada con instancias estratificadas. El método en inglés se llama *10 fold cross-validation*. Esto quiere decir que el conjunto de datos de entrenamiento se divide

en 10 partes iguales, usándose 9 para entrenar y 1 para los tests. Con ello se intenta que el conjunto de datos sobre el que se hacen las pruebas sea lo más independiente posible del conjunto de datos con que se ha entrenado.

Por otro lado, el clasificador usado ha sido *PART*. Este clasificador genera reglas a partir de los casos de entrenamiento, y construye un árbol de decisión. Según la propia definición de la documentación:

Se usa para generar una lista de decisiones PART. Utiliza el paradigma de resolución divide y vencerás. Construye un árbol de decisión C4.5 parcial en cada iteración y convierte la mejor hoja en una regla.

Con este clasificador y sin filtros en los atributos, los resultados han sido:

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          90               90      %
Incorrectly Classified Instances        10               10      %
Kappa statistic                          0.8841
Mean absolute error                      0.0028
Root mean squared error                  0.0518
Relative absolute error                 10.9513 %
Root relative squared error             46.4136 %
Total Number of Instances              100
```

Figura 46: Resultados con PART sin filtrado.

Las medidas de precisión obtenidas son las siguientes:

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               1,000    0,010    0,667      1,000    0,800      0,812    0,995     0,667     492WW
               ?        0,000    ?          ?        ?          ?        ?         ?         492WS
               ?        0,000    ?          ?        ?          ?        ?         ?         492WP
               ?        0,000    ?          ?        ?          ?        ?         ?         492FW
               ?        0,000    ?          ?        ?          ?        ?         ?         492MW
               0,846    0,023    0,846      0,846    0,846      0,823    0,994     0,944     492FS
               0,786    0,012    0,917      0,786    0,846      0,827    0,989     0,938     492FP
               0,792    0,039    0,864      0,792    0,826      0,776    0,964     0,883     492MS
               1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     492MP

               1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     548FS
               ?        0,000    ?          ?        ?          ?        ?         ?         548FP
               1,000    0,032    0,667      1,000    0,800      0,803    0,984     0,667     548MS
               ?        0,000    ?          ?        ?          ?        ?         ?         548MP

               1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     547WW

               1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     551MW

Weighted Avg.  0,900    0,016    0,909      0,900    0,901      0,883    0,988     0,929
```

Figura 47: Medidas de precisión con PART sin filtrado.

Las medidas se refieren a las etiquetas que resuelven las anáforas. Como son pronominales, las soluciones se refieren a las etiquetas 492 (sustantivo genérico), 548 (sustantivo del clúster nombre propio de persona), 547 (sustantivo del clúster nombre propio de lugar) y 551 (sustantivo del clúster momento temporal).

Luego tenemos todas las combinaciones de sustantivos con género y número. Por ejemplo:

- 492WW: sustantivo con género y número indefinidos o invariantes.
- 492MS: sustantivo masculino singular.
- 492MP: sustantivo masculino plural.
- 492FS: sustantivo femenino singular.
- 492FP: sustantivo femenino plural.
- 492WS: sustantivo con género indefinido o invariante, singular.
- 492WP: sustantivo con género indefinido o invariante, plural.
- 492MW: sustantivo masculino, número invariante o indefinido.
- 492FW: sustantivo femenino, número invariante o indefinido.

Y similar para las otras etiquetas. A continuación se resumen qué significan las diferentes medidas:

- **TP Rate**: *true positive rate*, positivos verdaderos, son las instancias clasificadas correctamente como pertenecientes a la clase positiva.
  La clase positiva es la que acepta la hipótesis nula. La hipótesis nula en nuestro caso es que una determinada etiqueta es la solución de la anáfora.
  Por ejemplo para los sustantivos femeninos singular, es 0,846 (84,6% de acierto). Esto significa que, de todas las instancias que han aceptado la hipótesis nula de que la anáfora se resuelve mediante un sustantivo femenino singular, un 84,6% están en lo cierto.

  Matemáticamente sigue la siguiente función:

  $$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

  Donde TP es verdaderos positivos y FN es falsos negativos.

- **FP Rate**: *false positive rate*, falsos positivos, es la tasa de instancias clasificadas incorrectamente como pertenecientes a la clase positiva. Es la tasa complementaria de la anterior, que suman el total de casos (100% o lo que es lo mismo, 1).

  $$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$

  Donde TN es verdaderos negativos.

- **Precision**: precisión, también llamado valor predictivo positivo (PPV), es la tasa resultado de dividir el número de verdaderos positivos entre la suma de verdaderos positivos más los falsos positivos.

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

- **Recall**: exhaustividad, es lo mismo que el TP Rate.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

- **F-Measure**: se calcula usando la precisión y el recall. Es la media armónica de ambas medidas. Su valor máximo es 1.

$$F = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

Los mejores resultados se han conseguido cuando la solución de la anáfora es un sustantivo masculino plural así como con los nombres propios de persona y de lugar, en los que vemos que se consigue un 1 (100% de acierto).
Las reglas que ha generado *Weka* son de este tipo:

```
PART decision list
------------------

anafora = 562FS AND
word9 = 478WW: 548FS (7.0)

anafora = 562MP: 492MP (7.0)

word9 = 492FP: 492FP (8.0)

anafora = 562WS AND
word12 = 485FS: 548MS2 (4.0)

anafora = 562WS: 548MS (9.0/3.0)

word9 = 485MS AND
word6 = 478WW: 492MS (6.0)

word9 = 492FS AND
word8 = 485FS: 492FS (3.0)

word9 = 492FS: 492MS (2.0)

word9 = 492MP: 492MP (2.0)

word9 = 492MS2: 492MS2 (2.0)

word9 = 488WW: 492FS (2.0)

word7 = 547WW: 547WW (9.6/1.6)

word12 = 490WW: 492MS (4.6/1.0)

word12 = 478WW AND
word15 = 492MS: 492FS2 (4.0)

word12 = 478WW: 492FP (3.8/0.8)

word12 = 548FS: 492FS (3.0)

word12 = 486WW AND
word8 = 504WW: 492WW (3.0/1.0)

word9 = 492MS: 492MS (6.0/2.0)

word8 = 492MS: 492MP (2.0)

word6 = 492FS: 551MW (2.0)

anafora = 562FS: 548FS (3.0)

word6 = 492MS2: 492MS (2.0)

: 492FS (5.0/2.0)

Number of Rules  :      23


Time taken to build model: 0.21 seconds
```

Figura 48: Reglas generadas por Weka. Sin filtrado previo de atributos.

Se han generado 23 reglas, que son bastante intuitivas. Por ejemplo la primera de todas nos dice que si en la posición 10 (en la que está la anáfora) tenemos un 562FS (pronombre personal femenino singular), y en la posición 9 (la palabra anterior a la anáfora) tenemos un 478 (preposición), la palabra que resuelve la anáfora será un 548FS (nombre propio de persona femenino singular). Y esa regla se cumple para 7 instancias.

Otras reglas tienen también los casos en los que no se cumple la regla. Por ejemplo en la quinta, vemos que entre paréntesis pone "(9.0/3.0)" que significa que de 9 instancias que siguen esa regla, no se cumple en 3.

Ahora bien, estos resultados pueden mejorarse aplicando filtros que preprocesen los datos de entrada. De todos los que hay, el que mejor me ha funcionado es el filtro *ClassConditionalProbabilities.*

Si acudimos a la documentación, vemos que lo que hace es:

Convierte los valores de atributos nominales y / o numéricos en probabilidades condicionales de clase. Si hay k clases, entonces se crean k nuevos atributos para cada uno de los originales. Puede ser útil para convertir atributos nominales con muchos valores distintos en algo más manejable.

Lo que hace es generar *k* nuevos atributos, resultado de aplicar probabilidades condicionadas de todos con todos.

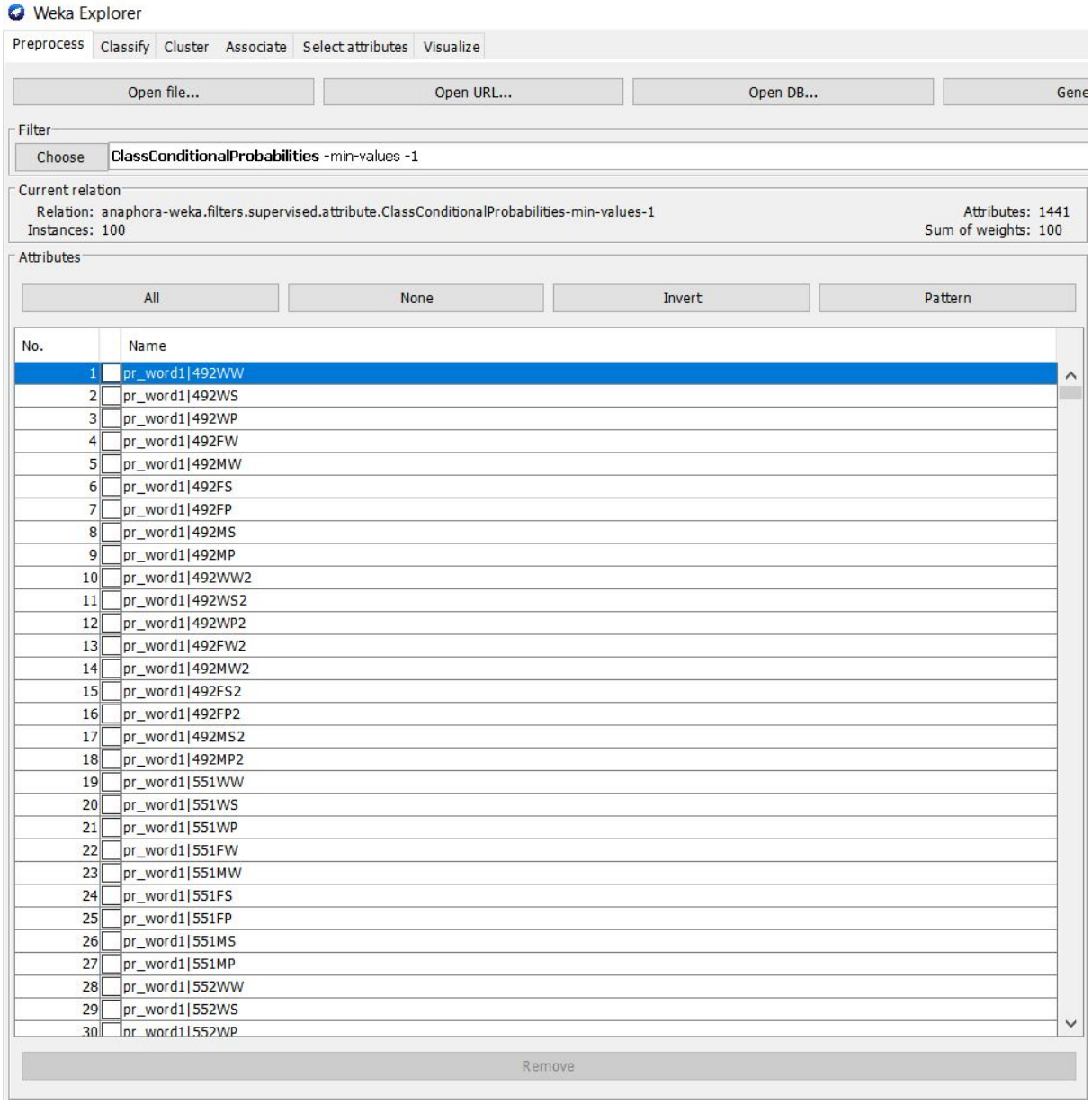


Figura 49: Preprocesado de atributos con el filtro ClassConditionalProbabilities.

Cuando volvemos a ejecutar el clasificador *PART*, obtenemos mejores resultados.

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          97               97      %
Incorrectly Classified Instances         3                3      %
Kappa statistic                          0.9656
Mean absolute error                      0.0009
Root mean squared error                  0.0273
Relative absolute error                  3.6219 %
Root relative squared error             24.7351 %
Total Number of Instances              100
```

Figura 50: Resultados con PART usando filtrado.

Se incrementa el acierto hasta el 97%, fallando solo 3 instancias. Si analizamos las medidas de precisión, nos encontramos con que también son mejores:

	0,923	0,000	1,000	0,923	0,960	0,955	0,991	0,955	492FS
	0,929	0,000	1,000	0,929	0,963	0,958	0,997	0,974	492FP
	0,958	0,013	0,958	0,958	0,958	0,945	0,991	0,953	492MS
	1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	492MP
	1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	492FS2
	?	0,000	?	?	?	?	?	?	492FP2
	1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	492MS2
	1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	551MW
	1,000	0,011	0,889	1,000	0,941	0,938	0,995	0,889	547WW
	1,000	0,011	0,909	1,000	0,952	0,948	0,994	0,909	548FS
	?	0,000	?	?	?	?	?	?	548FP
	1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	548MS
	?	0,000	?	?	?	?	?	?	548MP
Weighted Avg.	0,970	0,005	0,972	0,970	0,970	0,965	0,995	0,961	

Figura 51: Medidas de precisión con PART con filtrado.

En cuanto a las reglas generadas, son mucho más completas.

```
pr_word9|492WW > 0.004854 AND
pr_word5|492FS > 0.009479: 547WW (9.0/1.0)

pr_word9|548FS > 0.004673 AND
pr_word4|547WW <= 0.009434: 548FS (11.0/1.0)

pr_anafora|548MS > 0.014493 AND
pr_word5|551MW <= 0.004854 AND
pr_word4|547WW <= 0.009434: 548MS (6.0)

pr_word9|492WW > 0.004854 AND
pr_word4|492WW <= 0.004854: 551MW (2.0)

pr_word9|548FS <= 0.009346 AND
pr_anafora|492FS2 > 0.014925 AND
pr_word7|492FS <= 0.009346: 492FS2 (4.0)

pr_word9|548FS <= 0.009346 AND
pr_word9|492FS > 0.004608 AND
pr_word12|492MS > 0.004386 AND
pr_word11|492MS > 0.013158 AND
pr_word13|492MS > 0.004425: 492MS (18.79/1.0)

pr_word8|548MS2 <= 0.004808 AND
pr_word9|492FS > 0.004608 AND
pr_anafora|492MP <= 0.013514: 492FS (12.21/0.21)

pr_word8|548MS2 <= 0.004808 AND
pr_word8|492WW <= 0.004854 AND
pr_word8|492MS2 <= 0.004854 AND
pr_anafora|492FP <= 0.038961 AND
pr_word9|492FS <= 0.004608 AND
pr_word8|548MS <= 0.004762 AND
pr_anafora|548MS <= 0.014493: 492MP (11.0)

pr_word8|548MS2 <= 0.004808 AND
pr_word7|492WW <= 0.004854 AND
pr_word8|492MS2 <= 0.004854 AND
pr_word7|492MS <= 0.004545: 492FP (13.0)

pr_word7|492FP <= 0.004587 AND
pr_word7|492WW <= 0.004854 AND
pr_word8|492MS2 <= 0.004854: 492MS (5.0)

pr_word5|548MS2 > 0.004808: 548MS2 (4.0)

pr_word5|492WW <= 0.004854: 492MS2 (2.0)

: 492WW (2.0)

Number of Rules  :      13


Time taken to build model: 0.45 seconds
```

Figura 52: Reglas generadas por Weka. Con filtrado previo de atributos.

Las reglas se basan en probabilidades condicionadas de ahí que las reglas expresen justamente esas relaciones.

Son resultados bastante buenos pero hay que cogerlos con cuidado porque el modelo de entrenamiento contiene apenas 100 instancias. Para obtener resultados más claros habría que entrenarlo con muchas oraciones más. Desafortunadamente no hay, al menos que conozca, corpus resueltos de anáforas en español accesibles desde Internet, así que queda para trabajos futuros el crear o encontrar un corpus de buen tamaño y probar de nuevo este enfoque.

5. MARCO REGULADOR

Si bien es cierto que a día de hoy no contamos con un marco regulador como tal, me gustaría destacar que la Comisión Europea ha publicado en febrero de 2020 un Libro Blanco sobre Inteligencia Artificial.

Antes de enunciar las ideas más relevantes que contiene este documento, quisiera definir algunos conceptos:

5.1 Definiciones

Comisión Europea

Es una institución europea independiente de los Gobiernos de los Estados Miembros. Representa los intereses de la Unión Europea (en adelante UE). Se compone de los comisarios más el Presidente.

- Comisarios: son designados por el Consejo Europeo, actualmente 1 comisario por Estado Miembro. El de España es Josep Borrell, que posteriormente fue propuesto y elegido como Alto Representante de la Unión Europea.
- Presidente: define las orientaciones así como la organización interna de la Comisión y realiza el nombramiento de vicepresidentes. Actualmente es Ursula von der Leyen, desde diciembre de 2019.

Sus competencias son: tomar la iniciativa acerca de proponer nueva legislación, control y supervisión del cumplimiento del derecho en la UE. Representa a la UE en el exterior (junto al Alto Representante) y negocia acuerdos internacionales de gestión y asignación del presupuesto de la Unión Europea.

Libro Verde

Es un informe gubernamental provisional mediante el cual se elaboran propuestas de políticas de cualquier tipo, para su debate y discusión. No compromete al gobierno, y deja abierta su decisión final hasta que haya podido considerar la reacción pública al respecto. Pueden dar lugar posteriormente a un Libro Blanco.

Libro Blanco

Es un documento emitido por un Gobierno, que presenta las preferencias políticas gubernamentales antes de que cierta legislación sea introducida. Sirve para poner a prueba la opinión pública sobre cuestiones controvertidas y ayudan al Gobierno a evaluar el impacto que tendrá la legislación. Al contrario que los Libros Verdes, los Blancos son mucho menos abiertos, y detallan más en profundidad las estrategias que se piensan seguir. Suelen ser más técnicos en su contenido.

A continuación expongo el marco regulador (futuro) que he encontrado sobre la IA.

5.2 Libro Blanco sobre Inteligencia Artificial

Este documento fue publicado por la Comisión Europea el 19 de febrero de 2020. A continuación resumo las ideas principales que contiene dicho documento:

La inteligencia artificial está cambiando y cambiará nuestras vidas, en ámbitos como la salud, la agricultura, mitigar el cambio climático, la seguridad y muchas otras facetas.

La Unión Europea aspira a convertirse en líder mundial de la innovación en la economía de los datos y sus aplicaciones, y por tanto es necesario un marco regulador para conseguirlo, así como para dar más seguridad a los ciudadanos.

Los pilares fundamentales que sigue el Libro Blanco son dos:

1. Un marco político por el que se establecen medidas para armonizar los esfuerzos a escala regional, nacional y europea. En colaboración con los sectores público y privado, los objetivos del marco son movilizar recursos para obtener un **«ecosistema de excelencia»** a lo largo de toda la cadena de valor, partiendo de la investigación y la innovación, así como crear los incentivos adecuados para acelerar la adopción de soluciones basadas en la inteligencia artificial, también por parte de las pequeñas y medianas empresas (pymes).

2. Los elementos clave de un futuro marco normativo para la inteligencia artificial en Europa que generen un **«ecosistema de confianza»** exclusivo. Para hacerlo, este marco debe velar por el cumplimiento de las normas de la UE, especialmente las normas de protección de los derechos fundamentales y los derechos de los consumidores, y en concreto con relación a los sistemas de inteligencia artificial que operan en la UE y presentan un riesgo elevado.

El documento continúa argumentando que a día de hoy la Unión Europea tiene una gran cantidad de datos que se están infrautilizando, y que podrían usarse para mejorar nuestras vidas. Europa cuenta con excelentes centros de investigación y empresas emergentes innovadoras, y es líder en sectores como la robótica. Se hace necesario pues invertir en tecnologías e infraestructuras de la siguiente generación, así como en competencias digitales como la alfabetización sobre datos, reforzando la soberanía tecnológica de Europa en el sector de las tecnologías y las infraestructuras clave para dinamizar la economía de los datos.

A lo largo de los últimos tres años, la financiación para investigación e innovación en inteligencia artificial ha aumentado en 1.500 millones de euros, es decir, un incremento del 70% en comparación con el período anterior. Sin embargo solo alcanzamos un total de

inversión (datos de 2016) en torno a los 3.200 millones de euros, muy lejos de potencias como EEUU que invirtió 12.100 millones de euros.

Un punto de inflexión que se debe aprovechar es la siguiente oleada de datos. Actualmente (datos de 2018), se producen 33 zetabytes de información anuales, y se estima que en 2025 aumente hasta 175 zetabytes. Un zetabyte o zettabyte es una unidad de almacenamiento de información, y equivale a 10^{21} bytes.

Los avances recientes en computación cuántica generarán aumentos exponenciales en la capacidad de tratamiento de estos datos. Estos avances se materializan en una sólida fortaleza académica, simuladores cuánticos y entornos de programación.

En cuanto a los dos pilares mencionados antes, nos definen el ecosistema **de excelencia** como una serie de acciones conjuntas para conseguir que la cooperación entre Estados Miembros lleven a la eficiencia, tanto de las inversiones económicas como de los resultados obtenidos. Para ello se ha elaborado un Plan que durará hasta 2027. El objetivo es atraer una inversión anual de 20.000 millones de euros para IA durante la próxima década. Además será necesario que la IA pueda por sí misma analizar de forma crítica el uso de recursos y consumo de energía, para ser respetuosa con el medio ambiente. Otras medidas que nos conducirán al ecosistema de excelencia son:

- No fragmentar los centros de competencia, sino lograr sinergias y redes entre los centros de investigación sobre la IA.
- Mejorar los planes educativos actuales, orientándolos a la era digital, e incrementar la concienciación sobre la IA, para que los ciudadanos entiendan y tomen decisiones conociendo la influencia que esta tendrá en nuestras vidas.
- Mejorar las habilidades necesarias de los estudiantes universitarios para trabajar profesionalmente en el ámbito de la IA, así como desarrollar planes formativos para los trabajadores actuales.
- Preocuparse por las pymes, para que tengan acceso a la financiación necesaria para adaptarse.
- Asociarse con el sector privado.
- Promover la adopción de la IA en el sector público.

En cuanto al **ecosistema de confianza,** el documento nos habla sobre las amenazas que la IA traerá a los ciudadanos. Estos temen quedar indefensos a la hora de proteger sus derechos y su seguridad frente a las decisiones automatizadas que realicen los algoritmos de la IA.

Respecto a eso, la Unión Europea ha elaborado una lista de requisitos que deben cumplir estos sistemas. Estos son:

1. Acción y supervisión humanas.
2. Solidez técnica y seguridad.

3. Gestión de la privacidad y de los datos.
4. Transparencia
5. Equidad.
6. Bienestar social y medioambiental.
7. Rendición de cuentas.

Los desarrolladores e implementadores de la inteligencia artificial ya están sujetos a la legislación europea respecto a derechos fundamentales (protección de datos, privacidad, no discriminación, entre otros). Sin embargo la opacidad hace que sea complejo regularla. Esta opacidad se manifiesta en que la gran mayoría de la población desconoce cómo funcionan sus algoritmos. Es necesario esforzarse en lograr mayor transparencia.

Un problema importante es la ausencia de un marco común europeo. Alemania por ejemplo propone un sistema de regulación basado en 5 niveles basados en el riesgo, Malta un sistema voluntario de certificación, y Dinamarca ha puesto en marcha un prototipo de sello de ética de los datos (el documento no explica qué es esto).

Continúa elaborando los posibles efectos nocivos de la IA. Por ejemplo, si entrenamos un sistema con datos principalmente relativos a hombres, se puede traducir en peores resultados en relación con las mujeres. Es necesario una supervisión o corrección previas, para evitar malos diseños. También representa un peligro el posible uso de la IA para la vigilancia masiva de la población, siguiendo y analizando sus quehaceres cotidianos, vulnerando la legislación de protección de datos, y otros derechos fundamentales como la privacidad y la intimidad. Otro ejemplo sería la automatización de toma de decisiones respecto a predecir la reincidencia delictiva de una persona, ya que la IA puede acabar con prejuicios raciales o de género (Tolan S., Miron M., 2019).

Lo que se argumenta es que los algoritmos que acaben con sesgos de comportamiento basados en la subjetividad con que fueron programados, pueden acabar tomando decisiones complejas e imprevisibles.

Otro gran problema es la responsabilidad civil. Un fabricante es responsable de los daños causados por un producto defectuoso. No obstante, en el caso de un sistema basado en la IA, como un vehículo autónomo, puede resultar difícil demostrar la existencia de un defecto en el producto, el daño que este ha generado y el nexo causal entre ambos.

El documento finaliza exponiendo cuáles podrían ser las características de un futuro marco regulador:

- Distinguir las partes que conforman la IA, y legislar sobre ellas. Se hace una separación clara entre los datos y el algoritmo.

- Establecer niveles de riesgo. Por ejemplo una IA de alto riesgo es aquella que se emplea en un sector con riesgos significativos (no aclara qué riesgos son estos aunque intuyo que pueden ser sectores como la sanidad).
- Transparencia, supervisión humana y otras medidas para las IA de alto riesgo.
- La regulación legal afectará al desarrollador de la IA, así como al implementador, estén o no establecidos en la Unión Europea.
- Evaluaciones y etiquetados de los sistemas de IA, mostrando su riesgo. Si este es bajo será opcional.
- Crear una estructura de gobernanza que garantice la mayor participación de todas las partes interesadas.

6. MARCO SOCIO-ECONÓMICO

A nivel socioeconómico, existen dos factores principales a tener en cuenta:

- Impacto actual de la IA en la sociedad, áreas que afecta, especialmente la economía. Datos económicos actuales nacionales y europeos.
- Impacto futuro. Multitud de iniciativas, planes y estrategias tanto nacionales como internacionales que nos afectan. Parte del impacto futuro se ha descrito en el apartado anterior mediante el Libro Blanco sobre IA.

6.1 Impacto actual de la IA en la sociedad

Uno de los informes más relevantes sobre el tema es la publicación en 2018 de "*El impacto de la inteligencia artificial en el aprendizaje, la enseñanza y la educación"*. Lo elaboró la Comisión Europea y describe qué es la inteligencia artificial, desarrollos actuales y futuros, y su impacto en el área del aprendizaje, enseñanza y educación. Resumiéndolo, nos explica que la IA va a cambiar el modelo educativo de manera notable, y nos pone ejemplos como las mejoras en educación especial para personas con dislexia, hiperactividad o déficit de atención, diagnosticando de manera temprana esos comportamientos mediante el rastreo de sus patrones oculares en clase. Otras aplicaciones son la generación de predicciones en las calificaciones de los alumnos, de tal manera que se pueden aplicar metodologías específicas para cada uno según su progreso presente y futuro.

En cuanto a otras ramas como el desarrollo económico, principalmente existen dos tipos de estudios:

- De qué manera la IA mejora la economía. En general coinciden en que las aportaciones de la IA van a ser o están siendo ya muy significativas, en lo referente a innovación (Henderson, 2018), *big data* (Allam, 2019), Medicina (Prater, 2018), Economía (Jones, 2017) y muchas más áreas como el internet de las cosas, ciudades inteligentes...

- De qué manera la IA empeora la economía, especialmente en lo referente a los puestos de trabajo y el mercado laboral (Agrawal, 2019).

Si resumimos las ideas principales de todos los autores anteriores, nos encontramos con lo siguiente:

- Ventajas:
    - La innovación se acelera puesto que podemos combinar nuestros conocimientos con algoritmos avanzados de predicción.

    - El uso de la IA aplicada al big data permite analizar millones de fuentes de información por ejemplo en las redes sociales, para detectar emociones, patrones de conducta, cambios de ideología política en la gente, opiniones públicas sobre cierto acontecimiento, predecir comportamientos y mucho más. En este punto se hace hincapié en la necesidad de un decálogo de ética y respeto de la legalidad.
    - La inversión en IA crece en las grandes potencias económicas mundiales año tras año. Lidera la carrera EEUU.
    - Aplicación de la IA a campos como la educación.
    - Uso de la IA para generar modelos aplicables a ciudades inteligentes e internet de las cosas.

- Inconvenientes:
    - El desarrollo de la IA va a alterar el mercado laboral en varias áreas, especialmente las dedicadas a la predicción de sucesos. Nos pone como ejemplo todos los puestos de trabajo que dedican capital humano a predecir fenómenos atmosféricos y terrestres, desde tormentas hasta terremotos. Es algo que se puede automatizar y no se necesitan tantas personas, sino más máquinas. También en trabajos como las áreas de recursos humanos, donde el encargado selecciona (predice) cuales pueden ser los mejores candidatos para un puesto, en base a su perfil, currículum etc, y que pueden ser realizados por IA. Incluso en áreas tan delicadas como la medicina, determinados procesos como la cirugía pueden ser mejor efectuados por máquinas. Nos pone como ejemplo a un cirujano que opera de cáncer cerebral a un paciente, y para estar seguro de que elimina todo el tumor, acaba quitando más materia cerebral de la necesaria.

6.2 Impacto futuro de la IA en la sociedad

Se traduce en la profusión de planes, estrategias, acuerdos y un largo etcétera que realizan las empresas privadas, el sector público, y los organismos internacionales.

En el contexto que nos afecta, hablaré de España y de Europa.

España

Uno de los pasos más destacados que ha dado España ha sido crear una Secretaría de Estado de Digitalización e Inteligencia Artificial. Si bien es cierto que el tema de la digitalización lleva muchos años en el sector público, es la primera vez que se crea una secretaría específica para la inteligencia artificial. Pertenece al Ministerio de Asuntos Económicos y Transformación Digital. El BOE que crea esta secretaría es de enero de 2020, tras formarse el gobierno de coalición de PSOE-Unidas Podemos a inicios de año. El nombre de la secretaría

es toda una declaración de intenciones sobre la importancia que el sector público prevé que tenga la IA.

Relacionado con lo anterior, recientemente se ha publicado el Plan España Digital 2025 que le dedica un apartado entero a la "Economía del Dato" y a la "Inteligencia Artificial":

- España apoyará la creación de un ecosistema de excelencia para la economía del dato y la inteligencia artificial (esto ya lo vimos en el Libro Blanco sobre inteligencia artificial de la Comisión Europea).
- Elaborar una Estrategia Nacional de Inteligencia Artificial así como continuar con la Estrategia Española de I+D+i en Inteligencia Artificial creada en 2019.
- Convertir a España en un referente en la transformación hacia una economía del dato.
- Impulsar la Inteligencia Artificial como motor de innovación y crecimiento económico social, inclusivo y sostenible.
- Desarrollar un marco ético y jurídico para la IA basado en valores compartidos.
- Preparar a España para las transformaciones socioeconómicas que origina la IA.

Todo lo anterior se traduce en 4 medidas estrella a desarrollar:

1. El Gobierno elaborará una Estrategia Nacional de Inteligencia Artificial con revisiones periódicas.
2. El Gobierno creará en la Administración General del Estado la Oficina del Dato, para aprovechar la amplia experiencia existente en la Administración y optimizar el uso de recursos. Al frente estará un *Chief Data Officer* (CDO) que será responsable de garantizar una buena gobernanza en el uso de los datos públicos. La Oficina del Dato se encargará de diseñar y proponer estrategias que permitan poner a disposición de empresas y particulares los datos de las Administraciones.
3. Creación de un Consejo Asesor de Inteligencia Artificial, formado por expertos de reconocido prestigio en el ámbito de la IA que asesorarán al Gobierno.
4. Elaboración de una Estrategia *Cloud*: espacios compartidos europeos del dato. Sigue la línea que comenzó la Comisión Europea cuando lanzó la iniciativa *European Cloud Federation*. España concretamente potenciará un espacio ibérico conjuntamente con Portugal para impulsar tecnologías avanzadas de computación de datos.

Todas las iniciativas relativas a la IA que están surgiendo en España por parte del sector público, tienen su origen en los diferentes objetivos tecnológicos derivados de la Agenda 2030, compuesto por 17 Objetivos de Desarrollo Sostenible, en los que la IA se puede aplicar en varios de ellos, así como en las diferentes publicaciones europeas.

Lógicamente todo lo que se viene necesitará de estructuras (principalmente de gobernanza) que controlen el funcionamiento. Un ejemplo es el anuncio por parte del Gobierno en julio de 2020 de que se creará un Consejo Consultivo para la Transformación Digital.

Europa

En 2018 se presentó por primera vez un Plan sobre Inteligencia Artificial realizado por la Comisión Europea.

La Comisión Europea presentó en 2019 el Programa Europa Digital 2021-2027. Si acudimos al segundo objetivo específico, nos encontramos que se llama "Inteligencia Artificial", y sus medidas son:

- Intensificar y reforzar las capacidades básicas de inteligencia artificial en la Unión, incluidos los recursos de datos y las bibliotecas de algoritmos de conformidad con la legislación sobre protección de datos.
- Hacer accesibles dichas capacidades a todas las empresas y administraciones públicas.
- Reforzar y poner en red las instalaciones de ensayo y experimentación de inteligencia artificial existentes en los Estados miembros.

A ello añadimos la creciente inversión económica prevista en IA, de la que ya se habla en el Libro Blanco sobre la IA.

7. PLANIFICACIÓN Y PRESUPUESTO

7.1 Planificación temporal

La planificación temporal ha sido compleja, debido a que estamos en una situación excepcional por motivo de la pandemia causada por la COVID-19. En un primer momento la planificación iba a ser realizar el proyecto desde enero hasta junio pero no pudo ser puesto que . Pasado lo más duro de la ola de contagios, Reuse me facilita el entorno de trabajo a mediados de junio.

Para planificarme he usado 2 herramientas muy conocidas, y metodología ágil.

7.1.1 Trello

Esta herramienta consiste en una plataforma web que ofrece al usuario un tablero *kanban* online. De esta manera puedes crear tus propias tarjetas y asignarles actividades. El término *kanban* significa valla publicitaria en japonés. Usualmente tenemos 3 contenedores donde se pueden colocar tarjetas:

1. *To do*: aquí se colocan las actividades que aún no has empezado. Puedes asignarles etiquetas, como si son más o menos urgentes, obligatorias u opcionales.
2. *Doing*: aquí se colocan las actividades que estás haciendo.
3. *Done*: aquí se colocan las actividades que has finalizado.

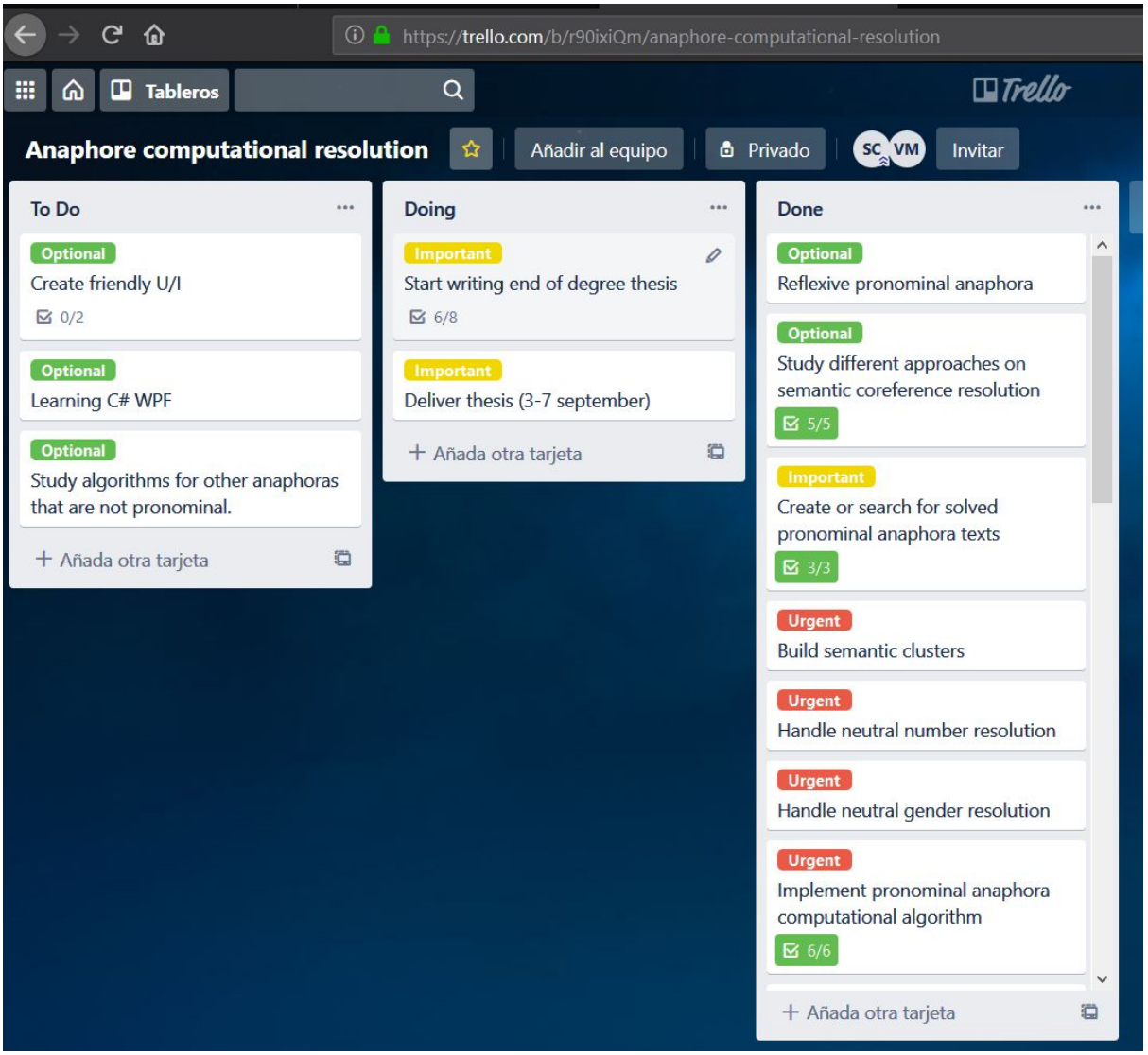


Figura 53: Tablero de Trello.

A continuación expongo las diferentes actividades que he realizado, en orden:

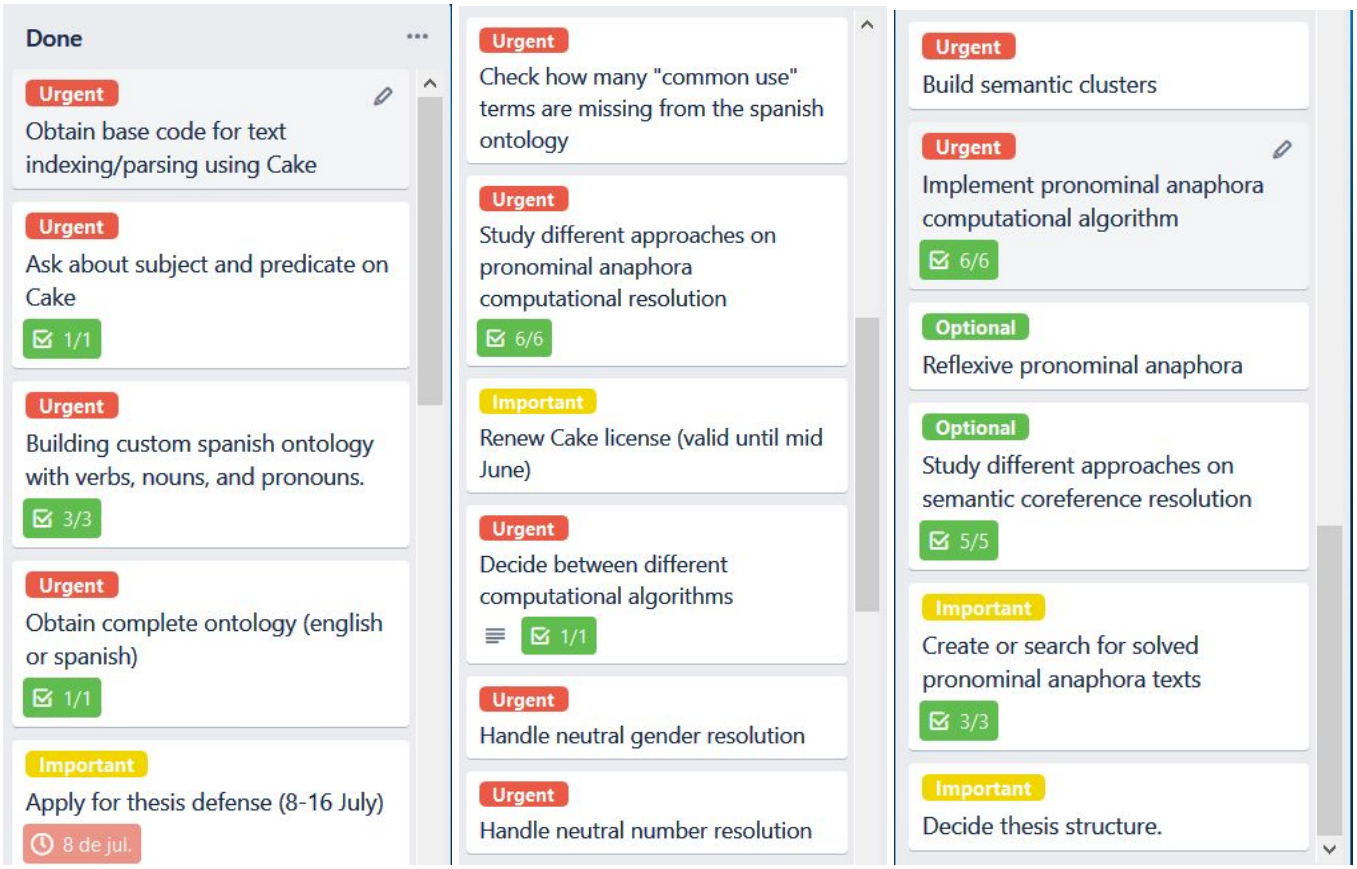


Figura 54: Lista de tareas finalizadas.

Algunas actividades tienen tareas, por ejemplo esta tiene 6:

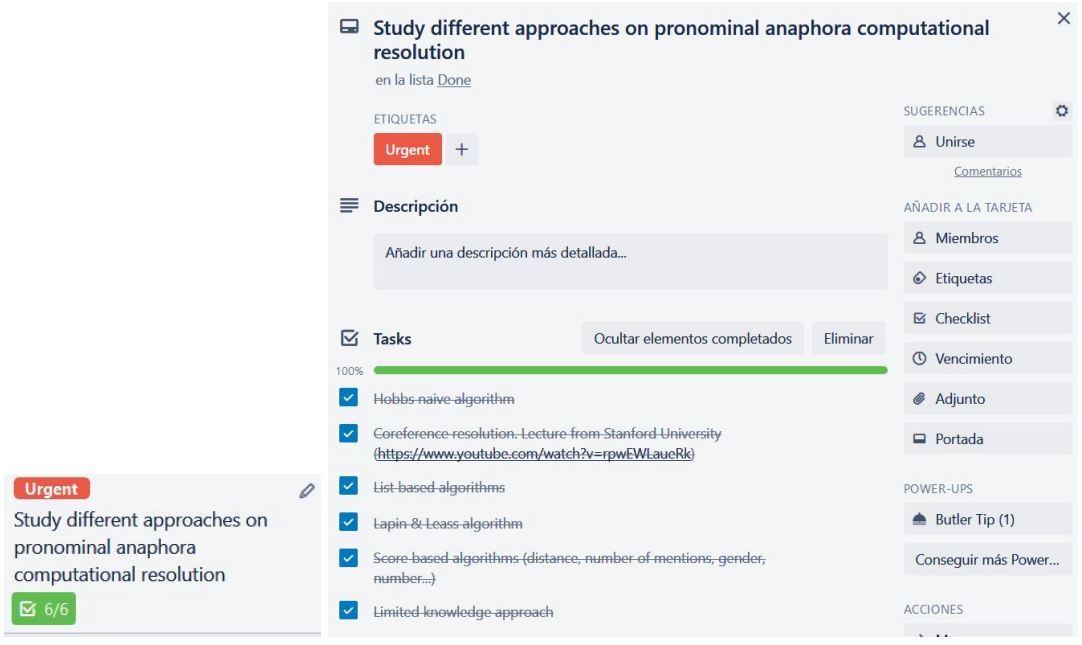


Figura 55: Lista de tareas de una actividad.

A continuación expongo las diferentes actividades que se están realizando, en orden:

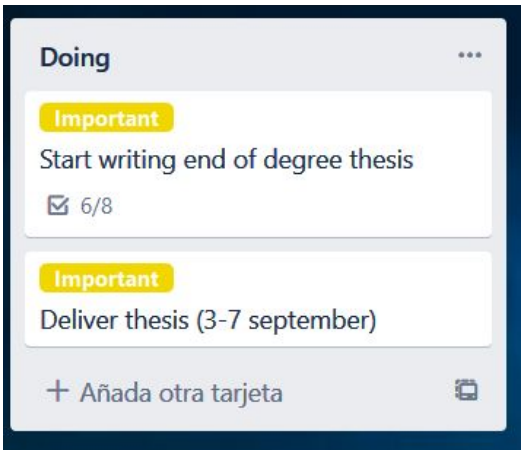


Figura 56: Lista de tareas en proceso.

Y por último, las tareas que están por hacer pero que no realizaré (las marqué como opcionales). Me planteé hacer una interfaz gráfica para el algoritmo computacional, pero al fin y al cabo, este algoritmo formará parte de alguna biblioteca de clases o similar, porque para aplicación gráfica ya está el propio *KM*. Así que no tenía mucho sentido realizarla. La otra actividad que no he realizado es buscar algoritmos o investigar cómo resolver anáforas que no sean pronominales, ya que el proceso será muy similar, solo que en vez de resolver hacia sustantivos, se hará hacia adverbios o lo que toque.

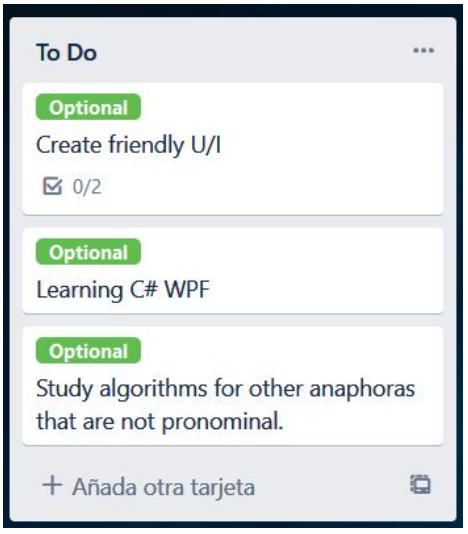


Figura 57: Lista de tareas por hacer.

Por último comentar que una funcionalidad muy interesante que tiene Trello es disponer de un historial de actividad en el que puedes consultar todos los cambios y quiénes los han realizado:

Figura 58: Historial de actividades de Trello.

7.1.2 Tom's Planner

Esta herramienta nos permite realizar de forma online un diagrama de Gantt. Este diagrama sirve para exponer el tiempo que vas a dedicar a cada actividad y a sus tareas. Lo que no se indica es la relación que pueda existir entre ellas.

En cuanto a las fechas, tengo las siguientes:

Actividad	Persona	Estado	Inicio	Final
Creación de la base de datos			**24-06-20**	**30-06-20**
Obtener términos de Wiktionary	Santiago	Finalizado	24-06-20	25-06-20
Limpiar términos	Santiago	Finalizado	26-06-20	26-06-20
Construir términos femeninos y plurales	Santiago	Finalizado	29-06-20	29-06-20
Introducir términos en la base de datos	Santiago	Finalizado	30-06-20	30-06-20
Fin creación de base de datos			30-06-20	30-06-20
Solución computacional			**01-07-20**	**20-07-20**
Estudio de alternativas	Santiago	Finalizado	01-07-20	07-07-20
Implementación algoritmo	Santiago	Finalizado	08-07-20	16-07-20
Pruebas	Santiago	Finalizado	17-07-20	20-07-20
Fin solución computacional			20-07-20	20-07-20
Solución con IA			**07-08-20**	**19-08-20**
Estudio de Weka	Santiago	Finalizado	07-08-20	11-08-20
Construcción de oraciones resueltas	Santiago	Finalizado	12-08-20	17-08-20
Construcción del modelo	Santiago	Finalizado	18-08-20	18-08-20
Pruebas	Santiago	Finalizado	19-08-20	19-08-20
Fin solución con IA			19-08-20	19-08-20
Resultados			**20-08-20**	**25-08-20**
▾ **Análisis cuantitativo**	Santiago	Finalizado	20-08-20	21-08-20
Análisis de términos introducidos en DB	Santiago	Finalizado	20-08-20	20-08-20
Porcentajes de acierto computacionales y de IA	Santiago	Finalizado	21-08-20	21-08-20
▾ **Análisis cualititativo**	Santiago	Finalizado	24-08-20	25-08-20
Calidad de los resultados de ambas soluciones	Santiago	Finalizado	24-08-20	24-08-20
Métricas del código computacional	Santiago	Finalizado	25-08-20	25-08-20
Fin análisis de los resultados			25-08-20	25-08-20
Documentación (TFG)			**06-08-20**	**03-09-20**
Escribir el TFG	Santiago	En proceso	06-08-20	28-08-20
Entrega del TFG	Santiago	Pendiente	03-09-20	03-09-20
Fin documentación			03-09-20	03-09-20

Figura 59: Desglose de fechas con Tom 's Planner.

Y el diagrama queda así:

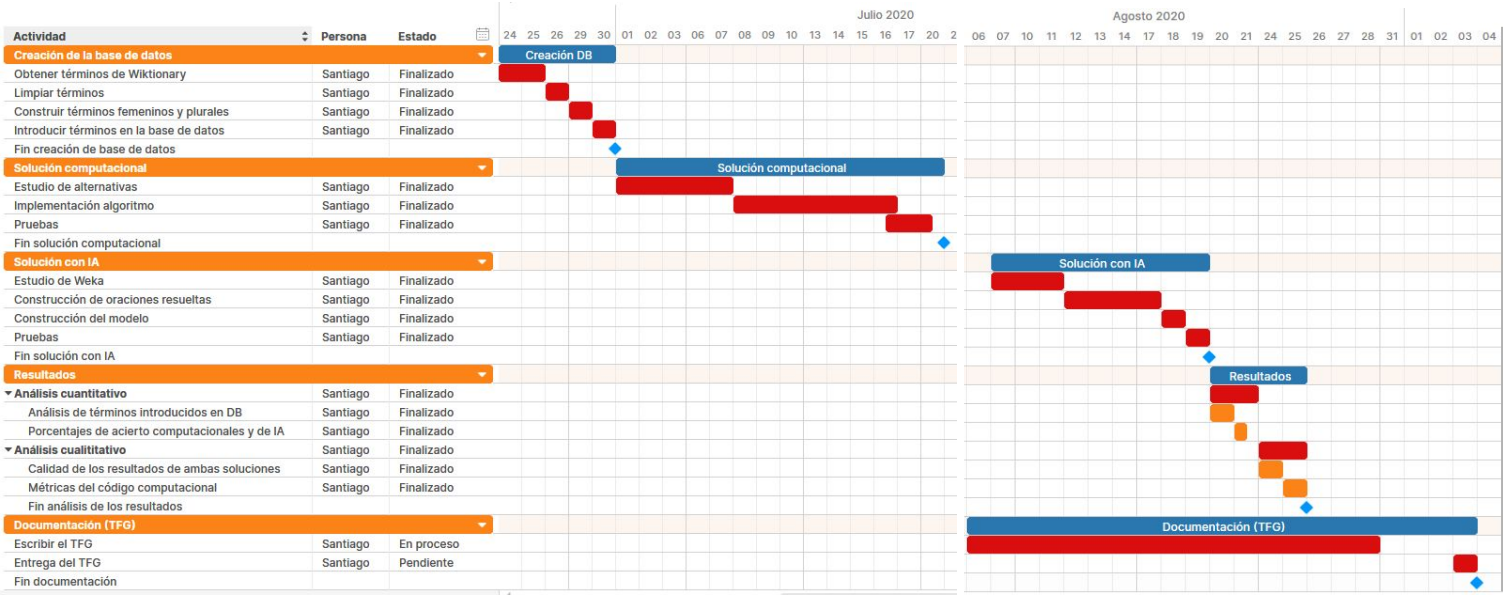


Figura 60: Diagrama de Gantt con Tom 's Planner.

El período vacío entre el 20 de julio y el 6 de agosto fueron unos días en los que no tuve acceso a Internet y no pude trabajar.

7.2 Presupuesto

En cuanto al presupuesto, el único trabajador como tal del proyecto he sido yo. La duración del proyecto han sido 56 días, dedicándole aproximadamente una media de 4 horas al día.

El coste por hora depende mucho del salario anual. Dependiendo de las fuentes que se consulten, un analista programador o ingeniero informático o ingeniero de software, suelen cobrar desde 30.000€ hasta más de 50.000€ anuales, por lo que es complicado estimarlo. Depende mucho del puesto y de la empresa, si es privada, sector público, o eres autónomo. Después de consultar varias fuentes (Indeed, diferentes Universidades, InfoEmpleo, Xataka, Secretaría de Estado de Presupuestos) puedo decir que el salario medio de un perfil analista programador está entre 35.000€ y 45.000€ brutos anuales.

Por otro lado, es necesario saber cuántas horas laborables tiene un año. Nuevamente esto depende de la empresa y de más factores, pero en general se sitúa alrededor de las 1900 horas anuales, repartidas de la siguiente forma:

- 365 días menos 30 días (vacaciones) = 335 días.
- 335 días menos 14 festivos laborales (esto depende de cada Comunidad Autónoma) = 321 días.
- 321 menos 78 días (fines de semana durante 52 semanas) = 243 días.
- 243 x 8 hs./día = **1944 hs. anuales**

Lo anterior se ha obtenido según el Real Decreto Legislativo 2/2015, de 23 de octubre, por el que se aprueba el texto refundido de la Ley del Estatuto de los Trabajadores.

Por lo tanto la estimación quedaría en torno a los 22€/hora (salario anual dividido entre horas laborables anuales).

Tabla 4: Estimación del coste del proyecto. Elaboración propia.

Horas trabajadas diarias	Días trabajados	Total horas	Coste por hora	TOTAL
4	56	224	22€	4.928€

Se podría haber incluido algún coste por la electricidad usada, pero estos 56 días, a 4 horas diarias, con un portátil de 40 vatios y un coste de 0,11€ el kilovatio, apenas me salía 1€ y medio, así que no lo he tenido en cuenta. Relativo a las amortizaciones, el portátil tiene 7 años, hace tiempo que está amortizado, así que tampoco lo he tenido en cuenta.

8. CONCLUSIONES Y TRABAJOS FUTUROS

Los resultados de esta investigación han sido muy fructíferos, por un lado he conseguido obtener unos resultados muy positivos, y por otro me ha servido para afianzar conocimientos sobre esta rama de la IA tan interesante.

Hay que tener en cuenta que el idioma usado para experimentar es el español, que cuenta con una clara distinción entre género masculino y femenino. En otros idiomas como el inglés esto no existe, por lo cual es más complicado resolver las anáforas con algoritmia computacional (tienes menos decisiones que tomar y con las que filtrar los antecedentes).

Para trabajos futuros sería interesante aumentar el nivel de clusterización de sustantivos y pronombres. Por otro lado, investigar y desarrollar algoritmos que resuelvan el fenómeno lingüístico de la catáfora. Es parecido a la anáfora, solo que la referencia se hace anticipando información que vendrá más adelante en el texto. Por último, crear un *corpus* etiquetado con anáforas resueltas en español, y cuyo tamaño sea de varios cientos o miles de oraciones, para futuras investigaciones.

REFERENCIAS BIBLIOGRÁFICAS

Kumar, Ela (2010). *Natural Language Processing*, pp. 1-2

Liddy, Elizabeth (2001). *Natural Language Processing*, pp. 2-3

Real Academia Española, 2005. *Diccionario panhispánico de dudas*.

Saiz, M (2002). *Influencia y aplicación de papeles sintácticos e información semántica en la resolución de la anáfora en español.*

Wiseman, S. Rush, A. Shieber, S. Weston, J. (2016). *Learning Anaphoricity and Antecedent Ranking Features for Coreference Resolution*

Wiseman, S. Rush, A. Shieber, S. (2016). *Learning Global Features for Coreference Resolution*.

Clark, K. Manning, D. (2016). *Improving Coreference Resolution by Learning Entity-Level DistributedRepresentations.*

Seidl, Martina (2015). *UML @ Classroom: An Introduction to Object-Oriented Modeling*.

Comisión Europea (2020). *Libro Blanco sobre la inteligencia artificial, un enfoque europeo orientado a la excelencia y la confianza*, pp. 1-31

Tolan S., Miron M. (2019). *Why Machine Learning May Lead to Unfairness*.

Comisión Europea (2018). *El impacto de la inteligencia artificial en el aprendizaje, la enseñanza y la educación,* pp. 1-42

Henderson, Rebecca (2018). *The Impact of Artificial Intelligence on Innovation,* pp. 1-6

Allam, Zaheer (2019). *On Big Data, Artificial Intelligence and Smart Cities*.

Prater, Adam (2018). *Protecting Your Patients' Interests in the Era of Big Data, Artificial Intelligence, and Predictive Analytics*.

Agrawal, Ajay (2019). *Artificial Intelligence: The Ambiguous Labor Market Impact of Automating Prediction.*

Organización de las Naciones Unidas (2015). *Agenda 2030*. Disponible en: https://www.agenda2030.gob.es/

Ministerio de Asuntos Económicos y Transformación Digital (2020). *Plan España Digital 2025*. Disponible en: https://www.mineco.gob.es/stfls/mineco/prensa/ficheros/noticias/2018/Agenda_Digital_2025.pdf

Comisión Europea (2019). *Programa Europa Digital para el período 2021-2027*.

ANEXO

Glosario de términos

La fuente de las siguientes definiciones son los diccionarios de la Real Academia Española, así como diversas páginas web como Wikipedia.

Corpus: Conjunto cerrado de textos o de datos destinado a la investigación científica.

Parsing: Un analizador sintáctico o simplemente analizador, es un programa informático que analiza una cadena de símbolos de acuerdo a las reglas de una gramática formal. El término proviene del latín *pars*, que significa parte.

Stem: la raíz o parte principal de una palabra.

Token: resultado de separar un texto en unidades más pequeñas, usualmente palabras o caracteres individuales.

Naive: ingenuo en inglés. Es el nombre que se le dió a un algoritmo clásico de resolución de anáforas. Se explica en el Apartado 2 de este trabajo, "Estado del arte".

Parse tree: árbol sintáctico o de derivación. Representa la estructura sintáctica ordenada de una cadena de texto, de acuerdo con los sistemas gramaticales.

Noun phrase: se traduce como sintagma nominal. Es el sintagma o grupo de palabras que forma un constituyente sintáctico maximal, cuyo núcleo está constituido por un nombre o pronombre.

Verb phrase: se traduce como sintagma verbal. Es el verbo o grupo de palabras que forma un constituyente sintáctica verbal, cuyo núcleo está constituido por un verbo.

Sentence: se traduce como oración. Es el conjunto de palabras que expresa un juicio con sentido completo y autonomía sintáctica. Muchas personas confunden oración con frase. Tradicionalmente se usa frase porque la palabra oración se relaciona más con su acepción en un contexto religioso. Una frase carece de núcleo verbal, por lo que el término correcto es oración si el conjunto de palabras tiene algún verbo.

Salience: se traduce como prominencia, relevancia, importancia, rasgo sobresaliente de algo.

Knowledge Manager: se traduce como gestor de conocimiento. En el contexto de este trabajo, es una herramienta comercial de The Reuse Company. Se ha usado para hacer el trabajo.

The Reuse Company / **Reuse**: empresa especializada en la aplicación de tecnologías de análisis y representación semántica a una amplia gama de industrias. Según su web, tienen oficinas en Suecia y España, además de una delegación japonesa.

Query: se traduce como consulta. Es un término informático que se utiliza para hacer referencia a una interacción con una base de datos.

API: La interfaz de programación de aplicaciones, conocida también por la sigla API, en inglés, *application programming interface*, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software.

Dump: se traduce como volcado (de datos, de memoria...). Consiste en volcar, descargar, almacenar toda la información de una estructura de datos (base de datos, memoria o lo que sea) para almacenarla en otro sitio, usualmente con fines de copia de seguridad.

Crawler: traducido como como rastreador, es un programa que analiza la estructura de los sitios web, usualmente para encontrar diversos tipos de documentos, realizar tareas de mapeado y más.

Links: traducido como enlaces, son elementos de un documento electrónico que permiten acceder automáticamente a otro documento o a otra parte del mismo.

XML: *eXtensible Markup Language*, traducido como lenguaje de marcado extensible, permite definir una gramática de marcado para almacenar información de manera estructurada.

Input: traducido como entrada, se refiere a la inserción de información por parte del usuario o programa.

Runtime: traducido como tiempo de ejecución, es el intervalo de tiempo en que un programa se está ejecutando en un sistema operativo.

Plug-in: también se puede encontrar como *plugin*, se traduce como complemento. Es una aplicación o programa informático adicional que se asocia con otro programa (el principal) para ofrecerle determinadas funcionalidades.

SOAP: *Simple Access Object Protocol* traducido como Protocolo Simple de Acceso a Objetos, es un protocolo estándar que define de qué manera objetos en procesos diferentes pueden comunicarse. Está basado en *XML*.

Main(): también denominado *main*, es el procedimiento de inicio de un programa, y recibe ese nombre en muchos lenguajes de programación.

Math(): es una biblioteca de clases muy usada en muchos lenguajes de programación, contiene multitud de funciones matemáticas. Por ejemplo en Java tenemos todas estas.

Cloud: traducido como "computación en la nube" o simplemente "nube", en informática se refiere a todos los posibles servicios tecnológicos que están disponibles sin una gestión activa directa por parte del usuario, y que suelen accederse a través de Internet.

IA/AI: Inteligencia Artificial, o en inglés Artificial Intelligence, se refiere a una rama de las ciencias de la computación que busca conseguir que las máquinas imiten procesos cognitivos pertenecientes a los humanos, ya sea aprender, resolver, jugar, percibir...etc.

OWASP, SANS y CWE:

1. **OWASP** es un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro. Está dirigido por una fundación sin ánimo de lucro.
2. El Instituto **SANS** es una institución con ánimo de lucro fundada en 1989 que agrupa a 165.000 profesionales de la seguridad informática.

   Sus principales objetivos son:

   - Reunir información sobre todo lo referente a seguridad informática
   - Ofrecer capacitación y certificación en el ámbito de la seguridad informática
3. **CWE** es un sistema de más de 600 categorías para las debilidades y vulnerabilidades del software.

Chief Data Officer: traducido como director de datos, es el responsable de elaborar la estrategia acerca de qué tipo de información retendrá la empresa y para qué fines, así como convertirla en un activo valioso para la empresa mediante su análisis y extracción de datos.

Kanban: término japonés que se traduce como valla publicitaria, es un método de producción que persigue la fabricación eficiente de productos, usando el tiempo y recursos necesarios en cada proceso. En su implementación más sencilla, usa tarjetas que se pegan en los contenedores de materiales, y se despegan cuando se han utilizado, para asegurar la reposición de los mismos. Se considera un sistema derivado de *Just in Time* o método de justo a tiempo.

To do, doing, done: traducido como por hacer, haciendo y hecho.

BOE: Boletín Oficial del Estado.

COVID-19: es una enfermedad causada por el virus SARS-CoV-2, un tipo de coronavirus. Se inició en 2019, y a finales de ese año se detectó en China, esparciéndose rápidamente por el resto del planeta. Al momento de escribir este trabajo (verano de 2020), apenas hemos superado en España la primera oleada de esta pandemia, y ya volvemos a tener miles de infectados diarios. Desde el 1 de enero hasta el 24 de mayo, la pandemia se llevó por delante como mínimo a 43.945[1] españoles (cifras oficiales aportadas por el Gobierno: 29.152).

Produce síntomas similares a la gripe, aunque en los peores casos provoca neumonía, sepsis y choque séptico. Por ahora, no hay vacuna. A finales de agosto de 2020, se han informado de manera oficial alrededor de 25,5 millones de casos y más de 850.000 muertos en todo el globo.

[1] Esta cifra se estima a partir de los datos publicados por el INE. "*El número estimado de defunciones en España durante las 21 primeras semanas de 2020 (hasta el 24 de mayo), asciende a 225.930 personas, lo que supone un aumento del 24,1% (**43.945** más) respecto al mismo periodo del año anterior. Por comunidades, los mayores aumentos de mortalidad en ese periodo se dan en la Comunidad de Madrid (72,7%), Castilla-La Mancha (58%) y Cataluña (41%). La semana del año 2020 con mayor número de defunciones fue la 14, que va del 30 de marzo al 5 de abril, con 20.575 personas fallecidas, un 154,65% más que en la misma semana de 2019.*"

Este trabajo se terminó de escribir una tarde a finales de agosto de 2020 en Madrid.