UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Extracción de información de sentencias judiciales

Yimin Pan
Especialidad: Computación

Director: Lluís Padrò Cirera
Codirector: Jose Carmona Vergas
Tutor GEP: Joan Subirats Soler

Febrero 2020

Abstract

The objective of this project is to create and fill a database with the information extracted form the judgments of the Tribunal Supremo (Sala de lo contencioso administrativo) availables in the database of vLex. The extraction process is carried out by the combination of natural language processing techniques, named entity recognition to identify potencial informations and then filter out false positives by matching of patterns with regular expression. This data, will allaw a further analysis of how litigation has evolved in Spain, who are the political actors involved; which are the most disputed policy issues, and also determine until what level has the increase of litigation reached. Thus, allowing the society to know more precisely the role of the courts and their impact in the politics.

Resumen

El objetivo de este proyecto es crear y llenar una base de datos con la información extraída de las sentencias del Tribunal Supremo (Sala de lo contencioso administrativo) disponibles en las bases de datos de vLex. La extracción se hace posible mediante técnicas de procesamiento de lenguaje natural, así como clasificación de entidades combinado con la búsqueda de patrones sobre el texto lematizado mediante expresión regular. Estos datos, permitirán un posterior análisis cuantitativo sobre cómo ha evolucionado el litigio en España, quiénes son los actores políticos involucrados; cuáles son los problemas de políticas públicas más recurridas, y hasta qué punto existe una mayor "litigiosidad". Permitiendo a la sociedad en su conjunto conocer con más precisión el papel de los tribunales y su impacto en las políticas.

Resum

L'objectiu d'aquest projecte és crear i omplir una base de dades amb la informació extreta de les sentències del Tribunal Supremo (Sala de lo contencioso administrativoa) disponibles a les bases de dades de vLex. L'extracció es fa possible mitjançant tècniques de processament de llenguatge natural, així com classificació d'entitats combinat amb cerca de patrons sobre el text lematitzat mitjançant expressió regular. Aquestes dades, permetran un posterior anàlisi quantitativa sobre com ha evolucionat el litigi a Espanya, quins són els actors polítics involucrats; quins són els problemes de polítiques públiques més recorregudes, i fins a quin punt hi ha una major l·litigiositat". Permetent a la societat en el seu conjunt conèixer amb més precisió el paper dels tribunals i el seu impacte en les polítiques.

Índice general

Índice de figuras

Índice de tablas

Índice de códigos

Capítulo 1

Contexto

El proyecto que se llevará a cabo es un trabajo fin de grado de la facultad de informática. Se trata de elaborar una herramienta para extraer información relevante en sentencias judiciales, que fue una parte del proyecto de colaboración entre la universidad de Barcelona y la Politécnica, del cual participa el director y codirector del presente TFG[1].

1.1 Introducción

Los tribunales forman una parte nuclear en el funcionamiento de los sistemas democráticos, intentan preservar los derechos fundamentales de los ciudadanos. En algunos casos son una garantía ante caso de mala praxis o incumplimiento de legalidad por parte de los gobiernos y la administraciones. En los últimos años tenemos ejemplos de que la litigación ha afectado a la política. El caso más reciente es el de organización de consumidores ante el Tribunal de justicia Europeo por el cobro de las cláusulas suelo de las entidades bancarias. Dicho esto, sería interesante analizar los casos de los tribunales, pero antes de comenzar tenemos que hacer una aclaración sobre conceptos que se usará a lo largo del proyecto.

[1]Trabajo Fin de Grado

1.2 Definiciones

1.2.1 Litigación

Se conoce como **litigación o activismo judicial**, un tipo de estrategia o movilización de los grupos de interés que recurren a los tribunales para influenciar los resultados de la política. En algunos países se han analizado los litigios estratégicos, son una forma de respuesta ciudadana a alteraciones de orden jurídico o vulneración de derechos humanos. En general hay una tendencia internacional a la litigaciones por parte de organizaciones ciudadanas.("Aplicación de técnicas de inteligencia artificial al análisis de las decisiones judiciales: actores y temas en conflicto en los tribunales españoles (Polisjuris)" 2019).

1.2.2 Recurso de Casación

Es un recurso extraordinario que tiene objetivo a anular una sentencia por la mala aplicación de una ley (*error in iudicando*), la resolución normalmente le corresponde a la Tribunal Suprema.

1.2.3 Procedimiento contencioso-administrativo

Por `procedimiento contencioso` entendemos que es aquel que se somete a un tribunal y que representa un litigio, disputa o contienda entre partes. Pues se trata de resolver el conflicto mediante el juicio.

Por otra parte también existe `procesos no contenciosos`, dónde la diferencia está en que no existe tal litigio, aunque también requiera un juicio para su resolución.

Finalmente, `proceso contencioso-administrativo` se refiere al ámbito de la jurisdicción que se encarga de controlar que las Administraciones actúen conforme a la leyes y fines preestablecidos.

1.2.4 Litigantes

Dentro de los que litigan, podemos diferenciar dos tipos de litigantes.

1. Repeat Players: grupos experimentado y bien financiados que recurren a tribunales repetidamente.

2. Lonely Players: grupos con menos recursos que recurren a contactos o incluso a una sola ocasión.

Pues es lógico pensar que existe un sesgo a favor hacia un tipo de litigante que otro. Aunque existen también otros variables de entorno político que condicionan al activismo político como las normas procesales de comparecencia ante el tribunal, la disponibilidad de asistencia y los gasto en el tribunal.

1.2.5 Jurisprudencia

Se entiende por jurisprudencia a la doctrina establecida por los órganos judiciales del Estado (por lo general, el Tribunal Supremo o Tribunales Superiores de Justicia) que se repite en más de una resolución. Esto significa que para conocer el contenido completo de las normas vigentes hay que considerar cómo han sido aplicadas en el pasado. En otras palabras, la jurisprudencia es el entendimiento de las normas jurídicas basado en las sentencias que han resuelto casos basándose en esas normas.(*Conceptos Jurídicos: Jurisprudencia* 2017)

1.2.6 Machine Learning

Aprendizaje automático en español, son algoritmos de análisis de datos capaz de crear un modelo matemático que pueda predecir o hacer decisiones sin instrucciones implícitas. Busca patrones entre los datos y intenta aumentar la precisión (o otras métricas) mediante el continuo aprendizaje. Un ejemplo sería la predicción de precio de inmobiliarias o filtrado de correos electrónicos.

1.2.7 Natural Language Processing

Procesamiento de lenguaje natural, es una rama de la inteligencia artificial encargado de la interacción entre la computadora y el humano usando lenguaje natural. La mayoría de técnicas confía en ML para derivar el sentido de las frases del lenguaje humano. El traductor de Google sería un ejemplo de aplicación de NLP.

1.2.8 Crawler

Es un programa que automáticamente busca documentos que hay en el internet. Descarga los ficheros html y los analiza, de modo que puede seguir los links que hay en este para continuar con la navegación. Es la base de los motores de búsqueda.
En nuestro caso, lo implementaremos para descargar los ficheros de sentencia que nos interesea.

1.2.9 API

Interfaz de programación de aplicaciones, es un conjunto de funciones que ofrece cierta librería para ser usado en otro *software*. Facilita la comunicación entre componentes de diferentes *softwares*, de modo que no hace falta entrar en detalles de la implementación.

1.3 Descripción del Problema

En general el análisis sobre política siempre se ha centrado en parlamentos y el gobierno, dejando aparte los tribunales. Por lo que el objetivo de este proyecto es contribuir a llenar el vacío a partir del análisis del caso español. Mediante la jurisprudencia, intentando observar cuales han sido los principales temas en conflicto, quienes son los actores involucrados, los gobiernos con más casos y cuales de ellos han sido estimados.

Posteriormente a partir de los datos extraídos de las sentencias se creará una base de datos que servirá para hacer estudios sobre la litigación en España. Pero esto queda fuera del alcance de este TFG, por lo que no se detallará.

1.4 Solución general

Este estudio es posible en el presente gracias a dos tendencias.
Uno, que los tribunales han empezado a publicar las sentencias en formato electrónico, haciendo posible al análisis de Big Data en el ámbito legal. En este caso tenemos 164.500 mil sentencias de la sala de lo contencioso administrativo del Tribunal Supremo que se almacenan en los servicios de base de datos de la compañía `vlex`.
Y el otro, al avance de técnicas de inteligencia artificial como el machine learning y NLP[2], haciendo posible el procesado automático de estos archivos que manualmente en esta escala sería prácticamente intratable.

1.5 Solución específica

Para empezar implementaremos un crawler que nos permite navegar y descargar las sentencias que están en las bases de datos `vlex`, esto es posible usando el API[3]

[2]natural language processing
[3]Application Programming Interface

que nos proporciona la universidad.
Una vez descargados los archivos, nos interesa extraer la información relevante del texto, que son:

1. Datos de identificación
   (a) Numero de identificación.
   (b) Tipo de recurso.
   (c) Tribunal origen.
   (d) Ubicación del tribunal.
   (e) Fecha de inicio.
   (f) Fecha del juicio.
2. Datos sobre el tema en conflicto y los actores.
   (a) Norma sobre la que se hace litigación.
   (b) Codificación del policy issue[4].
   (c) Administración o gobierno demandado.
   (d) Tipo de demandante: individuo o grupo ciudadano (ONG, Grupo religiosos...etc).
   (e) Asociación profesional.
   (f) Asociación de empresas.
   (g) Asociación de negocio.
   (h) Empresa a título individual.
3. Fundamentos de derecho y fallo.
   (a) Identificación de normativa europea citada por el demandante.
   (b) resultado de la sentencia (fallo).
   (c) Costas procesales y cuantía.

junto con metadatos disponibles con el documento.

[4]temas de política públicas clasificadas en 23 códigos y 243 subcódigos

Las técnicas utilizadas para extraer los diferentes valores variará según el tipo y la estructura del texto. Pero la mayoría (como fechas o ID...) se puede extraer fácilmente por coincidencia de patrón. Pero los campos con mayor variabilidad, como los nombres de personas o grupos, las normas involucradas requieren el uso del sistema de reconocimiento de entidades. Aunque existan ya sistemas desarrollados que se puede usar, como Freeling[5], se espera que con el aprendizaje automático se mejore su rendimiento por la experiencia en los datos de este tipo.

1.6 Actores implicados

1.6.1 Desarrollador

En este caso el equipo de desarrolladores se compone de un solo miembro, el autor del proyecto que lo llevará a cabo como trabajo fin de carrera. Se encarga de todas las tareas como gestionar, planificar, desarrollar, documentar y por último presentar el proyecto.

1.6.2 Director

El profesor Lluís Padrò Cirera del departamento Computer Science (CS) de la facultad de informática hará de supervisor del proyecto. Siendo especialista en el área de procesamiento de lenguaje natural, ayudará y guiará al estudiante desarrollar correctamente el proyecto.

1.6.3 Codirector

El profesor Jose Carmona Vergas también del departamento CS, experto en ciencias de procesos y datos, hará de codirector del proyecto, con una funcionalidad similar al del director.

1.6.4 Departamento de investigación

El proyecto originariamente se trata de una colaboración entre departamentos de dos universidades (UB[6] y UPC[7]), donde el departamento de Ciencia de Computadores desarrolla las herramientas necesarias que el departamento de Ciencias Políticas

[5]un analizador de lenguaje open-source
[6]Universitat de Barcelona
[7]Universitat Politècnica de Catalunya

pueda llevar a cabo el estudio de las sentencias. El equipo de investigación está compuesto por 2 estudiantes de doctorado y 5 profesores de las universidades, de allí también los profesores Padrò y Carmona.

1.6.5 Usuarios

Los principales usuarios serían los investigadores de ciencias políticas o derecho, que usarán esta base de datos para realizar análisis estadísticos relacionado con sus estudios. De allí extraer conclusiones sobre la política o legislación actual.

Capítulo 2

Justificación

El uso de esta metodología nueva, basado en el uso de procesamiento del lenguaje natural y herramientas de *machine learning* sería el primer intento en España de identificar, clasificar y analizar sistemáticamente las decisiones de los tribunales y su impacto en las políticas públicas usando técnicas de inteligencia artificial. Si se observa más allá, en Estados Unidos se han utilizado jurisprudencia recopilada y codificada manualmente. Pero esto ha cambiado hace poco, después de que los tribunales han empezado a publicar las sentencias en formato electrónico, hay un aumento de interés en estos temas.
En una página web de Stanford "Discover Legal Technology"se observa una lista de más de miles de compañía que han empezado a implementar *software* para el sector legal. Sin embargo, después de navegar un poco, no encontramos ningún software que se satisfaga totalmente a nuestras necesidades. Es natural, ya que estamos en un caso muy específico, nos centramos en el caso de sentencias en España.

Por el otro lado, ya existen algunos estudios relacionados, que nos sirve de guía, como un TFG realizado en la universidad de La Laguna(Díaz 2017), llevado a cabo por un estudiante de la ingeniería informática. Pero a diferencia, analiza sentencias del CENDOJ[1] que tiene acceso al público gratuitamente. Esto hace que esas sentencias tengan una estructura diferente a las que se van a analizar en este proyecto. Pero de todas formas, está bien para tener una referencia.

En conclusión, el *software* que se desarrolla con este proyecto tiene que implementar unas funcionalidades muy específicas, por lo que no se ha podido encontrar

[1]Centro de Documentación Judicial

ningún *software* comercial que resuelva el mismo problema en el mercado. Pero actualmente sí existen bastantes estudios sobre el tema de extracción de información, dado que es una área muy activa en investigación. (Krass 2018)

Capítulo 3

Alcance del proyecto

A continuación se define en orden los objetivos a lograr para solucionar el problema descrito anteriormente, los objetivos funcionales, no funcionales y posibles riesgos.

3.1 Objetivos funcionales

3.1.1 Descargar las sentencias

Las sentencias están almacenadas en los servicios de base de datos de (vlex[1]), por lo cual implementaremos un crawler que nos permite navegar y descargar estas, todo esto haciendo uso del API que se nos proporciona.

Subobjetivos

1. Conocer el funcionamiento de un crawler: buscar información y proyectos ya hechos.

2. Aprender a utilizar el API: leerse la documentación y descripción de las funciones.

3. Pensar una manera de organizar y almacenar las sentencias descargadas (son millones de ficheros).

[1]Una de las mayores colecciones de información jurídica del mundo en una plataforma de inteligencia artificial

3.1.2 Extraer información de partes estructurada

De los ficheros de texto (html) descargados, identificar los campos indicados anteriormente en la sección de solución específica. Los textos siguen un formato, por lo que no será difícil identificarlos por coincidencia de patrón.

Subobjetivos

1. Analizar los ficheros html.
2. Ver los *tags* que corresponden a cada campo.

3.1.3 Extraer información de las partes con más variabilidad

Utilizaremos técnicas de procesamiento de lenguaje natural para los partes de texto sin formato predeterminado, como por ejemplo, el nombre de las entidades que están repartidos por todo el texto. Se podrá usar librerias como el freeling para facilitar el trabajo, y se espera que con el entrenamiento (machine learning) se mejore su precisión.

Subobjetivos

1. Entender el flujo típico de NLP: tokenización, lematización, análisis morfológico, análisis sintáctico.
2. Leer la documentación de freeling y saber aplicarlo al proyecto.
3. Buscar otras librerías de NLP que pueda ser útil.
4. Determina la técnica de machine learning para el entrenamiento.

3.1.4 Almacenar las informaciones extraídas

Por último, si los pasos anteriores se han logrado con éxito y queda tiempo, guardaremos los datos extraídos en ficheros con una estructura fija como por ejemplo JSON.

3.2 Aspecto no funcional

Al ser una herramienta se enfocará más en su rendimiento que en aspectos como la seguridad, fiabilidad y escalabilidad que en este caso no tiene demasiado sentido. Sin embargo, sí que se considera la modularidad del *software*, se intentará desarrollar las funcionalidades independientemente. La usabilidad también es un aspecto importante, ya que probablemente los usuarios finales serán personas sin experiencia de programación (investigadores de ciencias políticas).

3.3 Obstáculos y riesgos

Como obstáculo principal cabe destacar la inexperiencia en los métodos de procesamiento de lenguaje natural, por lo que el tiempo de entrega puede ser una limitación ya que habrá que contarle el tiempo de aprendizaje a parte de la de implementación. Cabe decir también que los procesos de aprendizaje en `Machine Learning` son costosos en cuanto a tiempo. Pero como se explica posteriormente, en el capítulo de riesgos, ya se tiene en cuenta al hacer la planificación de las tareas, asignándoles más tiempo.

Por el otro lado, el uso de librerías externas y APIs, al ser diseñado por otros, también añaden variabilidad al proyecto. En los casos peores pueden estar obsoletos, por lo que deberíamos buscar alternativas que sustituyen sus funcionalidades.

Capítulo 4

Metodología y rigor

Dado que no se trata de un proyecto TFG unipersonal donde no se habrá que comunicar constantemente con los clientes, los requerimientos están ya bien definidos al principio. Por lo que adaptaremos el método cascada para el desarrollo del software, definiremos las tareas a realizar y sus respectivos deadlines, de este modo tenemos una secuencia de actividades que se corresponde bastante bien al conjunto de funcionalidades donde unos son prerequisito de otros que hemos definido en apartados anteriores.

Por otra parte, se realizará reuniones semanales con el director para resolver dudas y asegurar que incrementalmente se van añadiendo funcionalidades al software.

4.1 Herramientas a usar

4.1.1 Gantter

Una aplicación gratuita para desarrollar los diagramas de Gantt. Permite tener una visualización de las tareas planificadas.

4.1.2 Git

Se usará el repositorio de Github como herramienta para controlar las versiones del software. En el caso que haya errores podemos volver fácilmente a versiones anteriores.

4.1.3 Correo electrónico

Para comunicarse con el director o tutor de GEP, principalmente se usará el correo electrónico del `racó` o gmail.

4.1.4 Overleaf

Es un editor de texto online gratuito para `latex`, será usado para elaborar las entregas de GEP y la memoria final. Es prefente ante `Office Word` o `Google docs`, al ser fácil de trabajar con expresiones matemáticas o ecuaciones.

4.1.5 Atom

Será el editor de texto utilizado para la implementación del código. Destaca por su simplicidad y compatibilidad con los lenguajes de programación.

4.1.6 Trello

Será usado para hacer seguimientos de las tareas a realizar cada semana.

4.1.7 Lenguaje de programación

Python será el lenguaje utilizado, al tener una multitud de librerías de machine learning, y el API de vlex también es para Python, se adapta bien a la necesidad del proyecto.

4.1.8 Freeling

Como habíamos comentado antes, será la herramienta que usaremos para el análisis sintáctico de los textos.

4.1.9 Otras librerías y APIs

Habrá la necesidad de incorporar más librerías para funcionalidades indicadas anteriormente. Por lo mínimo, el de vlex y uno de crawler.

Capítulo 5

Descripción de las tareas

5.1 Duración del proyeccto

El proyecto al ser un TFG tiene un límite de durada definido, que en principio será el segundo turno de lectura de 2020-Q1, que corresponde al 29 de junio. Teniendo en cuenta que se entrega como mínimo con una semana de antelación, añadiéndolo una semana más de margen para no ir tan apretado, estamos al 15 de junio. Y por otra parte, la fecha de comienzo es el mismo que la primera sesión de GEP, el día 17 de febrero. Dicho esto, contamos con unos 118 días para este trabajo.

Teneiendo en cuenta que el TFG son 18 créditos, donde cada crédito equivale a 25 horas en la UPC, calculamos unas 450 horas de trabajo, pero después de hacer la descripción de todas las tareas se extiende a 480 horas. Y repartiéndolo entre los 118 días, se estima trabajar unas 4 horas diarias para el proyecto.

5.2 Tareas

A continuación se clasificará las actividades en diferentes grupos según su definición.

Gestión de proyectos

- Estudio de materias de GEP: lectura de ficheros de cada módulo.
- Definición del contexto y alcance.
- Planificación temporal del proyecto (diagrama Gantt).

- Calculo del presupuesto e informe de sostenibilidad.
- Integración del documento final: revisión de las tareas anteriores, elaboración del documento final y su incorporación a la memoria.
- Preparación para la presentación oral.

Reuniones

A lo largo del proyecto se realizarán reuniones con el director para hacer control sobre el progreso y realizar algunas adaptaciones si es necesario. La frecuencia depende del avance y las dudas que puedan ocurrir, en media cada 10 días, y cada reunión dura 1 hora aproximadamente. Por lo que en total calculamos unos 10 horas.

Trabajo previo

- Estudio previo sobre el tema.
- Búsqueda de *software* ya existente.
- Testeo de funcionalidades de los *software* encontrados.
- Definir funcionalidades a implementar en el proyecto.
- Determinar herramientas a utilizar.

Aprendizaje

Las tareas de aprendizaje relacionadas con las técnicas o herramientas a utilizar en el proyecto.

- Estudio sobre el API que proporciona `vlex`.
- Estudio sobre *crawler* para Python, así como librerías y funcionamiento.
- Estudio sobre `Freeling`.
- Estudio sobre NLP + Machine Learning.

Diseño e implementación del *software*

Corresponden a los objetivos descritos en capítulos anteriores.

- Implementación del *crawler*.
- Estudio de los ficheros html descargados.
- Implementación de código para extracción de información bien estructurada.
- Implementación de técnicas NLP para extraer el resto de información:
  - Preprocesado de los datos de entrada.
  - Incorporar `Freeling`.
  - Aplicar técnicas de machine learning.
- Testing de las funcionalidades anteriores: comprobar que cumple con los objetivos.
- Diseño de interfaz para guardar o presentar los resultados.
- Valoración de resultado y posibles mejoras.

Memoria

Por último, después de realizar la última entrega de GEP, se empezará a desarrollar la memoria, incorporando el documento final de la parte de gestión de proyecto y añadiéndole las partes técnicas del proyecto.

Posteriormente, se preparará para la presentación, incluyendo el tiempo ncesario para elaborar los ppts[1].

5.3 Tabla de tareas

Al final resulta unas **480 horas**, cercano a como se había pensado y a los créditos que vale el trabajo.

[1]Power Points

5.4 Dependencias

Es fácil de razonar sobre las dependencias que se especifica en la tabla anterior. En las tareas relacionadas con GEP, las entregas son incrementales, es decir, la tercera entrega incluye la segunda, la segunda a la primera... De este modo, no podemos realizar una entrega sin haber acabado las anteriores. Lo mismo con la memoria, se elaborará a partir del documento final. Y la presentación lógicamente lo preparamos después de la lectura de la memoria.

En la parte de trabajo previo, no podemos testear los *softwares* sin haberlos encontrado. Y las tampoco podemos determinar las funcionalidades a implementar y herramientas a utilizar sin haber hecho un estudio previo del tema.

Por último, el desarrollo e implementación es lo que conlleva más tiempo, ya que incluye también el tiempo de aprendizaje de las herramientas, de allí las dependencias que vienen del grupo T4, no podemos aplicar una técnica sin haberlo analizado y estudiado.
Las dependencias entre las tareas de T5 siguen el flujo como se había descrito en el apartado de objetivos, para empezar a trabajar, primero tenemos que implementar el *crawler* y descargar los ficheros. Después, para la extracción de información, tenemos dos partes diferenciadas que se podría trabajar concurrentemente sin problema. Sin embargo, las tareas del grupo 5.4 sí que tienen dependencias, que es el flujo de trabajo típico de NLP: preprocesado, estructurar los objetos, análisis y transformación.
El testing de la funcionalidades y valoración de resultado, se realizará después de tener implementadas esas dos partes, como lógico.

5.5 Recursos

A continuación se detallará los tipos de recursos necesarios para el desarrollo del proyecto.

Recurso humano

En este caso, al ser el TFG unipersonal, una persona se encargará de todo el proyecto. Tanto en la parte de gestión de proyecto como el desarrollo del *software*, siguiendo la planificación definida por el Gantt.

El director y tutor son los otros actores involucrados, que tienen la función de supervisar al estudiante, uno en la parte técnica y otro en cuestiones de gestión del proyecto.

Recurso material

Todo el recurso material físico para llevar a cabo el proyecto es una computadora. Moderna y potente si es posible, para las tareas de aprendizaje automático. En todo caso, podríamos añadirle el lugar de trabajo del desarrollador, que principalmente es en casa o en la biblioteca de la UPC. Los *software* necesarios son las descritos en el capítulo de metodología y herramientas : Gantter, Overleaf, Atom, Vlex, Git, Trello, correo electrónico, el racó, Atenea, Python, Freeling, y más librerías de Machine Learning .

Tabla de Tareas			
Código	Tarea	Tiempo (horas)	Dependencia
T1	**Gestión de Proyecto**	**115**	
T1.1	Estudio de materia de GEP	20	
T1.2	Contexto y alcance	25	T1.1
T1.3	Planificación	20	T1.2
T1.4	Presupuesto y sostenibilidad	20	T1.3
T1.5	Documento final	20	T1.4
T2	**Reuniones**	**10**	
T2.1	Reuniones con el director	10	
T3	**Trabajo Previo**	**40**	
T3.1	Estudio previo sobre el tema	20	
T3.2	Búsqueda de *software* ya existente	5	
T3.3	Testeo de funcionalidades de *software* encontrados	8	T3.2
T3.4	Definir funcionalidad a implementar en el proyecto	5	T3.1
T3.5	Determinar herramienta a utilizar en el proyecto	2	T3.4
T4	**Aprendizaje**	**60**	
T4.1	Estudio sobre API de `vlex`	10	
T4.2	Estudio sobre *crawlers*	10	
T4.3	Estudio sobre `Freeling`	15	
T4.4	Estudio sobre NPL^2+ML^3	25	
T5	**Diseño e implementación**	**190**	
T5.1	Implementación del *crawler*	20	T4.1, T4.2
T5.2	Análisi de ficheros descargados	5	T5.1
T5.3	Extracción de información estructurada	30	T5.2
T5.4.1	Preprocesado de datos	10	T4.4, T5.2
T5.4.2	Incorporar `Freeling`	20	T4.3, T5.2
T5.4.3	Aplicar `Machine Learning`	50	T4.4, T5.4.1
T5.5	Testing de funcionalidades	20	T5.3, T5.4.3
T5.6	Diseño de interfaz	25	
T5.7	Valoración de resultado y mejoras	10	T5.5
T6	**Memoria**	**65**	
T6.1	Elaboración de la memoria	50	T1.5
T6.2	Presentación	15	T6.1
	Suma Total de Horas	**480**	

Cuadro 5.1: *Tabla de tareas y dependencias. Elaboración propia*

Capítulo 6

Estimaciones y Gantt

Se usará `Gantter`[1] para la elaboración del diagrama de Gantt, donde se puede observar las dependencias entre las tareas y sus fechas de inicio y fin. Las horas asignada a cada actividad son las descritas en la tabla de tareas. La fecha fin del proyecto como se indica en el capítulo anterior, es el 15/06/2020. Con la excepción de la presentación, que tiene límite hasta el día de la lectura 29/06/2020.

[1] Aplicación web gratuita para crear gráficas de Gantt.

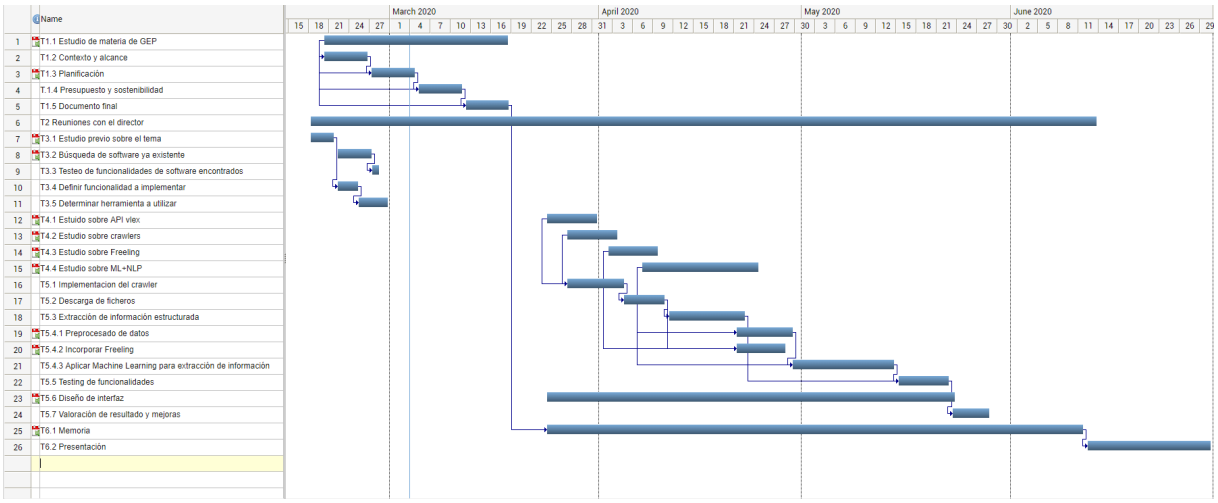


Figura 6.1: *Diagrama Gantt. Elaboración propia mediante* **Gannter**

Capítulo 7

Gestión del riesgo: Planes alternativos y obstáculos

A continuación se mencionarán los obstáculos que pueden ocurrir durante el proyecto (algunos ya comentado anteriormente) y qué alternativas posibles hay en caso que sucedan.

7.1 Mal funcionamiento de las librerías

Dado que en este proyecto se utilizará varias librerías y APIs, si algunos de estos no funcionara o esté obsoleto, puede afectar bastante en la secuencia de actividades, ya que como hemos visto existen dependencias entre la implementación de las funcionalidades. En este caso, la única opción que nos queda es buscar alternativas que puedan sustituirles, que implica añadir más tiempo al proyecto. Pasaremos a dedicar 4 horas diarias a 5 o 6 hasta que ya no afecte al resto del plan.

7.2 Problemas al acceso de los base de datos

Es un problema bastante grave si la universidad dejara de proporcionar acceso a los bases de datos, así como deshabilitando el API de `vlex`. En este caso deberíamos redefinir el proyecto, ya que el objetivo principal es el análisis de las sentencias almacenadas allí.

7.3 Inexperiencia con las herramientas

Esta situación ya se ha tenido en cuenta a la hora de hacer la planificación, pues se dedicará horas únicamente a reforzar los conocimientos necesarios, por lo que no debería ser un problema.

7.4 Rendimiento del sistema

No se debería presentar problemas con la extracción de las partes estructuradas, sin embargo no se puede asegurar nada respecto a la otra parte. Ya que como sabemos, machine learning se basa en probabilidades, no es raro obtener sistemas con poca precisión (50) en según qué tareas de análisis de textos, donde la dimensionalidad es un problema que limita al rendimiento.
El rendimiento depende de varios factores, uno es el preprocesado y el otro la técnica aplicada(sigmoider 2017). Se podría probar combinaciones diferentes de estas, solo que el inconveniente es dedicar más tiempo al proyecto. Ya que como se había planificado, se espera obtener un resultado aceptable después de unas pocas pruebas, los procesos de aprendizaje automático suelen ser costosos de tiempo.

Capítulo 8

Presupuesto

8.1 Identificación y estimación de los costes

Partiendo de la planificación de los capítulos anteriores, ahora se realizará el análisis económico de este proyecto. Así como el estudio sobre los costes humanos, *hardware*, *software* e indirectos. Luego, también se tendrá en cuenta las contingencias y imprevistos que pueda ocurrir en el proyecto.

8.1.1 Coste humano

Estudio sobre los salarios

Para calcular el coste humano, primero habrá que saber cuando gana cada personas que realiza los diferentes roles, que vienen determinado por las tareas de la tabla 5.1. Para la estimación de los salarios se ha tenido como referencia la información de puestos de trabajo encontrada en `Glassdoor`(*Glassdoor Job Search — Find the job that fits your life*). Y aplicándole un 0.3 más, en concepto de seguridad social.

Tabla de precio por hora (€/h)			
Jefe de proyecto	Analista	Programador	Tester
29	22	19	20

Cuadro 8.1: *Tabla de sueldos. Información extraída en glassdoor, elaboración propia*

Coste de las tareas

El coste de las tareas de la tabla, se calculará a partir de su sueldo (euro por hora) multiplicándolo por las horas que trabaja en cada tarea.
Algunas asignaciones de tareas resulta un poco ambiguo, en el caso de los estudios sobre algunas técnicas lo podemos asignar al analista o al desarrollador, en este caso lo hemos asignado al primero. Pero incluso diría que es más adecuado asignarlo al desarrollador, ya que es el que se encarga de la implementación. Esto pasa debido a que se trata de un TFG, y todo el proyecto lo lleva una persona (el estudiante). En un proyecto real, esto no debería pasar, ya que las personas que asumen cada rol es profesional en esas tareas y no requieren procesos de aprendizaje planificadas implícitamente.

Tabla coste de tareas			
Tarea	Rol	Horas	Coste €
T1.1 Estudio de materia de GEP	Jefe de proyecto	20	580
T1.2 Contexto y alcance	Jefe de proyecto	25	725
T1.3 Planificación	Jefe de proyecto	20	580
T1.4 Presupuesto y sostenibilidad	Jefe de proyecto	20	580
T1.5 Documento final	Jefe de proyecto	20	580
T2.1 Reuniones con el director	Jefe de proyecto	10	290
T3.1 Estudio previo sobre el tema	Analista	20	440
T3.2 Búsqueda de *software* ya existente	Analista	5	110
T3.3 Testeo de funcionalidades de *software* encontrados	Analista	8	176
T3.4 Definir funcionalidad a implementar en el proyecto	Analista	5	110
T3.5 Determinar herramienta a utilizar en el proyecto	Analista	2	44
T4.1 Estudio sobre API de `vlex`	Analista	10	220
T4.2 Estudio sobre *crawlers*	Analista	10	220
T4.3 Estudio sobre `Freeling`	Analista	15	330
T4.4 Estudio sobre NPL[1]+ML[2]	Analista	25	550
T5.1 Implementación del *crawler*	Programador	20	380
T5.2 Análisi de ficheros descargados	Programador	5	95
T5.3 Extracción de información estructurada	Programador	30	570
T5.4.1 Preprocesado de datos	Programador	10	190
T5.4.2 Incorporar `Freeling`	Programador	20	380
T5.4.3 Aplicar `Machine Learning`	Programador	50	950
T5.5 Testing de funcionalidades	Tester	20	400
T5.6 Diseño de interfaz	Programador	25	475
T5.7 Valoración de resultado y mejoras	Tester	10	200
T6.1 Elaboración de la memoria	Jefe de proyecto	50	1450
T6.2 Presentación	Jefe de proyecto	15	435
Total		**480**	**11060**

Cuadro 8.2: *Tabla coste de tareas. Elaboración propia*

8.1.2 Coste hardware

Hacienda permite amortizar las herramientas y equipamiento informático en 3-4 años, por lo que consideraremos 4 años para la vida útil del *hardware*. Dicho esto, el calculo de la amortización no es más que:

$$precioHora = \frac{precio}{horasVidaUtil}$$
$$Amortizacion = precioHora \cdot horasDuracionProyecto$$

Un año són 220 días hábiles y suponiendo que se trabaja 8 horas al día, 4 años de vida útil serán 7040 horas. Y la duración del proyecto son como hemos planificado anteriormente 480 horas.

Tabla de costes de hardware			
Hardware	Valor €	Precio por hora €/h	Amortización €
Portatil Asus Zephyrus	1499	0.21	100.1
PC laboratorio FIB[3]	1000	0.14	68.2
Ratón Razer Mamba	80	0.01	4.8
Alfrombrilla Razer	20	0.003	1.4
Total			**174.5**

Cuadro 8.3: *Tabla de costes del hardware. Elaboración propia*

8.1.3 Coste software

El coste de *software* se calcula igual que el *hardware*, solo que el periodo de renovación o amortización según Hacienda es más corto, son 2 años, que en término de horas son 3520 horas hábiles.

Tabla de costes de software			
Software	Valor €	Precio por hora €/h	Amortización €
Overleaf	0	0	0
Github	0	0	0
Python	0	0	0
Freeling	0	0	0
Atenea UPC	0	0	0
Google	0	0	0
Windows 10 Professional	199	0.06	27.1
Total			**27.1**

Cuadro 8.4: *Tabla de costes del software. Elaboración propia*

8.1.4 Costes indirectos

Los costes indirectos incluyen aquellos que no afectan directamente al proyecto, ya que se pagan constantemente, sin embargo sí lo tenemos que contabilizar. Lo calculamos de forma similar a los anteriores, solo que ahora consideramos todas las horas de un año, es decir:

$$356dias \cdot \frac{24horas}{dia} = 8544horas$$

Tabla de costes indirectos			
Concepto	Precio anual €	Precio por hora €/h	Amortización €
Agua	276	0.03	14.4
Electricidad	420	0.05	24
Gas	576	0.07	33.6
Internet y teléfono	540	0.06	28.8
Vivienda	15600	1.8	864
Total			**964.8**

Cuadro 8.5: *Tabla de costes indirectos. Elaboración propia*

8.1.5 Contingencias

Para contemplar posibles imprevistos durante el proyecto se considera un porcentaje de 15 % de los costes anteriores: humano, hardware, software e indirectos.

$$(11060 + 174{,}5 + 27{,}1 + 964{,}8) * 0{,}15 = 1833{,}9$$

8.1.6 Imprevistos

Ahora asumiremos los costes de solucionar los riesgo o obstáculos que del capítulo 7. Aunque muchos de ellos, como se había comentado, ya se ha tenido en cuenta al realizar la planificación, hay otros que necesitan recursos extra para alternativas o soluciones al problema. El coste, entonces, está ligado a la probabilidad de que suceda el imprevisto. Y los que el coste son horas extras se calculará multiplicando las horas necesarias con el sueldo del que realiza la tarea.

Tabla de costes de imprevistos			
Concepto	Precio	Riesgo %	Coste
Librerías obsoletas	380	20	76
Avería del portátil	1499	10	149.9
Total			**225.9**

Cuadro 8.6: *Tabla de costes de imprevistos. Elaboración propia*

8.1.7 Presupuesto final

El sumatorio de los totales anteriores. Cabe decir que el valor de todos los costes anteriores ya contiene el IVA, por lo que no hace falta recalcularlo.

Presupuesto final	
Tipo de coste	Precio €
Costes humano	11060
Costes hardware	174.5
Costes software	27.1
Costes indirectos	964.8
Contingencias	1833.9
Imprevistos	225.9
Total	**14286.2**

Cuadro 8.7: *Tabla de totales. Elaboración propia*

8.2 Control de gestión

Para asegurar que el proyecto se lleve a cabo dentro del presupuesto calculado anteriormente, después de acabar cada tarea se realizará un cálculo del coste real de la actividad, de este modo es fácil controlar dónde se ha producido la desviación, si se trata de una desviación en tarifa o en eficiencia y calcular su valor mediante las fórmulas que aparecen después(GEP 2019, pg.15).

Dicho esto, el desvío de cada parte (humano, hardware, software...), es la diferencia entre el coste real y el estimado. Que en principio el coste real siempre es igual o mayor que el estimado, ya que por ejemplo en casos de que en las actividades si nos sobrase tiempo podemos aprovecharlo para mejorar la calidad de la tarea. Por lo que la diferencia esta nunca será negativa, y la desviación total resulta del sumatorio de

estos.
Por otra parte, para no sobresalir del presupuesto se evitará al máximo usar materiales que no estén descritos anteriormente. Y si alguna actividad se ha llevado a cabo con suficiente éxito y sobra tiempo, se podrá empezar la siguiente tarea por adelantado para tener más margen de imprevistos. Y si en caso contrario, hubiera un aumento de coste (o horas) respeto al estimado, se intentará mejorar la eficiencia, ya que no es posible reducir la tarifa, de las tareas siguientes tareas para así compensar.

$$\text{Desvío por tarifa} = (\text{coste estimado} - \text{coste real}) * \text{consumo horas real}$$

$$\text{Desvío por eficiencia} = (\text{horas estimado–horas real}) * \text{precio real}$$

Capítulo 9

Informe de sostenibilidad

9.1 Autoevaluación

La encuesta relacionada con el tema de sostenibilidad ha sido útil para comprobar si tengo conocimiento suficiente sobre el tema. De hecho, antes de cursar GEP y hacer esta documentación, a lo largo de la carrera tuve que hacer trabajos donde también estaba involucrado la parte de sostenibilidad. Sin embargo, nunca entré muy en detalle, analizando los diferentes componentes, la económica, ambiental y social. Pues no estaba muy informado sobre este tema y tampoco le daba tanta importancia como el proyecto en si.
Dicho esto, la realización de este proyecto me ayuda a concienciar y reflexionar sobre este aspecto, de cara a futuros proyectos se trabajará mas en profundidad.

9.2 Dimensión económica

Como se muestra en el capítulo de estimación de presupuestos, para minimizar el coste hemos usado el mínimo de *hardware* necesario, tan solo las computadoras y herramientas para trabajar con eficiencia. Similarmente con el *software*, que principalmente son gratuitas. Pues la mayor parte del presupuesto viene del coste humano, y este no es posible disminuirlo a no ser que reducir las horas dedicadas a cada tarea. Pero tendría una implicación directa en la calidad del producto final, por lo que no compensaría.
Por otra parte, dado que no se ha encontrado información de proyectos similares es difícil hacer comparaciones. Pero como hemos comentado, lo único que se podría optimizar es el coste humano, pero tanto una reducción en tarifa como eficiencia tiene

efectos sobre la calidad final del producto.
En cuanto a la vida útil, al ser un producto de software, podríamos decir que su coste es mínimo. Tan solo se necesita una computadora y la electricidad para moverla, que hoy en día para la mayoría no se considera un coste adicional.

9.3 Dimensión ambiental

Similarmente a la dimensión ambiental, tanto durante el proceso de desarrollo como la vida útil el software únicamente consume electricidad, que se traduce a emisiones de CO_2 a la atmósfera. En cuanto a la fuente de la energía se conoce lo mínimo, pero en España sabemos que los fuentes renovables cubren alrededor de un 40 % de la demanda total(*Energía en España*).
La computadora principal que se usa en el proceso de desarrollo es un portátil, que consume en media unos 100W, mucho menor que una torre que suele tener una potencia de 500W. Después de su vida útil, si su funcionamiento aún es adecuado, se intentará llevar a segunda mano para aprovecharlo al máximo, que indirectamente reduce la huella ecológica en comparación con que la persona se compre otro nuevo. Y si ya no funciona, simplemente se llevará a un punto de recogida de residuo electrónico.

9.4 Dimensión social

Para empezar, con la realización de este proyecto, el actor más afectado será el autor del TFG. Con este trabajo, será capaz de poner a prueba los conocimientos adquiridos durante los 4 años del grado de ingeniería informática. Y cuando acabe el proyecto, habrá completado sus estudios de grado en la facultad de informática.
Por el otro lado, los principales beneficiarios durante la vida útil del *software* son los estudiantes de ciencias políticas o derecho. La herramienta desarrollada en el proyecto facilitará la extracción de datos de las sentencias, que se podría someter a posterior estudio mediante la estadística. De forma similar también se podría extender su uso en la formación de estos.

Capítulo 10

Sentencias judiciales

El objetivo principal de este proyecto, como explicado anteriormente, es desarollar una herramienta que permita descargar una gran cantidad de sentencias de vLex, extraer información de interés y almacenarlos para posterior estudio. Dicho esto, los campos que nos interesa de las sentencias vienen fijadas por la memoria científico técnica de ayudas fundación BBVA a equipos de investigación científica 2019 (POLISJURIS) **polisjuris** del que deriva este proyecto. Anteriormente también mencionado en el capítulo de descripción de contexto.

1. Datos de identificación
   (a) Numero de identificación.
   (b) Tipo de recurso.
   (c) Tribunal origen.
   (d) Ubicación del tribunal.
   (e) Fecha de inicio.
   (f) Fecha del juicio.
2. Datos sobre el tema en conflicto y los actores.
   (a) Norma sobre la que se hace litigación.
   (b) Codificación del policy issue[1].
   (c) Administración o gobierno demandado.

[1]temas de política públicas clasificadas en 23 códigos y 243 subcódigos

(d) Tipo de demandante: individuo o grupo ciudadano (ONG, Grupo religiosos...etc).

(e) Asociación profesional.

(f) Asociación de empresas.

(g) Asociación de negocio.

(h) Empresa a título individual.

3. Fundamentos de derecho y fallo.

   (a) Identificación de normativa europea citada por el demandante.

   (b) resultado de la sentencia (fallo).

   (c) Costas procesales y cuantía.

La plataforma vLex proporciona una colección inmensa de sentencias, de hecho es uno de los líderes mundial en el sentido de cantidad de depósito de documentos judiciales. Pero no todas las sentencias está en nuestro interés de estudio, por lo que solamente nos centraremos en descargar y analizar las sentencias que provienen de Salas de lo Contencioso-Administrativo del Tribunal Supremo (recursos de casación). Las sentencias que provienen de dicha sala, pues tienen un formato bastante definido, cosa que hace posible este trabajo, y en vLex las podemos encontrar como archivos pdf o html, que en el proyecto lo descargaremos en html por su facilidad de "parseçon algunas librerías de Python. A continuación se explica las partes que compone una sentencia.

10.1 Datos de identificación

En esta sección podemos encontrar información básica de la sentencia, así como el número de recurso que lo identifica, el emisor (tribunal de procedencia), ponente...etc

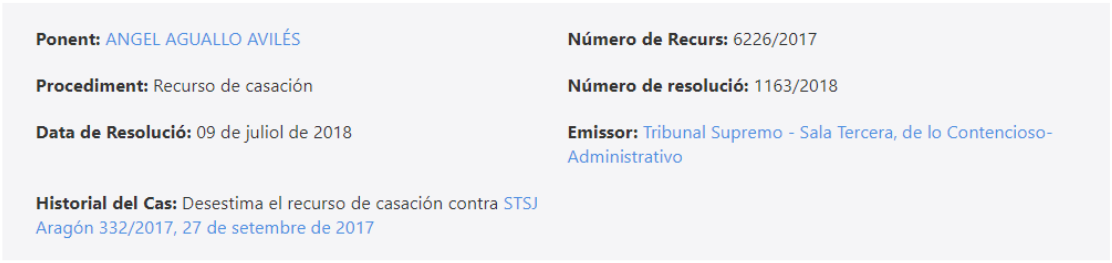


Figura 10.1: *Ejemplo de sección de datos. Extraída de vLex*

10.2 Resumen

Es un pequeño párrafo que resume el tema de conflicto y como se ha resolvido en la sentencia que se quiere hacer la casación. Es una sección opcional que puede no aparecer en todas las sentencias.

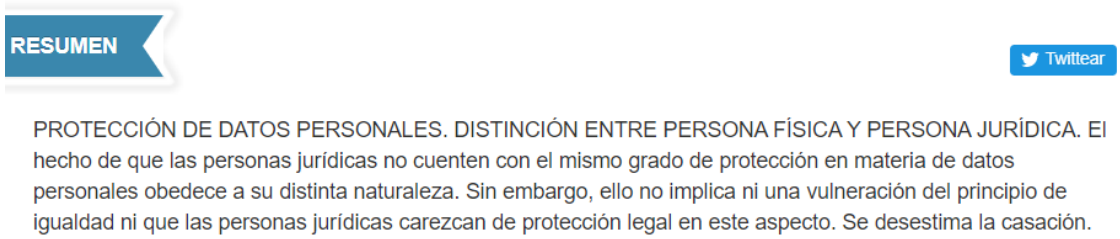


Figura 10.2: *Ejemplo de sección de resumen. Extraída de vLex*

10.3 Contexto

Es una versión extendida del resumen, presenta una estructura muy definida donde podemos encontrar información de las partes que recurren al juicio, los procuradores, el tribunal de origen, fecha de resolución y explicación del tema a litigar.

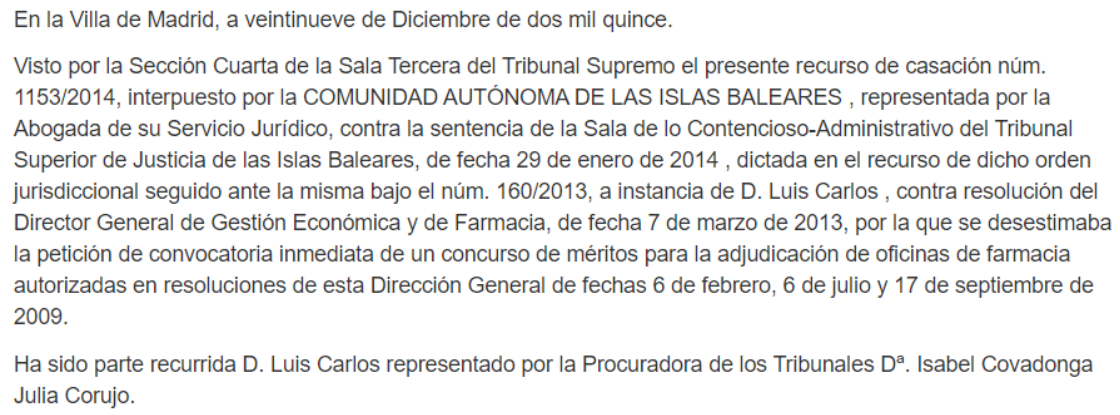


Figura 10.3: *Ejemplo de sección de contexto. Extraída de vLex*

10.3.1 Extra

En algunas sentencias podemos encontrar juntamente al contexto información detallada de los miembros del tribunal que ha interpuesto la sentencias, así como también tribunales anteriores que ha fallado la sentencia.

S E N T E N C I A

Sentencia Nº: /

Fecha de Sentencia: 29/12/2015

RECURSO CASACION Recurso Núm.: 1153 / 2014

Fallo/Acuerdo: Sentencia Desestimatoria

Votación: 15/12/2015

Procedencia: T.S.J.ILLES BALEARS SALA CON/AD

Ponente: Excmo. Sr. D. Ángel Ramón Arozamena Laso

Secretaría de Sala : Ilma. Sra. Dña. María Josefa Oliver Sánchez

Escrito por: MDC

Nota:

Convocatoria de concurso de méritos para la adjudicación de oficinas de farmacia. Se rechaza la excepción de cosa juzgada. Inactividad de la Administración. Motivación y congruencia de la sentencia.

RECURSO CASACION Num.: 1153/2014

Votación: 15/12/2015

Ponente Excmo. Sr. D.: Ángel Ramón Arozamena Laso

Secretaría Sr./Sra.: Ilma. Sra. Dña. María Josefa Oliver Sánchez

S E N T E N C I A /

TRIBUNAL SUPREMO.

SALA DE LO CONTENCIOSO-ADMINISTRATIVO SECCIÓN: CUARTA

Excmos. Sres.: Presidente:

D. Segundo Menéndez Pérez

Magistrados:

Dª. María del Pilar Teso Gamella

D. José Luis Requero Ibáñez

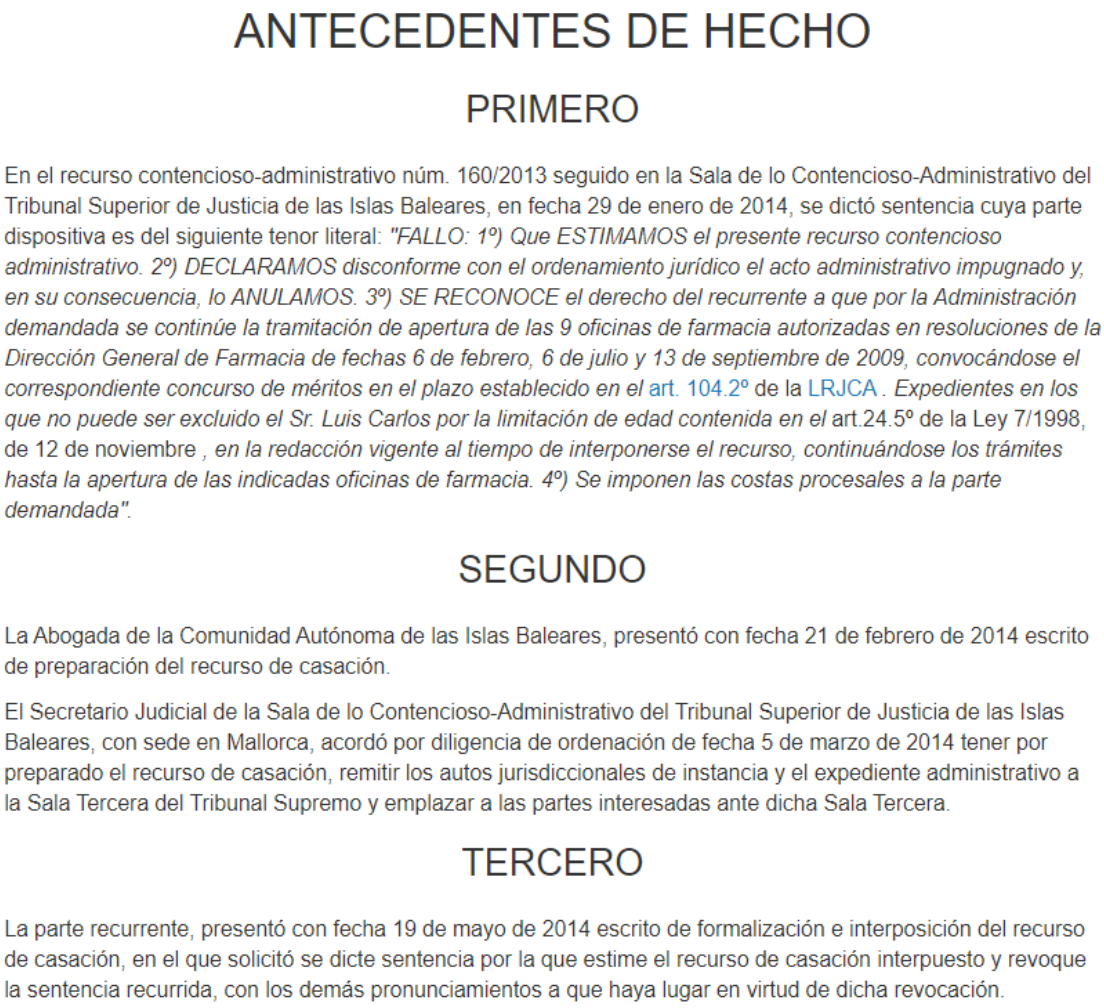
D. Jesús Cudero Blas

D. Ángel Ramón Arozamena Laso

Figura 10.4: *Ejemplo de sección de extra. Extraída de vLex*

10.4 Antecedentes de hecho

Se explica los sucesos que deriva al presente recurso de casación, la razón de insatisfacción del recurrente a la resolución por el tribunal anterior, acontecimientos que conlleva al conflicto, actuación e instrucción del tribunal superior.



ANTECEDENTES DE HECHO

PRIMERO

En el recurso contencioso-administrativo núm. 160/2013 seguido en la Sala de lo Contencioso-Administrativo del Tribunal Superior de Justicia de las Islas Baleares, en fecha 29 de enero de 2014, se dictó sentencia cuya parte dispositiva es del siguiente tenor literal: *"FALLO: 1º) Que ESTIMAMOS el presente recurso contencioso administrativo. 2º) DECLARAMOS disconforme con el ordenamiento jurídico el acto administrativo impugnado y, en su consecuencia, lo ANULAMOS. 3º) SE RECONOCE el derecho del recurrente a que por la Administración demandada se continúe la tramitación de apertura de las 9 oficinas de farmacia autorizadas en resoluciones de la Dirección General de Farmacia de fechas 6 de febrero, 6 de julio y 13 de septiembre de 2009, convocándose el correspondiente concurso de méritos en el plazo establecido en el* art. 104.2º de la LRJCA . *Expedientes en los que no puede ser excluido el Sr. Luis Carlos por la limitación de edad contenida en el* art.24.5º de la Ley 7/1998, de 12 de noviembre , *en la redacción vigente al tiempo de interponerse el recurso, continuándose los trámites hasta la apertura de las indicadas oficinas de farmacia. 4º) Se imponen las costas procesales a la parte demandada"*.

SEGUNDO

La Abogada de la Comunidad Autónoma de las Islas Baleares, presentó con fecha 21 de febrero de 2014 escrito de preparación del recurso de casación.

El Secretario Judicial de la Sala de lo Contencioso-Administrativo del Tribunal Superior de Justicia de las Islas Baleares, con sede en Mallorca, acordó por diligencia de ordenación de fecha 5 de marzo de 2014 tener por preparado el recurso de casación, remitir los autos jurisdiccionales de instancia y el expediente administrativo a la Sala Tercera del Tribunal Supremo y emplazar a las partes interesadas ante dicha Sala Tercera.

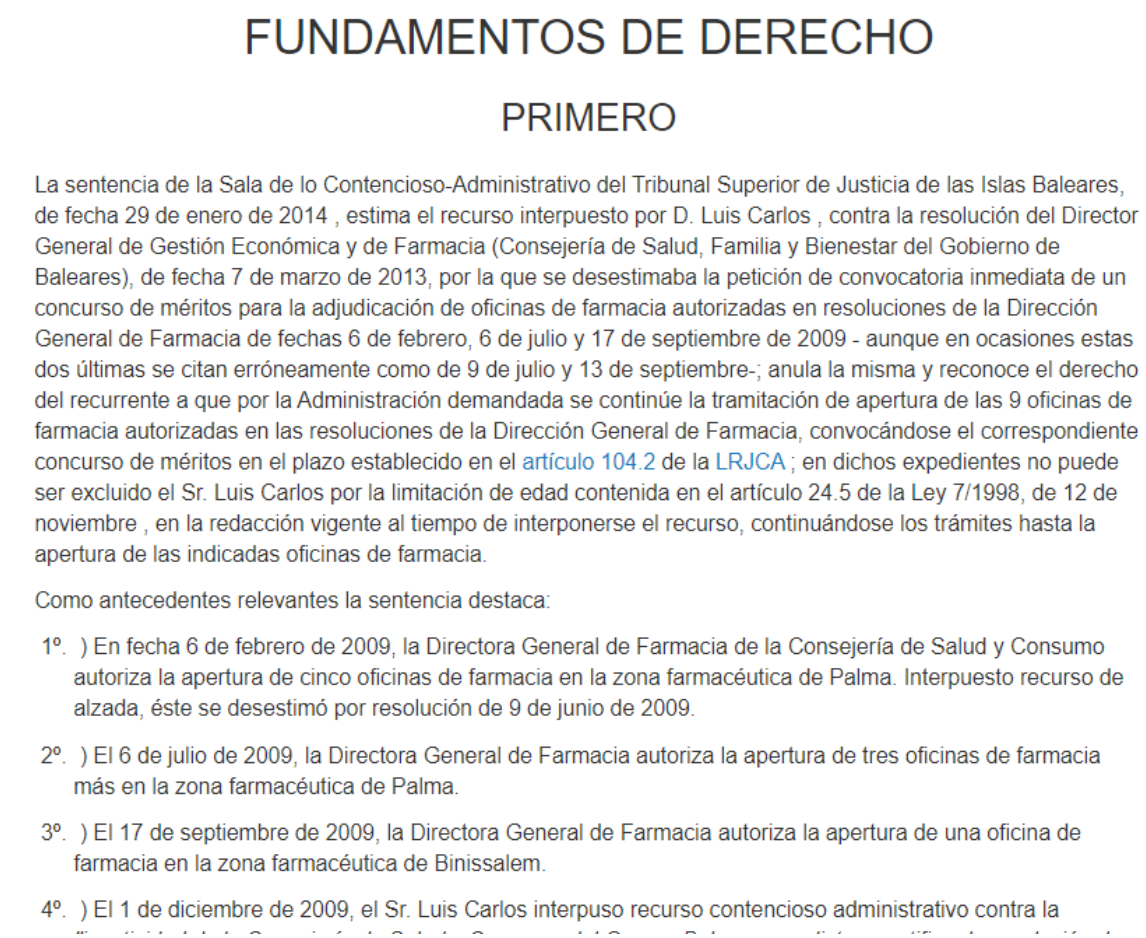
TERCERO

La parte recurrente, presentó con fecha 19 de mayo de 2014 escrito de formalización e interposición del recurso de casación, en el que solicitó se dicte sentencia por la que estime el recurso de casación interpuesto y revoque la sentencia recurrida, con los demás pronunciamientos a que haya lugar en virtud de dicha revocación.

Figura 10.5: *Ejemplo de sección de antecedentes de hecho. Extraída de vLex*

10.5 Fundamentos de derecho

En esta parte se detalla la legislación que recurre el demandante para el litigio, citando las normativas o artículos de la ley que se incumple en los antecedentes de hecho, así como también aportando información detallada de dicha normativa.



FUNDAMENTOS DE DERECHO

PRIMERO

La sentencia de la Sala de lo Contencioso-Administrativo del Tribunal Superior de Justicia de las Islas Baleares, de fecha 29 de enero de 2014 , estima el recurso interpuesto por D. Luis Carlos , contra la resolución del Director General de Gestión Económica y de Farmacia (Consejería de Salud, Familia y Bienestar del Gobierno de Baleares), de fecha 7 de marzo de 2013, por la que se desestimaba la petición de convocatoria inmediata de un concurso de méritos para la adjudicación de oficinas de farmacia autorizadas en resoluciones de la Dirección General de Farmacia de fechas 6 de febrero, 6 de julio y 17 de septiembre de 2009 - aunque en ocasiones estas dos últimas se citan erróneamente como de 9 de julio y 13 de septiembre-; anula la misma y reconoce el derecho del recurrente a que por la Administración demandada se continúe la tramitación de apertura de las 9 oficinas de farmacia autorizadas en las resoluciones de la Dirección General de Farmacia, convocándose el correspondiente concurso de méritos en el plazo establecido en el artículo 104.2 de la LRJCA ; en dichos expedientes no puede ser excluido el Sr. Luis Carlos por la limitación de edad contenida en el artículo 24.5 de la Ley 7/1998, de 12 de noviembre , en la redacción vigente al tiempo de interponerse el recurso, continuándose los trámites hasta la apertura de las indicadas oficinas de farmacia.

Como antecedentes relevantes la sentencia destaca:

1º.) En fecha 6 de febrero de 2009, la Directora General de Farmacia de la Consejería de Salud y Consumo autoriza la apertura de cinco oficinas de farmacia en la zona farmacéutica de Palma. Interpuesto recurso de alzada, éste se desestimó por resolución de 9 de junio de 2009.

2º.) El 6 de julio de 2009, la Directora General de Farmacia autoriza la apertura de tres oficinas de farmacia más en la zona farmacéutica de Palma.

3º.) El 17 de septiembre de 2009, la Directora General de Farmacia autoriza la apertura de una oficina de farmacia en la zona farmacéutica de Binissalem.

4º.) El 1 de diciembre de 2009, el Sr. Luis Carlos interpuso recurso contencioso administrativo contra la

Figura 10.6: *Ejemplo de sección de fundamentos de derecho. Extraída de vLex*

10.6 Fallo

El fallo es la decisión final del tribunal, la resolución puede ser favorable, no favorable o parcial al recurrente. El tribunal también tiene derecho a aplicar costas procesales a las dos partes.

FALLAMOS

No ha lugar al recurso de casación interpuesto por la COMUNIDAD AUTÓNOMA DE LAS ISLAS BALEARES , contra la sentencia de la Sala de lo Contencioso-Administrativo del Tribunal Superior de Justicia de las Islas Baleares, de fecha 29 de enero de 2014, dictada en el recurso núm. 160/2013 , contra resolución del Director General de Gestión Económica y de Farmacia, de fecha 7 de marzo de 2013, por la que se desestimaba la petición de convocatoria inmediata de un concurso de méritos para la adjudicación de oficinas de farmacia autorizadas en resoluciones de la Dirección General de fechas 6 de febrero, 6 de julio y 17 de septiembre de 2009. Con imposición de las costas a la parte recurrente, con el límite que fijamos en el último fundamento de derecho de esta sentencia.

Así por esta nuestra sentencia, que deberá insertarse en la Colección Legislativa lo pronunciamos, mandamos y firmamos

Segundo Menéndez Pérez María del Pilar Teso Gamella

José Luis Requero Ibáñez Jesús Cudero Blas

Ángel Ramón Arozamena Laso

PUBLICACIÓN.- Leída y publicada ha sido la anterior sentencia por el Excmo. Sr. Magistrado Ponente D. Ángel Ramón Arozamena Laso, estando la Sala celebrando audiencia pública en el mismo día de su fecha, de lo que, como Letrado de la Administración de Justicia, certifico.

Figura 10.7: *Ejemplo de sección de fallo. Extraída de vLex*

10.7 Policy Issue

Se refiere al tema de política pública que se trata la sentencia. `Comparative Agendas Project`[2] que se dedican a la recopilación y organización de datos sobre la política en todo el mundo, tienen codificado los temas de la política español en 23 códigos y 243 subcódigos según la legislación que se trata. Disponen en la página web `https://www.comparativeagendas.net/spain` un fichero en formato `excel` que se usará en el proyecto para la creación de una tabla en la base de datos local, con el propósito de identificar el tema político según la ley que se litiga en la sentencia. La razón de no usar directamente el `excel` para esta tarea es porque las BDs proporcionan una forma muy sencilla de realizar búsquedas mediante *queries* de tipo *select*.

[2]https://www.comparativeagendas.net/

Capítulo 11

Tecnologías aplicadas

En este capítulo se explicará las herramientas o librerías que se han usado en este proyecto así como también la base teórica para entenderlo.

11.1 Lenguaje de programación

La selección del lenguaje de programación es una decisión importante antes de realizar un proyecto, pues puede influenciar mucho al progreso de tal. Existen varios factores a tener en cuenta a la hora de elegir el lenguaje adecuado:

- El conocimiento previo y dominio sobre el lenguaje, que afecta principalmente a la eficiencia de programar.
- Características del lenguaje y paradigmas, si es orientado a objetos, funcional, el tipado... depende de la necesidad del proyecto.
- La velocidad de ejecución, que en mayor medida tiene que ver con que si es un lenguaje interpretado o compilado. Los lenguajes compilados (por ejemplo C++) tienen una gran ventaja en cuanto a eficiencia de ejecución respecto a los interpretados (como Python), necesitan un proceso de compilación previo donde realizan parte de trabajos que los lenguajes interpretados tienen que llevar a cabo en tiempo de ejecución.
- La sencillez del sintaxis y legibilidad del código siempre es plus, pero en un proyecto con cierta profesionalidad no es un aspecto muy relevante.

- Cantidad y calidad de librerías disponible. Que puede ser un aspecto determinativo, ya que desarrollar un proyecto totalmente desde cero no es una opción viable por la restricción de tiempo y complejidad.

Pero en el caso de este trabajo no hay demasiada exigencia en cuanto a la selección del lenguaje, los lenguajes de programación mas trabajados durante la carrera tanto C++ (PRO1, PRO2) como Java (PROP) pueden ser candidatos perfectamente. El paradigma orientado a objetos los hace adecuado para proyectos de este tipo, y además de esto, existe una gran comunidad de usuarios que implica mayor facilidad de encontrar soluciones a problemas que pueda surgir.

Sin embargo al final se decidió usar Python, la razón principal es por la existencia de librerías para procesamiento de texto disponible, por ejemplo NLTK (Natural Languaje Toolkit), spaCy... Y la facilidad de instalación siempre es una ventaja, con un simple comando: *pip install* ya lo tenemos disponible para integrarlo al programa. Por otra parte, la inexperiencia en Python (aunque se ha tocado un poco en LP[1]) no es un problema muy grande dado que tiene una sintaxis relativamente fácil, y el dominio de otros lenguajes siempre ayuda en el proceso de aprendizaje.

11.1.1 Python

Es un lenguaje creado por el programador holandés `Guido van Rossum` en el año 1991, la intención era diseñar un lenguaje de propósito general que haga énfasis en la legibilidad del código y simplicidad de sintaxis, esto implica que tiene una curva de aprendizaje baja y esté orientado a un amplio público, desde estudiantes de secundaria o usuarios sin ninguna experiencia de programación a profesionales del sector informático.
El soporte al multiparadigma como orientación a objetos y programación funcional lo hace competente frente otros lenguajes para proyectos de pequeña a mediana escala. Por otra parte, la disponibilidad de potentes librerías para operaciones matemáticas (`numPy`, `pandas`) en combinación con otras librerías como `tensorflow`, `scikit-learn` y `keras` lo hace perfecto para análisis de datos y aprendizaje automático. Todo esto hace que Python sea uno de los lenguajes más usados hoy en día, y tenga una de las comunidades de usuario más activas.

Como usuario de C++, en la transición a Python me he encontrado con diferencias bastante notables. Para empezar el *scoping* en Python es local, para modificar una variable global dentro de una función se ha de especificar implícitamente mediante

[1]Llenguatge de Programació

la directiva `global`. Por el otro lado los bucles en Python te permite iterar directamente sobre una lista o diccionario con una sola linea de codigo: `for i in list: ...` esta sencillez no la encontramos en C++, pero a la vez también perdemos la posibilidad de usar el típico `for (int i = 0; i <n; i++)`. La indentación no solo tiene sentido estético, sino que forma parte de la sintaxis, pues indica bloques de código, sustituyendo así al {...} de otros lenguajes.
Otro problema que me he dado cuenta después de un largo proceso de *debugging* es que en Python cuando una variable se declara dentro de la clase esta se comparte entre todas las instancias de dicha clase, a estas variables se les llaman `class variable`, un concepto parecido a variables estáticas. Este problema se debe principalmente al desconocimiento sobre este lenguaje, que es también una de las razones que decidí usar Python, aprender sobre lo que se desconoce.
Finalmente, al ser un lenguaje *scripting*(e interpretado) la diferencia de velocidad es notable respecto a C++, pero en nuestro caso los procesos más costos en tiempo son métodos `HTTP Get/Post` para descargar sentencias, que principalmente dependen de la conexión y velocidad de transferencia entre el cliente y servidor más que el algoritmo en si, por lo que la influencia del lenguaje en este aspecto es mínima.

11.2 Interfaz gráfica

La interfaz gráfica es una alternativa a la típica interfaz de texto y líneas de comandos de la consola, pues hace que la interacción entre el usuario y la máquina sea de una forma más fácil e intuitiva usando elementos gráficos. Incluso usuarios sin conocimiento previo sobre el software puede aprender a manejarlo en un corto periodo de tiempo, esta facilidad de uso está muy relacionado con el diseño de la interfaz. Actualmente los GUI (Graphic User Interface) están presentes en todo tipo de software, formando una parte esencial de este.

11.2.1 pySimpleGui

En la asignatura de IDI [2] nos enseñaron a usar `Qt` para crear la GUI, en PROP[3] aprendimos a usar `JavaFX`. Ambos comparten características similares, disponiendo de una herramienta para crear vistas "arrastrando" elementos gráficos (botones, layouts, tablas...) y el código entonces permite controlar el comportamiento de estos, esta combinación resulta en una sencillez a la hora de diseñar la vista y un alto nivel

[2]Introducció i diseny d'interficie
[3]Projecte de Programació

de personalización.
Sin embargo, al final decidí usar `pySimpleGUI`, la razón no es que no existan frameworks parecidos a los mencionados anteriormente, sinó mas por la curiosidad. Esta librería proclama poder implementar GUIs con una sola línea de código, algo muy poco habitual. Para este proyecto, la necesidad de GUI es clara, ya que está destinado a investigadores de derecho y ciencia política, que lo más probable es que no tengan experiencia en programación, por lo que usar la consola no es una opción viable. Sin embargo al final, no se usará elementos muy avanzados para el diseño, con unas cuantas vistas simples será suficiente (el tiempo es un factor que lo limita), por lo que el riesgo de probar este framework nuevo no es tan grande.

Ejemplos de pySimpleGUI (1)

```
1   import PySimpleGUI as sg
2
3   sg.theme('Light Blue 2')
4   layout = [[sg.Text('Enter 2 files to comare')],
5             [sg.Text('File 1', size=(8, 1)), sg.Input(), sg.FileBrowse()],
6             [sg.Text('File 2', size=(8, 1)), sg.Input(), sg.FileBrowse()],
7             [sg.Submit(), sg.Cancel()]]
8   window = sg.Window('File Compare', layout)
9   event, values = window.read()
10  window.close()
11  print(f'You clicked {event}')
12  print(f'You chose filenames {values[0]} and {values[1]}')
```

Índice de códigos 11.1: Ejemplo 1 de pySimpleGUI

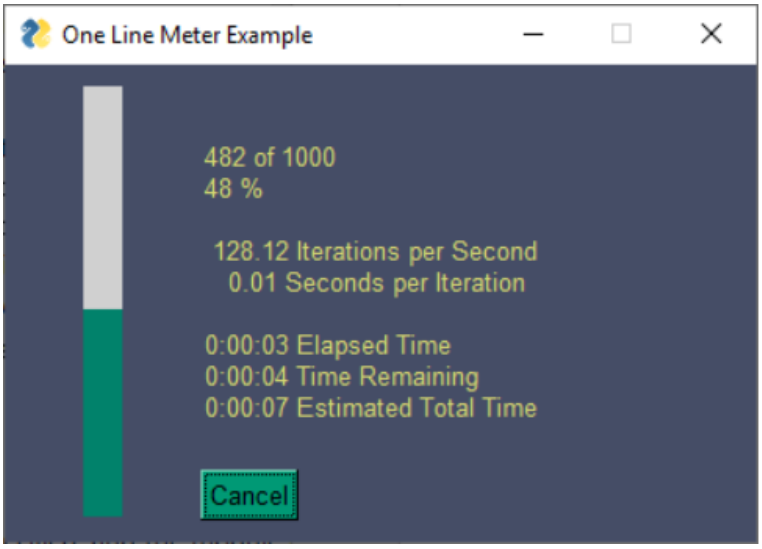


Figura 11.1: *Ejemplo 1 de pySimpleGUI. Extraído del cookbook oficial*

Ejemplos de pySimpleGUI (2)

```
1    import PySimpleGUI as sg
2
3    sg.theme('Dark Blue 8')
4    for i in range(1000):   # this is your "work loop" that you want to monitor
5        sg.OneLineProgressMeter('One Line Meter Example', i + 1, 1000, 'key')
```

Índice de códigos 11.2: Ejemplo 2 de pySimpleGUI

Figura 11.2: *Ejemplo 2 de pySimpleGUI. Extraído del cookbook oficial*

11.3 Base de datos

Un sistema para almacenar y gestionar datos es muy habitual en software de todo tipo, y las bases de datos es justamente una forma de organizar datos sistemáticamente.

En este proyecto, la necesidad es muy clara, queremos hacer Big Data sobre las sentencias judiciales , por lo que tenemos un sistema eficiente de guardar la información extraída. Como alternativas, se había pensado en el formato `json` dado que minimiza la memoria requerida para almacenamiento con la compresión de espacios blancos, pero había también la necesidad de hacer constantemente comprobaciones de si una sentencia ya existe en el sistema, con `json` si se tiene los documentos guardados en orden del identificador se puede conseguir hacer operaciones tipo *insert, select* en tiempo $O(\log n)$ con búsqueda binaria. Sin embargo este trabajo extra de ordenación de los documentos ya lo tenemos hecho con las bases de datos, la estructura interna habitualmente es un `árbol B+` guardando en los nodos pares $clave, puntero$, así optimizando la memoria y consiguiendo operaciones de inserción y borrado eficientes.

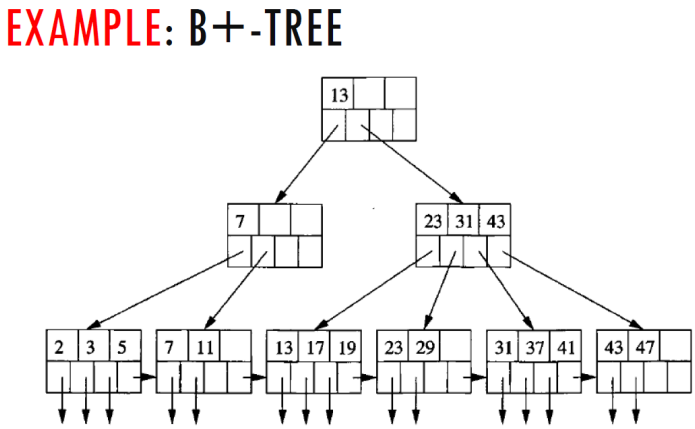


Figura 11.3: *Ejemplo de árbol B+. Extraído de Goodle Images*

SQL(Structured Query Language) es el lenguaje para el control de las BDs, permite realizar operaciones sobre esta de forma intuitiva. La sintaxis más simple para operaciones sobre filas sigue el estilo: `select/update/delete {columns} from {tableName} ...` pudiendo añadir filtros con `where`. Y la operación sobre tabla es de tipo: `create table {tableName} ( column1 datatype, column2 datatype, column3 datatype, .... )`

Los bases de datos de forma general se pueden clasificar en BD relacional y `NoSQL`, la diferencia es que en los de `NoSQL` los datos son representados como conjuntos de documentos (cada documento puede contener atributos diferentes) mientras que en `SQL` los datos están en tablas predefinidas fijando estrictamente el número de

columnas y sus respectivo tipo de valor, consecuentemente los *queries* SQL son más eficientes. En el proyecto los campos que hay que extraer están ya definidos en la memoria, por lo que no hay necesidad de usar base de datos NoSQL.

11.3.1 MySQL

`MySQL` es uno de los `RDBMS`[4] open-source más usados en el mercado, usa un modelo cliente-servidor que para poder acceder a esta BD desde el programa se ha de usar un driver *mysql.connector*. Y dispone de una consola de comandos `MySQL Shell` que permite ejecutar *queries* fuera del programa en el caso de que se necesita hacer comprobaciones de la base de datos.

Como alternativa en Python existen otras `RDBMS` como `SQLite` o `PostgreSQL`, todas son opciones válidas. De hecho, en realidad no hay una razón especial de porqué se ha elegido `MySQL`.

11.4 Html

`Html`(Hypertext Markup Language) es el formato estándar para representar páginas webs. Los ficheros de este tipo, está estructurado mediante *tags* que permiten definir el tipo de elemento gráfico a renderizar por los navegadores, que en combinación con `CSS`(Cascading Style Sheets) le hace dotar de una gran libertad de personalización en cuanto a elementos visuales. De este modo se forma una estructura arbólea de tags. A diferencia de otros lenguajes Markup como `XML`, los tags en html tienen mayor flexibilidad, es decir, pueden no estar perfectamente estructurado.
Por otra parte, html da soporte a lenguajes scripting como `JavaScript` para el control de la lógica y el flujo de la página web.

11.4.1 BeautifulSoup

Las sentencias que se descargan de vLex son de este formato, lo ideal es pasar todo el contenido a texto plano (*plain text*) para mayor facilidad de manejo. Para este propósito, se usa la librería `beautifulSoup` para simplificar el trabajo de *parsing*. Que permite cargar el fichero html como un objeto `soup` y realizar búsquedas y recorrido de tags de forma sencilla, permitiendo añadir filtros como id o clase: `soup.find("div", id = "main-content")`.

[4]Relational Database Management System

11.5 Expresión regular

Todo lenguaje finito es regular, y por el teorema de Kleene, todo lenguaje regular se puede representar mediante autómatas de estado finito (DFA[5], NFA[6]). En la teoría de lenguaje formal, según la jerarquía de Chomsky está clasificado como gramáticas de tipo 3, el subconjunto con menor expresividad.

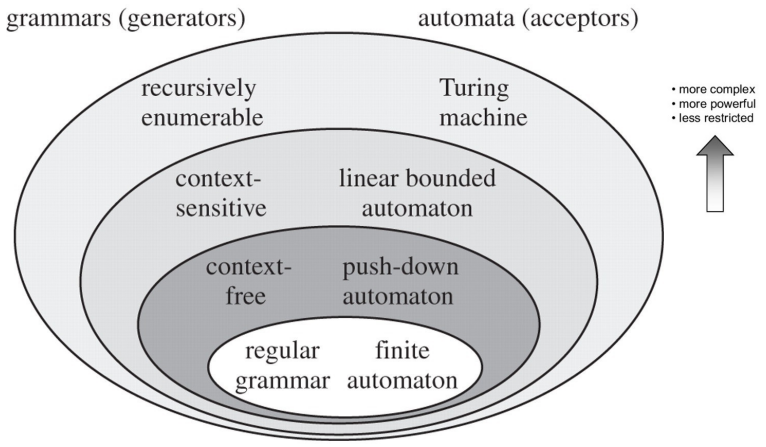


Figura 11.4: *Jerarquía de Chomsky. Extraído de Goodle Images*

Pues, la expresión regular es una forma para describir los lenguajes regulares, formalmente definido como:

- Λ (cadena vacía) y Φ (lenguaje vacío) son regulares.
- Para cada símbolo a del alfabeto Σ, a es regular.
- Si r_1 y r_2 son regulares, también lo son sus respectivos concatenación y unión: $(r_1 + r_2)$, $(r_1 \cdot r_2)$.
- Si r es regular, (r$*$) también lo es.
- Todas las expresiones sobre Σ obtenidas aplicando los pasos anteriores.

Dicho esto, la expresión regular es una herramienta potente para encontrar patrones en textos, hoy en día está integrado en la mayoría de lenguajes de programación con extensiones que incluso le dota la capacidad de expresar lenguajes no regulares.

[5]Deterministic finite automata
[6]Non deterministic finite automata

11.5.1 re

En Python las operaciones con expresión regular está incluido en el módulo re.py, donde podemos formar una cadena de carácteres para el *matching* con combinaciones de letras del alfabeto, dígitos o símbolos especiales que representan conjuntos de estos:

Character	Description	Example
\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"
\b	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\Bain" r"ain\B"
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

Figura 11.5: *Símbolos de re. Extraído de w3schools*

Y con la posibilidad de crear patrones de búsqueda más avanzado mediante los siguientes operadores:

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"world$"
*	Zero or more occurrences	"aix*"
+	One or more occurrences	"aix+"
{}	Exactly the specified number of occurrences	"al{2}"
|	Either or	"falls|stays"
()	Capture and group	

Figura 11.6: *Operadores de re. Extraído de w3schools*

Finalmente, la búsqueda se realiza con los métodos `re.search` o `re.match` sobre la expresión definida, pudiendo etiquetar y clasificar los fragmentos de texto que han hecho *match* mediante la directiva `(?P<etiqueta>...)`. Por ejemplo con la expresión `(Sr.|Sra.)(?P<name>[A-Z][a-z]*)` podemos encontrar nombres de personas, que es una secuencia seguida de Señor o Señora empezando por una letra mayúscula y luego un número indefindo de minúsculas. Si se ha encontrado algún *match*, entonces el objeto devuelto por la búsqueda permite recuperar el texto accediéndolo como un diccionario, mediante la etiqueta que se había definido: `matchText = res.group("name")`.

11.6 Procesamiento del lenguaje natural

El procesamiento de lenguaje natural o NLP (Natural Languaje Processing) en inglés, es un subcampo de la informática, lingüística e inteligencia artificial que se trata de la interacción entre máquina y humano mediante el lenguaje natural (lenguaje humano), en concreto son métodos para procesar y analizar datos lingüísticos no estructurados en un formato que sea entendible para máquinas.
Dado la ambigüedad de los lenguajes humanos y su alta abstracción, NLP nunca ha sido un problema fácil de tratar. Su origen se debe a los años 1950 con el `experimento Georgetown` donde se quería hacer traducción automática de 60 frases de ruso a inglés. En esa etapa con la publicación de "Computing Machinery and Intelligence" de Alan Turing (lo que posteriormente se conoce como `Turing Test` para

comprobar si la máquina es lo suficientemente inteligente) surgió una alta tendencia hacia la inteligencia artificial, la gente creía que la IA podía resolver cualquier tipo de problema, de hecho, los autores del `experimento Georgetown` confiaban que la traducción automática será un problema resuelto dentro de 5 años, pero resulta que el progreso es mucho más lento de lo que se imaginaban. La razón puede ser varias, pero la más intuitiva es por la limitación del *hardware* de aquella época. Dado que el rendimiento de las técnicas de IA o *machine learning* depende mucho del tamaño del *dataset* de entrada para el entrenamiento, de modo que la memoria es factor determinativo. Sin embargo, actualmente con el avance de la técnica y evolución del `hardware`, estos métodos ya lo son suficientemente maduros y tienen una amplia aplicación en la industria, por ejemplo el asistente virtual `Siri` de Apple, el traductor `Goodle Translate`, corrector de ortografía que tienen integrado los editores de documento como `Microsoft Word`...etc

De NLP deriva una multitud de técnicas, donde muchas de estas tiene aplicación directa en el mundo real, mientras que otras forma parte del flujo del procesamiento o simplemente son subtareas.

Tokenization

Forma parte del flujo típico de NLP, la tarea es dividir el texto de entrada en piezas, llamada *tokens*. La identificación de tokens se puede aproximar mediante separadores como espacio o signo de puntuación, de forma que cada palabra es un *token*. Una mejora para este proceso es lo que se conoce como `stopword removal`, mediante lo cual se elimina las palabras que tienen poco significado, tales como preposición, determinante y adverbio, por ejemplo: a, en, de, con, el, la, entonces, mucho ... Con este proceso se llega a reducir hasta un 40 % del tamaño final del conjunto de tokens, por lo que su efecto sobre la eficiencia es crucial.

Word Sense Disambiguation

Actualmente es una área muy activa de investigación, la tarea es averiguar el significado que tiene una palabra dentro del contexto (frase). El cerebro humano trabaja de una forma muy eficiente y precisa en este aspecto, dado que se piensa que el lenguaje natural ha evolucionado conforme a como funciona las neuronas. Actualmente la mejor aproximación a este problema es mediante *machine learning*, con aprendizaje no supervisado se entrena un clasificador sobre un corpus de palabras etiquetadas manualmente, el resultado de una clasificación con significado más general consigue

un 90 % de acierto en inglés, pero para una clasificación más fina apenas se consigue un 60 %.

Stemming y Lemmatizing

Son dos métodos alternativos a *tokenization* en cuanto a división del texto en piezas. La diferencia está en que estos métodos consideran el significado de las palabras a la hora de realizar *splitting*, por ejemplo no tiene sentido separar "Universidad Politécnica de Cataluña" en *tokens* diferentes, ya que su valor semántico se debe al conjunto de estas palabras. La salida por lo general se convierte a minúscula y en la forma primitiva de la palabra, para así conservar la unidad de significado, por ejemplo "cantar" y "cantante" se identificarán con el mismo *token*. De esta forma simplificando el trabajo de procesos posteriores sobre análisis semántico.

`Lemmatizing` reduce las palabras a su raíz lingüística (*lemma*), aunque algunas veces es suficiente con analizar los sufijos, por lo general no es una tarea fácil de implementar, se ha de tener bastante conocimiento sobre la morfología de las palabras del lenguaje.

`Stemming` es un método para aproximar `Lemmatizing` dado que éste es un proceso costoso. `Stemming` en cambio es mucho más eficiente y el resultado es lo suficientemente bueno. Los algoritmos más famosos son `Porter Stemmer` y `Snowball Stemmer`, que se basan en reglas para la eliminación de sufijos, prefijos o reescritura comleta de la palabra. Porter, por ejemplo tiene 60 reglas y una distribución en 5 fases, Snowball consigue un poco de mejora pero sigue básicamente el mismo principio.

PoS Tagging

Como complemento a los procesos anteriores, con `PoS tagging` clasificamos los *tokens* según su categoría gramatical. Existen dos tipos de algoritmos para esta tarea. Por un lado los que se basan en técnicas estocásticas (probabilística) que asume que se conoce todas las palabras y que estas pertenece a un conjunto finito de categorías, de tal forma se puede calcular la probabilidad de que pertenezca a una mediante diccionarios o análisis morfológico. Por otra parte, están los algoritmos basados en reglas, que aunque no consigan tanta precisión como los métodos anteriores, la ventaja es que no requieren mucha memoria y el resultado es decente, un ejemplo es `E. Brill's tagger`, uno de los primeros *taggers* y el más usado en inglés.

Named Entity Clasification

NEC forma una parte esencial de `Information Extraction`, que es uno de los objetivos de este proyecto. El trabajo de esta tarea es identificar las entidades que aparece en un texto y clasificarlas en diferentes clases como: organización, persona, lugar... donde el nivel de abstracción es un factor que determina la dificultad del problema. Por ejemplo para una clasificación más fina, de las entidades etiquetadas como organización lo podemos subdividir en ONG, empresa, departamentos del gobierno...
Existen dos aproximaciones para NEC, la más simple y directa es un sistema basado en reglas, por ejemplo una regla de comprobar si empieza por mayúscula ya puede cubrir gran parte de los casos. Pero la mejor opción, si hay recursos suficientes, es entrenar un clasificador con un corpus de contexto similar al texto que se procesará, de esta forma se sacrifica la universalidad para mejor rendimiento sobre un tópico específico. El formato IOB (Inside-Outside-Beginning) ya es un estándar para tareas de *chunking* como NEC, si un *token* es etiquetado como B quiere decir que es el comienzo de una entidad, si es I entonces forma parte de la entidad anterior y si es O implica que no es una entidad.

Bag of Words

Es un modelo muy popular para representar documentos mediante un vector. Este vector (bag) contiene todas las palabras (words) que aparece en el documentos así como su frecuencia, de modo que es un histograma de palabras. Normalmente, no se usa todo el bag, sino se preprocesa eliminando una parte de las palabras , como por ejemplo *stop words*. Teniendo en cuenta algunas propiedades de los corpus en lenguaje natural, como:

1. `La ley de Zipf`, una ley empírica que estipula que la frecuencia de palabras en un texto sigue una ley potencial:

   $$fr(w_i) = \frac{c}{(i+b)^a}$$

   Siendo w_i la i-ésima palabra más frecuente del texto. Los parámetros a, b y c son dependientes al tipo de corpus y estudios recientes afirma que $a \approx 1$. Por lo que la segunda palabra más frecuente tiene aproximadamente la mitad de frecuencia que la primera palabra.

2. `La ley de Heap`, que describe que el número de palabras diferentes en un texto sigue una función polinómica de grado inferior a 1. Sea d el número de palabras

diferentes, y N la longitud (número de palabra total) del texto, tenemos que:

$$\log d = k_1 + \beta * \log N$$

Dicho esto, si se realiza recorte sobre las palabras menos frecuentes, los documentos resultarán muy similares. Mientras que si se recorta las palabras más frecuentes, se pierde gran cantidad de información de las palabras.
Este modelo es muy usado en los problemas de clasificación de documento, en este caso, por ejemplo para clasificar si una sentencia es favorable o no. Dado que el bag sirve como `feature vector` para tanto `supervised learning` (clasificación) o `unsupervised learning` (clustering).

Jaccard Similarity

Es un método para estimar la similitud entre dos documentos, sea A i B los conjuntos que representan los dos documentos, entonces tenemos:

$$J(A, B) = \frac{A \cap B}{A \cup B}$$

Es decir, la proporción de partes iguales de A i B sobre el total. La representación de documento mediante conjuntos se puede conseguir de distintos modo, como por ejemplos el `bag of words` hablado anteriormente y el `k-shingles`.

Cosine Similarity

La representación de un documento mediante un vector de pesos (weight vector) `tf-idf`, tiene en cuenta la influencia de la frecuencia de la palabra sobre la comparación de documentos. Cuando más frecuente es una palabra, menos discriminante es, por lo tanto se le asignará menos peso en la comparación. (José Luis Balcázar 2018a, página 16) Entonces un documento d se representa como:

$$d = [w_{d,1}, .., w_{d,2}, ..., w_{d,n}]$$

Donde cada peso w es el producto de dos términos:

$$w_{d,i} = tf_{d,i} * idf_i$$

La frecuencia de término es:

$$tf_{d,i} = \frac{f_{d,i}}{max_j * f_{d,j}}, \quad \text{donde } f_{d,i} \text{ es la frecuencia del término } i \text{ en el documento } d$$

Mientras que la frecuencia inversa de documento es:

$$idf_i = \log_2 \frac{D}{df_i}, \quad \begin{array}{l} \text{donde } D \text{ es el número total de documentos} \\ df_i \text{ es el número de documentos que contiene el término } i \end{array}$$

Dicho esto, se puede calcular la similitud coseno entre dos documentos:

$$sim(d1, d2) = \frac{d1 \cdot d2}{\mid d1 \mid \cdot \mid d2 \mid}$$

$$\text{donde } v \cdot w = \sum_i v_i \cdot w_i, \mid v \mid = \sqrt{v \cdot w} = \sqrt{\sum_i v_i^2}$$

11.6.1 Freeling

Es una librería *open-source* totalmente implementada en C++ y desarrollada por el centro de investigación TALP[7] de la UPC, fue un proyecto creado y dirigido actualmente por el director del presente TFG. La primera versión fue lanzada en el año 2003, y hoy en día ya está disponible en la versión 4.1 distribuido bajo la licencia AGPL[8] donde el usuario tiene derecho de copia y uso del *software*, sin embargo el uso comercial o distribución de proyecto que incluya `Freeling` ha de estar bajo la misma licencia AGPL, en caso contrario se puede comprar una licencia dual que lo excluye de distribuirlo en AGPL.
`Freeling` permite realizar una gran variedad de funciones de análisis lingüístico, en una amplia selección de idiomas:

[7] Centro de Tecnologias y Aplicaciones del Lenguaje y el Habla
[8] Affero GNU General Public License

	as	ca	cy	de	en	es	fr	gl	hr	it	nb	pt	ru	sl
Tokenization	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Sentence splitting	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Number detection		X		X	X	X	X	X		X		X	X	
Date detection		X		X	X	X	X	X				X	X	
Morphological dictionary	X	X	X	X	X	X	X	X		X	X	X	X	X
Affix rules	X	X	X	X	X	X	X	X		X	X	X		
Multiword detection	X	X	X		X	X	X	X		X		X		
Basic named entity detection	X	X	X		X	X	X	X		X		X	X	X
B-I-O named entity detection		X			X	X		X				X		
Named Entity Classification		X			X	X						X		
Quantity detection		X			X	X		X				X	X	
PoS tagging	X	X	X	X	X	X	X	X		X	X	X	X	X
Phonetic encoding					X	X								
WN sense annotation		X			X	X	X	X	X					X
UKB sense disambiguation		X			X	X	X	X	X					X
Shallow parsing	X	X			X	X		X				X		
Full/dependency parsing	X	X			X	X		X	X					X
Semantic Role Labelling		X		X	X	X								
Coreference resolution					X	X								

Figura 11.7: *Tabla de funcionalidad de Freeling. Extraído de la página oficial*

En el proyecto, se usará principalmente el módulo NEC de `Freeling`, para identificar entidades y combinarlo con expresión regular para decidir si es la información que se desea extraer. Para este propósito la librería proporciona dos tipo de aproximaciones. La primera, es un módulo llamado `Basic NER` que implementa un sistema basado en reglas, así como detección de mayúsculas teniendo en cuenta algunas palabras funcionales y mayúscula al principio de frases. El otro módulo, `BIO NER` se basa en el entrenamiento de un clasificador para decidir si una palabra es O, B o I de una entidad (formato IOB explicado anteriormente), sin embargo este módulo es más avanzado que `Basic NER` y requiere el uso complementario de otro modulo `Feature Extraction Module` de forma que el clasificador reciba un *feature vector* con el mis-

mo número y tipo de elementos para cada palabra. La diferencia es bastante clara, `Basic NER` es más simple, rápido y menor dependencia del contexto de los texto, consiguiendo un porcentaje de acierto de 85 %; mientras que el otro modulo puede alcanzar por encima de 90 %, pero se ha de tener un corpus manualmente anotado las entidades y la adaptación a otro contexto (por ejemplo cambio de lenguaje) implica un entrenamiento nuevo.

11.6.2 Textserver

`Freeling` está implementado en C++, y aunque se puede integrar proyectos hechos en otro lenguaje se ha de compilarlo desde *source*, siguiendo la instrucción del manual y después de varios intentos y fallos, se decide por usar la plataforma `Textserver` que proporciona TALP. `Textserver` es un *webservice* que permite realizar tareas de procesamiento lingüístico desde servidores remotas, proporciona las mismas funcionalidades que `Freeling` ya que lo tiene integrado, sin embargo tenemos una menor flexibilidad en el sentido de que no es posible modificar su comportamiento ni entrenar clasificadores NEC con un corpus determinado. Pero en realidad, en este caso no es un problema dado que no disponemos de dicho corpus sobre las sentencias judiciales (se habría que extraer y anotar manualmente más de miles de sentencias) y por la limitación de tiempo del TFG, no es una opción viable. Dicho esto, enviar el texto a procesar al servidor mediante `HTTP POST` y recuperar la salida en el formato que queramos: XML, CoNLL o JSON.

```
1     import requests
2     request_data = {'username':user,
3                     'password':pwd,
4                     'text_input':text,
5                     'language':lang,
6                     'output':out,
7                     'interactive':'1' }
8
9   service = "NEC" # service name
10  url = "http://frodo.lsi.upc.edu:8080/TextWS/textservlet/ws/processQuery/"+service
11
12  resp = requests.post(url, files=request_data)
```

Índice de códigos 11.3: Ejemplo de programa con Textserver

El uso de un servicio *cloud* añade un tiempo adicional al proceso de extracción de información debido a la conexión al servidor remoto, el efecto de este tiempo extra es bastante notable, ya que a la práctica es incluso mayor que el tiempo de procesado del texto en `Textserver`.

11.6.3 spaCy

Por la razón anterior, se decide usar la librería `spaCy` para subtareas donde se requiere analizar sobre un fragmento de texto reducido, dado que una tarea tan pequeña no compensa el *overhead* de la conexión al servidor. Y `spaCy` al ser local, no requiere de dicha conexión.
`SpaCy` junto a `NLTK` son las librería más famosas para NLP en Python, se caracteriza por la facilidad de uso e instalación (con *pip*), presentando funcionalidades avanzadas como entrenamiento de modelos estadísticos mediante *Deep Learning* para una gran variedad de problemas de NLP.
La razón por la cual no se decidió realizar todo el proceso de NLP en `spaCy` es que se confía que `Freeling` con varios años de experiencia en el idioma español, tenga mejor rendimiento que otra librería que principalmente se dedica al contexto inglés.

Capítulo 12

Diseño e implementación del proyecto

12.1 Diagrama de casos de uso

Los diagramas de casos de uso es una forma ilustrativa de mostrar toda la funcionalidad del sistema, es la figura siguiente podemos ver todo lo que el usuario puede hacer.

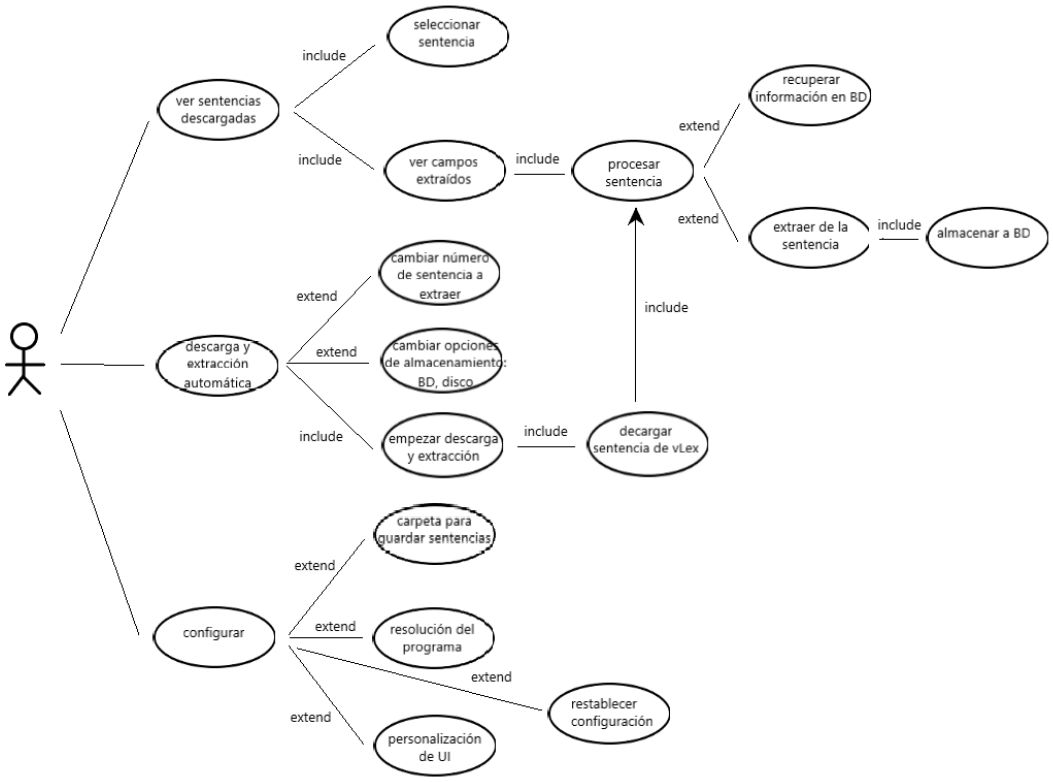


Figura 12.1: *Diagrama casos de uso. Elaboración propia*

12.2 Arquitectura de software

En el proyecto se sigue la arquitectura usada y estudiada en la asignatura de PROP, la programación por capas, en concreto la separación en tres módulos independientes.
Capa de presentación, es la capa que ve el usuario, se encarga del control de la interfaz gráfica así como la tarea de presentar al usuario la información de la capa de dominio de una manera que sea fácil de entender, y captura la entrada del usuario pasándoselo a la capa de dominio.
Capa de dominio (capa de lógica de negocio), en esta capa reside toda la lógica del programa, en nuestro caso se encarga de la descarga y extracción de información de las sentencias, está conectado a la capa de datos y presentación haciendo posible la comunicación entre estas.
Capa de datos, se encarga de gestionar la información almacenada, así como acciones de guardar, eliminar o modificar. Normalmente está formado por controladores de

bases de datos que se comunica con la capa de dominio.
Este tipo de arquitectura tiene la ventaja de que todos estos módulos son independientes, eso quiere decir que pueden ser desarrolladas paralelamente consiguiendo así más eficiencia de trabajo en cuanto se trate de un proyecto en grupo. Otra ventaja es la reusabilidad del código, cambios en una capa no afecta a las otras capas por lo que solamente hace falta hacer modificaciones en módulo deseado.

En el proyecto no se sigue perfectamente esta arquitectura de software, ya que existen algunas conexiones entre la capa de presentación y datos por comodidad a la hora de programar. Pero por lo generar se satisface el principio de diseño por capas.

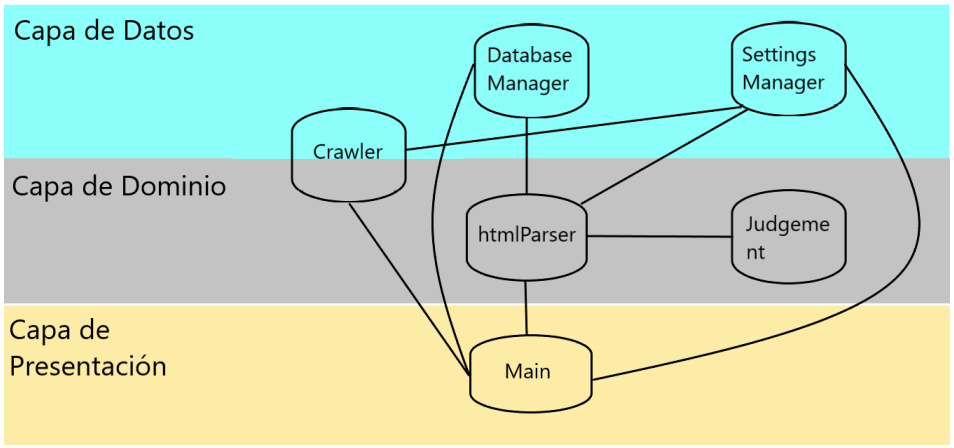


Figura 12.2: *Módulos del sistema. Elaboración propia*

12.3 Flujo de datos

En la figura siguiente podemos observar el flujo de los datos del sistema, el módulo `Crawler` recibe una lista de número de identificación de `vLex` mediante el API que proporciona, y después con los *ids* se consigue descargar las sentencias en formato html de la base de datos de `vLex`. Opcionalmente estas sentencias se almacenan en disco, una carpeta que indicará el usuario, y para la información extraída de las sentencias se almacenará en la base de datos local.

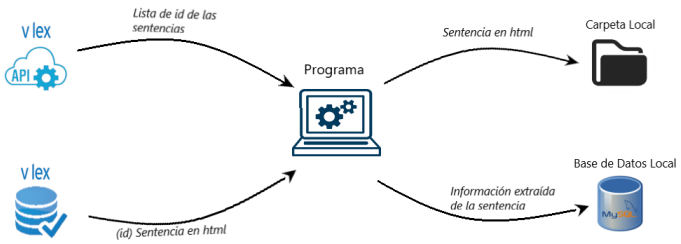


Figura 12.3: *Flujo de datos. Elaboración propia*

12.4 Módulos

A continuación se detallará la implementación de cada módulo.

12.4.1 Main.py

En este módulo se implementa todo el control de la interfaz gráfica así como las operaciones que derivan de ésta. Habitualmente, en otras librerías, hay un controlador para cada vista, sin embargo el funcionamiento de `pySimpleGUI` es diferente, por lo que el `Main.py` se encarga de todo el trabajo.
Para crear una vista o `window` se ha de definir una estructura de tipo lista de `widgets`(botones, textos, imagen...). Como vemos en el caso siguiente de la vista principal, tenemos un *widget* tipo imagen ("jurisprudence.png") que es un logotipo para la aplicación, seguida de cuatro botones, tres para pasar a otras vistas y otro para cerrar la aplicación.

```
1 def createMainLayout():
2     buttonSize = (int(windowSize[0]*perPixel*ar/2),int(windowSize[1]*perPixel/10))
3     buttonPad = (int(windowSize[0]*1/4),1)
4     path = imagesPath + "jurisprudence.png"
5     imSize = (int(windowSize[0]/4), int(windowSize[1]/4))
6     imPad = (int(windowSize[0]*3/8),int(windowSize[1]*1/16))
7     layout = [  [sg.Image(data=get_img_data(path, imSize, True), visible=True, pad =
       imPad)],
8                 [sg.Button('Documentos Descargados', pad = buttonPad, size =
     buttonSize, key = "-downloaded-")],
9                 [sg.Button('Descargar documentos de vLex', pad = buttonPad, size =
     buttonSize, key = "-download-")],
```

```
10                  [sg.Button('Configuracion', pad = buttonPad, size = buttonSize, key
    = "-config-")],
11                  [sg.Button('Salir', pad = buttonPad, size = buttonSize, key = "-exit
    -")]  ]
12
13      return layout
```

Índice de códigos 12.1: Código layout principal

Cada vista ha de estar en un bucle permanente (`while True`) dado que se ha de capturar todas las entradas del usuario como eventos, de los cuales se identifican con la clave (*key*) que tiene asociado a la hora de crear el *widget*, de tal forma se puede decidir la acción a realizar dependiendo de estos. Dicho esto, para salir de la vista o el cerrar el programa (si se trata de la vista en nivel más alta) habrá que hacer un `break` sobre el bucle, mientras que un transición a otra vista de nivel inferior implica crear una ventana (o *layout*) secundario y entrar en otro `while True`. En el programa, en total tenemos 4 vista, la vista de menú principal que aparece cuando se inicia la aplicación, y otras tres que corresponde a las tres funcionalidades principales que vemos en el diagrama de casos de uso. El siguiente fragmento de código muestra como se implementa la interacción entre la vista principal y la de "Ver sentencias descargadas", dado que las otras se implementa similarmente.

```
1  if __name__ == "__main__":
2      layout = createMainLayout()
3      window = sg.Window(title, layout, size = settingsManager.getResolution())
4      actWindow = 1
5      databaseManager.initialize()
6      htmlParser.setPath(settingsManager.getFolder())
7      crawler.setPath(settingsManager.getFolder())
8      while True:
9          event, values = window.read()
10         sg.theme(settingsManager.getTheme())
11         window.size = settingsManager.getResolution()
12         if event in (None, '-exit-'):
13             break
14         # ----------------- Donwloaded Window (2) ----------------------
15         elif event == "-downloaded-":
16             layout2 = createDownloadedLayout()
17             window.Hide()
18             window2 = sg.Window(title, layout2, size = windowSize)
19             actWindow = 2
20             while True:
21                 event2, values2 = window2.read()
22                 if event2 in (None,'-back-'):
23                     actWindow = 1
24                     window2.Close()
25                     window.UnHide()
26                     break
27                 elif event2 in ("-open-","-table-"): # double clicked on row or open
    button clicked
28                     selectedRow = values2["-table-"][0]
29                     id = data[selectedRow][0]
30                     text = ""
```

```
31                      # create a thread to extract the information
32                      thread = threading.Thread(target=getTextFromParser, args = [id],
    daemon=True)
33                      thread.start()
34                      while True: # the main loop shows a loading gif
35                          sg.popup_animated(sg.DEFAULT_BASE64_LOADING_GIF, 'Extracting
    information...', time_between_frames=100)
36                          if not thread.is_alive():
37                              break
38                      sg.popup_animated(None)
39                      sg.popup(text)
```

Índice de códigos 12.2: Código control de vistas

En la línea 32 de código, podemos observar el uso de un thread adicional para realizar el proceso de extracción de la sentencia mediante la función `getTextFromParser(id)`. La razón es que `pySimpleGUI` requiere que el thread principal resida en el bucle para realizar una actualización y control constante sobre la vista, no queremos que el programa se bloquee (en cuanto a GUI) mientras se está realizando un proceso costoso, sino que sea reactivo en todo momento, de modo que no queda más remedio que mandar esta tarea a otro thread secundario.
La comunicación entre threads es mediante variables globales, en este caso cuando el thread secundario haya extraído la información de la sentencia elegida, lo dejará en una variable `text`, el thread principal saldrá del bucle infinito (ya que el thread secundario ya no sigue activo) y mostrará el texto mediante esa variable.

12.4.2 Crawler.py

Este módulo principalmente se encarga de dos tareas, conseguir una lista de ids de las sentencias y descargar la sentencia mediante el id.

Obtener ids

se accede al API de vLex por `http://api.vlex.com/search.json?q=`, donde se puede añadir filtros de búsqueda a "q=...", como por ejemplo palabras claves `http://api.vlex.com/search.json?q=SentenciaSalaContenciosoAdiministrativo`. La función `buildUrl(page)` se encarga de este trabajo, crear el url correcto y añadir todas los filtros y opciones que se han definido en una variable global, el parámetro que recibe es una opción para indicar la página a retornar por vLex, dado que en una página solo puede haber 100 sentencias como máximo, por lo que si se quiere extraer una cantidad mayor se habrá que hacer múltiples accesos modificando la página.

```
1 def buildUrl(page):
2     url = baseUrl;
3     first = True;
```

```
    for opt in options:
        word = opt.replace(" ", spaceSymbol)
        if (first):
            url += word
            first = False;
        else:
            url += "&"+word
    url += "&page="+str(page)+"&"+"per_page="+str(per_page)
    return url
```

Índice de códigos 12.3: Código función para crear url

`pyCurl` es una librería de Python que permite reallizar una funcionalidad similar a `cURL` de la consola para transferir datos de un servidor mediante un url. A continuación lo usamos sobre el url creado de forma que se recibe un objeto en json como respuesta de vLex, para este servicio se ha de tener una cuenta creada (usrPwd). Posteriormente el objeto json se convierte en un diccionario que para cada sentencia contiene informaciones básicas como el id y resumen. Mediante este diccionario, se itera y se crea una cola (*queue*) de ids de las sentencias aprovechando posteriormente su propiedad FIFO[1].

```
def curl(url):
    data = BytesIO()
    c = pycurl.Curl()
    c.setopt(pycurl.URL, url)
    c.setopt(pycurl.USERPWD, usrPwd)
    c.setopt(pycurl.WRITEFUNCTION, data.write)
    c.perform()
    return json.loads(data.getvalue())

def getJudgementIds(page):
    url = buildUrl(page)
    dict = curl(url)
    idQueue = queue.Queue(maxsize=per_page)
    resultList = dict["results"]
    for result in resultList:
        try:
            idQueue.put(result["id"])
        except KeyError:
            print("no id")
    return idQueue
```

Índice de códigos 12.4: Código curl y creación de cola de id

Descargar sentencia

Una vez que tengamos el id de la sentencia, podemos accederla mediante el url:

$$http://supremo.vlex.es/vid/ + id\_sentencia$$

[1]First In First Out

En combinación con el método `HTTP Get` permite obtener todo el contenido de la sentencia de formato html. Opcionalmente si el usuario desea, esta sentencia se puede guardar en un *path* indicado.

```
1 def downloadDocument(id):
2     url = downloadUrl+str(id)
3     r = requests.get(url, allow_redirects=True)
4     name = str(id)+'.html'
5     path = pathToFiles + "\\" + name
6     open(path, 'wb').write(r.content)
```

Índice de códigos 12.5: Código descarga de sentencia mediante id

12.4.3 HtmlParser.py

Tener la sentencia en html tiene la ventaja de poder visualizar un formato más ordenado con un navegador, pero como inconveniente dificulta los procesos de extracción de información, ya que `Textserver` requiere el texto de entrada en texto plano (string). Para este propósito, este módulo se encarga de extraer las diferentes secciones de la sentencia (detallada en capítulos anteriores) de un fichero html.
Para cada sentencia de entrada en html, se creará una instancia de la clase `Judgement` para almacenar las diferentes secciones de la sentencia extraídas del html. Para empezar, se carga el fichero en `BeautifulSoup` y se extrae todo el contenido que hay bajo el tag `<div class='content'>`. Adicionalmente algunas sentencias también dispone de una sección Resumen"que se puede identificar con el tag `<p class='vid_abstract'>`. Sin embargo, las secciones (antecedente, fallo...) no se almacena directamente como string en `Judgement`, sino que en una estructura de:

$$parrafo \in subseccion \in seccion$$

Que en cuanto a implementación cada sección es una lista de lista. Y en dentro de cada párrafo tenemos:

$$palabra \in frase \in parrafo$$

El algoritmo para la extracción de texto se basa en iterar sobre los descendientes del tag anterior, y comprobar si se trata de un `<span>` que puede derivar en dos casos. El primero, si el texto es Antecedente, Fundamento o Fallo, entonces estamos entrando en una sección nueva por lo que habrá que almacenar la sección anterior y vaciar el contenedor de la sección actual. El otro caso es que el tag sea de tipo Primero, Segundo ... que marca las subsecciones, de este modo guardamos la subsección actual a la sección (la lista) y inicializamos a vacío la lista que representa la subsección.

En caso de que no se trate de un , el contenido de este tag será un párrafo, por lo que simplemente se añadirá en la lista de subsección actual.

```
def extractTextFromHtml(soup, judgement):
    resumTag = soup.find(lambda tag: tag.name == 'p' and tag.get('class') == ['vid_abstract'])
    if (resumTag is not None):
        judgement.resum = resumTag.text

    content = soup.find(lambda tag: tag.name == 'div' and tag.get('class') == ['content'])

    partsText = ["ANTECEDENTES DE HECHO", "FUNDAMENTOS DE DERECHO", "FALLO" , "FALLAMOS", "F A L L O", "F A L L A M O S"]
    # extract datos part
    datos = ""
    for tag in content.findAll(lambda tag: tag.name in ['p', 'span']):
        if (tag.name == 'span' and tag.text in partsText):
            break
        else:
            datos += tag.text + "\n"
    judgement.datos = datos

    # text has 3 other parts: antecedente, fundamento , fallo. Each part in subparts: primero, segundo .... And then these subparts in paragraphs
    part = []
    subpart = []
    actualPart = -1
    for tag in content.findAll(lambda tag: tag.name in ['p', 'span']):
        if tag.name == 'span' and tag['class'][0] == "l1":
            if tag.text in partsText:
                part.append(subpart)
                # save the part in judgement instance, in case actualPart is different than -1
                if (actualPart == 0):
                    judgement.antecedente = part
                elif (actualPart == 1):
                    judgement.fundamento = part
                elif (actualPart != -1):
                    judgement.fallo = part
                actualPart = partsText.index(tag.text)
                part = []
                subpart = []
            else:   # PRIMERO, SEGUNDO
                part.append(subpart)
                subpart = []
        else:
            subpart.append(tag.text)

    part.append(subpart)
    if (actualPart == 0):
        judgement.antecedente = part
    elif (actualPart == 1):
        judgement.fundamento = part
    elif (actualPart != -1):
        judgement.fallo = part
```

Índice de códigos 12.6: Código extraer partes de la sentencia desde el html

La extracción de información (los campos que pide en la memoria), por lo general no se realiza en este módulo. Sin embargo, las sentencias en html siempre aparecen la fecha de resolución y el tipo de recurso en una posición muy fija. Por lo que el trabajo de extracción de estos se llevará a cabo con las siguientes funciones.

```
def getResourceType(soup):
    main = soup.find("div", id = "main-content")
    table = main.find("table")
    resList = table.find_all("td")
    return resList[5].text

def getResolutionDate(soup):
    main = soup.find("div", id = "main-content")
    table = main.find("table")
    resList = table.find_all("td")
    if (len(resList) == 10):
        return resList[7].text
    elif (len(resList) == 12):
        return resList[9].text
    return " "
```

Índice de códigos 12.7: Código identificación de fecha de resolución y tipo de recurso

12.4.4 Judgement.py

Es la clase que representa una sentencia. Se almacena las secciones mencionada anteriormente y los campos de información extraídos en diccionario.

```
def __init__(self, id):
        # instance variables
        # parts of a judgment
        self.resumen = ""
        self.datos = ""
        self.antecedente = []
        self.fundamento = []
        self.fallo = []
        # save extracted fields in a dict()
        self.info = {
            "Numero de Identificacion" : "",
            "Tipo de Recurso" : "",
            "Fecha de Inicio" : "",
            "Fecha de Resolucion" : "",
            "Tribunal Origen" : "",
            "Ubicacion Tribunal Origen" : "",
            # ---------------- Law -------------
            "Nomativa Litigacion": "",
            # ---------------- Actors -----------
            "Demandante": "",
            "Tipo de Demandante": "",
            # ---------------- Fallo ---------------
            "Fallo": "",
            "Costas": ""
            }
```

```
26          self.info["Numero de Identificacion"] = id
```

Índice de códigos 12.8: Atributos de la clase Judgement

Todo el proceso de extracción de los campos de interés se implementa en este módulo, así como también la interacción con `Textserver` para el servicio de `Named Entity Recognition`, que como respuesta se recibe un xml que aparte de NER, también se incluye resultados de lematización, PoS tagging, análisis morfológica y desambiguación de sentido.

```
def postToTextserver(text, lang='es', out='xml', service='entities'):
    # Create request
    request_data = {'username':user,
                    'password':pwd,
                    'text_input':text,
                    'language':lang,
                    'output':out,
                    'interactive':'1' }

    url = "http://frodo.lsi.upc.edu:8080/TextWS/textservlet/ws/processQuery/" +
    service
    # Send request and get response
    resp = requests.post(url, files=request_data)
    # HTTP error, raise exception
    if resp.status_code != requests.codes.ok :
        resp.raise_for_status()
        return "error from Textserver, code"+ str(resp.status_code)

    return resp.text
```

Índice de códigos 12.9: Código de función para procesar un texto por Textserver

A continuación se explicará como se extrae los campos mencionados en capítulos anteriores. Por lo general la técnica usada es NEC en combinación con *matching* de expresión regular sobre el texto lematizado, obtenida con la función siguiente.

```
def lemmatize(root):
    text = ""
    firstIndex = 2 # [wordCount, cpuTime, paragraphs ....]
    n = len(root)
    for i in range(firstIndex, n):
        paragraph = root[i]
        for sentence in paragraph:
            for token in sentence:
                if (token.get("form") != ","): # eliminate all commas
                    # only to lemma the entities without classification
                    if (token.get("nec") == None):
                        lemma = token.get("lemma")
                    else:
                        lemma = token.get("form")
                    lemma = lemma.replace("_", " ")
                    text += lemma + " "
        text += "\n"
    return text
```

Índice de códigos 12.10: Código de función para obtener texto en su lema

Número de identificación

El id de la sentencia es su `primary-key`, por lo que es una variable esencial, pues toda instancia de esta clase ha de tener un id, recibiéndolo como un parámetro en la constructora.

Tipo de Recurso y Fecha de Resolución

Como se ha explicado en el `htmlParser`, estos campos se identifica en el módulo anterior dado su facilidad de identificación en el fichero html.
Cabe decir que todo campo de tipo fecha han de ser strings en el diccionario, por lo que se convierte a un formato estándar `date` con el objetivo de insertarlo posteriormente a la base de datos.

```
1
2  def toDateFormat(dateString):
3      s = dateString.lower()
4      monthDict = {"enero": "1", "febrero": "2", "marzo": "3", "abril": "4", "mayo": "
       5", "junio": "6",
5                   "julio": "7", "agosto": "8", "septiembre": "9", "octubre": "10", "
       noviembre": "11", "diciembre": "12"}
6      p = re.compile("(dia )?(?P<day>\d+) de (?P<month>\w+) de (?P<year>\d+)")
7      match = p.search(s)
8      day = match.group('day')
9      month = monthDict[match.group('month')]
10     year = match.group('year')
11     return year + "-" + month + "-" + day
```

Índice de códigos 12.11: Código función para estandarizar una fecha

En la sentencia, todas las fechas extraídas sigue el fomato de `[día dd de month de yyyy]`, donde `month` es el mes en español. Pues la tarea es tan simple como mediante expresión regular, identificar estas tres variables (día, mes y año) y convertir mes a formato numérico. La función retorna un string `yyyy-mm-dd`, preparado para ser visualizado e insertado a la base de datos.

Tribunal de Origen

Existe dos alternativas para este campo. Si la sentencia incluye una sección de información extra (datos como tribunal de procedencia, fecha...), se puede identificar tan simple como mediante la siguiente expresión regular:

$$Procedencia : (?P < tribunal > .*)$$

En caso contrario, se construye una lista de todas las entidades reconocidas como 'ORG' eliminando repeticiones y filtrando algunos falsos positivos. Después se itera

sobre esta lista y para cada organización se busca si esta se encuentra en la misma frase con expresiones como `resolución de | dictada por | contra la sentencia de`, ya que es el patrón habitual de las sentencias.

```
    for tribunal in possibleTribunals:
                regexp = f'([^.,]*?)(resolucion|Resolucion|contra la sentencia de)
    ([^.,]*?){tribunal}([^.,]*?)'  # regular expression to find two words in same
    sentence (, ... ,)
                match = re.search(regexp, text)
                if (match):
                    # if found, then we have the tribunal where the original
    judgement comes from
                    self.info["Tribunal Origen"] = tribunal
                    break
            # another patter
            if (self.info["Tribunal Origen"] == ""):
                for tribunal in possibleTribunals:
                    regexp = f'(dictada por)[^0-9a-zA-Z]*{tribunal}'  # regular
    expression to find two words in same sentence (, ... ,)
                    match = re.search(regexp, text)
                    if (match):
                        # if found, then we have the tribunal where the original
    judgement comes from
                        self.info["Tribunal Origen"] = tribunal
                        break
```

Índice de códigos 12.12: Código extracción de Tribunal Origen

Localidad de Origen

Igual que el caso anterior, tenemos dos alternativas. Si la sentencia incluye la sección de campos extra, entonces simplemente se identifica con la expresión regular:

$$T.S.J.(?P < loc > \backslash w+)$$

Que a continuación con `spaCy` se comprueba que 'loc' es efectivamente una localidad. En el otro caso, se ha de iterar sobre la lista de entidades identificadas como 'LOC' y se crear una expresión de búsqueda que compruebe si la localidad aparece en la misma frase que el tribunal origen extraído anteriormente.

Fecha de inicio

Se itera sobre la lista de palabras identificadas como `pos='date'`, y se busca si aparece en la misma frase con la palabra `casasión`. El *matching* solo se hace sobre la sección de Contexto dado que sino nos encontramos con muchos falsos positivos.

Demandante y tipo de demandante

El demandante puede ser de tipo persona, grupo de personas u organización. Para esta distinción se crea una lista de pares `(nombre, tipo)` donde `tipo` $\in$ `['PER', 'ORG']` a partir de NEC. Dicho esto, seguimos el algoritmo anterior, para cada entidad de esta lista, buscar *match* en el texto combinándolo con palabras claves como `representación|instancia|petición`.
Una vez encontrado un *matching*, si se trata de una persona, entonces buscamos si en el texto alrededor del nombre de demandante aparece nombres de otras personas, si se da el caso, podemos decir que se trata de un grupo de personas.

```
    for name, type in candidates:
            nameInRegex = name.replace("(", "\(").replace(")", "\)")
            regexp = f"(representacion|instancia|peticion) de (el |la )?[^0-9a-zA-Z
]*{nameInRegex}"
            match = re.search(regexp, firstParagraph)
            if (match):
                self.info["Demandante"] = name
                self.info["Tipo de Demandante"] = type
                break

        if (self.info["Demandante"] == ""):
            for name, type in candidates:
                nameInRegex = name.replace("(", "\(").replace(")", "\)")
                regexp = f"{nameInRegex}[^0-9a-zA-Z]*representar"
                match = re.search(regexp, firstParagraph)
                if (match):
                    self.info["Demandante"] = name
                    self.info["Tipo de Demandante"] = type
                    break

        # check if actor is a group when its classified as person
        if (self.info["Tipo de Demandante"] == "Individuo"):
            found = True
            while (found):
                found = False
                for name, type in candidates:
                    if (type == "Individuo"):
                        demandanteInRegex = self.info["Demandante"].replace("(", "\(
").replace(")", "\)")
                        nameInRegex = name.replace("(", "\(").replace(")", "\)")
                        regexp = f"{nameInRegex}(?P<separator>([^0-9a-zA-Z]|y)*){
demandanteInRegex}"
                        match = re.search(regexp, firstParagraph)
                        if (match):
                            self.info["Demandante"] = name + match.group("separator"
) + self.info["Demandante"]
                            self.info["Tipo de Demandante"] = "Grupo de individuos"
                            found = True
                            break
```

Índice de códigos 12.13: Código extracción de Demandante

Legislación

La ley sobre la que se hace la litigación aparece en la sentencia en formato `Ley nn/yyyy`, por lo que es fácil encontrarlo mediante expresión regular:

$$Ley(?P < num > \backslash d+)(?P < year > \backslash d+)$$

Entonces una vez encontrado un *match* se actualiza los otros campos relacionados como tipo de ley, fecha de aprobación, ámbito geográfico, código y subcódigo. Toda esta información está almacenada en una tabla de leyes, creada a partir del excel de Comparative Policy Agenda, permitiendo recuperarla mediante la ley en dicho formato sirviendo de `primary key`.

Fallo y costas procesales

Como resultado de la sentencia, se identificará dos tipos de fallos: favorable y no favorable. A la práctica, las sentencias no favorables aparecen palabras como `desestimar|no haber lugar|no ha lugar`, por lo que la forma de identificarlo es muy simple, lo tenemos suficiente con hacer *matching* con expresión regular sobre la sección de Fallo buscando esas palabras claves.
Este método es útil y eficaz, pero no muy generalista dado que depende mucho de como se ha redactado la sentencia. Una alternativa es escoger unas sentencias manualmente clasificadas como favorable o no, y usarlas como modelo de comparación. Entonces para decidir el fallo de una nueva sentencia simplemente hacemos comparación con los modelos mediante métodos de comparación de documentos como `Similaridad de Jaccard` o `Similaridad de coseno` explicado en el capítulo de bases teóricas.
Si se dispone de suficientes muestras (sentencias), incluso podemos entrenar con *machine learning* un clasificador que reciba textos convertido en `bag o words` y decida el fallo. Sin embargo, no disponemos de tantas muestras ya clasificadas para el entrenamiento.

Por el otro lado para identificar si hay costas procesales, mediante expresión regular buscamos si aparece `no` y `costa` en una misma frase, con lo cual podemos decidir que no hay costas. Si no se ha encontrado ningún *match* entonces buscamos si aparece solamente `costa`, por lo que implica que hay costas. De este modo, distinguimos si la costa es para el recurrente o demandado mediante *matching* sobre la palabra `recurrente`.

Multithreading

La tarea de extracción de los campos anteriores se ha distribuido entre tres `threads`, que corresponde a la extracción sobre las tres partes principales de la sentencia: Contexto y Antecedente, Fundamentos de derecho y Fallo. La aceleración mediante paralelismo es bastante notable, dado que las tareas son costosas, y la independencia entre estas hace que no tengamos que preocuparnos por problemas como interferencia de memoria. La lentitud de estos procesos no se debe al algoritmo de extracción, sino a la conexión y transferencia de datos con los servidores (vLex y Textserver), algo que difícilmente se puede mejorar. Entonces la única opción para aumentar la eficiencia es usar `threads`, de modo que teóricamente se puede llegar a acelerar hasta un triple de velocidad, sin tener en cuenta los *overheads* que hay para la gestión de `threads`.

12.4.5 DatabaseManager.py

Este módulo implementa la interacción con la base de datos. Cada vez que se inicia el programa se llama a la función `inicialize()` que establece la conexión a la BD local de `MySQL` y comprueba que las dos tablas que se usa en el proyecto exista, y sino simplemente se crea.

La primera se encarga de almacenar los campos extraídos de la sentencia mencionados en el módulo anterior:

```
    id INT PRIMARY KEY,
    type VARCHAR(255),
    iniDate DATE,
    judgementDate DATE,
    oriTribunal VARCHAR(255),
    location VARCHAR(255),
    law VARCHAR(255),
    actor VARCHAR(255),
    typeOfActor VARCHAR(255),
    fallo VARCHAR(255),
    costs VARCHAR(255)
```

Índice de códigos 12.14: Tabla Judgements

La segunda, creada a partir del excel con todas las leyes españolas hasta el momento.

```
    id INT PRIMARY KEY,
    legislature INT,
    year INT,
    month INT,
    description LONGTEXT,
    type_of_law VARCHAR(255),
    majortopic INT,
```

```
8     subtopic INT
```

Índice de códigos 12.15: Tabla Laws

Entonces, para operaciones sobre las tablas se usa *queries* de SQL tipo `Select * from judgements where id = ...` o `Insert into judgements values(...)`. Durante la ejecución del programa no se hace modificaciones sobre la tabla de leyes, los datos se cargan una vez a la hora de su creación cuando la función `initialize()` detecta la ausencia de dicha tabla.
Las dos tablas trabajan conjuntamente en el sentido de que en la primera tabla solamente se almacena la ley, que servirá como `primary key` para identificarlo en la segunda tabla y recuperar otras informaciones sobre dicha ley. De modo que se consigue un ahorro de espacio importante cuando se escala el número de sentencias extraídas.

12.4.6 Settings.py

Este módulo tiene la tarea de modificar y leer el contenido del fichero de configuración `settings.cfg`. La modificación se debe al módulo `main` cuando el usuario cambia una opción de configuración. Y la lectura lo realiza `Crawler.py` y `HtmlParser.py` dado que la path para almacenar la sentencia es personalizable por el usuario.
Las configuraciones se representan como un diccionario donde la *key* es la configuración y el valor es la opción. Por lo que se ha implementado la función de modificación de tal manera que usa otro diccionario para el mapeo entre keys:

```
1     def save_settings(values):
2     global settings
3     if values:      # if there are stuff specified by another window, fill in those
      values
4         for key in SETTINGS_KEYS_TO_ELEMENT_KEYS:
5             try:
6                 settings[key] = values[SETTINGS_KEYS_TO_ELEMENT_KEYS[key]]
7             except Exception as e:
8                 print(f'Problem updating settings from window values. Key = {key}')
9
10    with open(SETTINGS_FILE, 'w') as f:
11        json.dump(settings, f)
12
13    sg.popup('Configuracion guardada correctamente')
```

Índice de códigos 12.16: Código función save_settings

12.4.7 Complejidad

El análisis de la complejidad del proceso de extracción de información se puede aproximar por las dos subtareas principales de esta, NEC y expresión regular:

1. El trabajo de NLP se lo encargamos a `Textserver`, que internamente integra `Freeling`. Por defecto se piensa que se usa el módulo `Basic NER` dado que es más universal que `BIO NER` al ser un sistema basado en reglas. De este modo, podemos estimar su complejidad dependiendo de la representación que se usa para las reglas. Habitualmente se usa un grafo para la representación de reglas donde cada nodo es una regla, entonces se consigue una propagación en tiempo $O(\log n)$ mediante técnicas de `divede & conquer`, siendo n el número de reglas. Otras técnicas de representación como árbol de decisión o *linked list* llegan a conseguir una complejidad lineal $O(n)$. Sin embargo, a la práctica, este coste temporal es notablemente menor que el tiempo requerido para la transferencia de datos con el servidor.

2. El coste de la expresión regular está asociado de como se implementa. Si se representa mediante un NFA, entonces tiene un coste de construcción lineal de $O(m)$, siendo m el tamaño de la expresión regular, y un coste de evaluación (*matching*) de $O(m*n)$, donde n es la longitud del texto, dado que se ha iniciar un recorrido para cada uno de los m estados, que en casos peores puede llegar a $O(m^2 * n)$ por el `backtraking` debido a la indeterminación de los NFAs.
   Por otra parte, la representación mediante DFA, tiene un coste de construcción de $O(2^m)$ (si el NFA tiene m estados, el DFA puede llegar a tener 2^m) y un coste de *matching* lineal $O(n)$ dado que solo existe una posible ruta para los DFAs.
   Finalmente, en Python las expresiones regulares se evalúan mediante NFAs.

Dicho esto, para cada proceso de extracción, enviamos una vez a procesar a `Textserver`, mientras que las comprobaciones mediante expresión regular se ejecuta una vez para cada entidad reconocida. Sea l la longitud de la lista de entidades, tenemos que el coste asintótico del proceso es $O(l * n * m)$.

Capítulo 13

Rendimiento del sistema

A continuación se comprueba el rendimiento del sistema desde dos aspecto, la eficiencia y la precisión de los campos extraídos. El experimento consiste en la prueba de un subconjunto de 20 sentencias escogidas aleatoriamente de vLex, dado que lo comprobación de correcteza se realiza manualmente sobre cada sentencia, no es posible hacer un *test* con una colección gigante. Por lo que puede ser un tanto impreciso para estimar el rendimiento real del sistema, sin embargo es útil para tener una idea general del funcionamiento.

13.1 Eficiencia

El proceso de extracción se puede descomponer en las siguientes subtareas que analizaremos en orden.

Descargar Sentencia de vLex

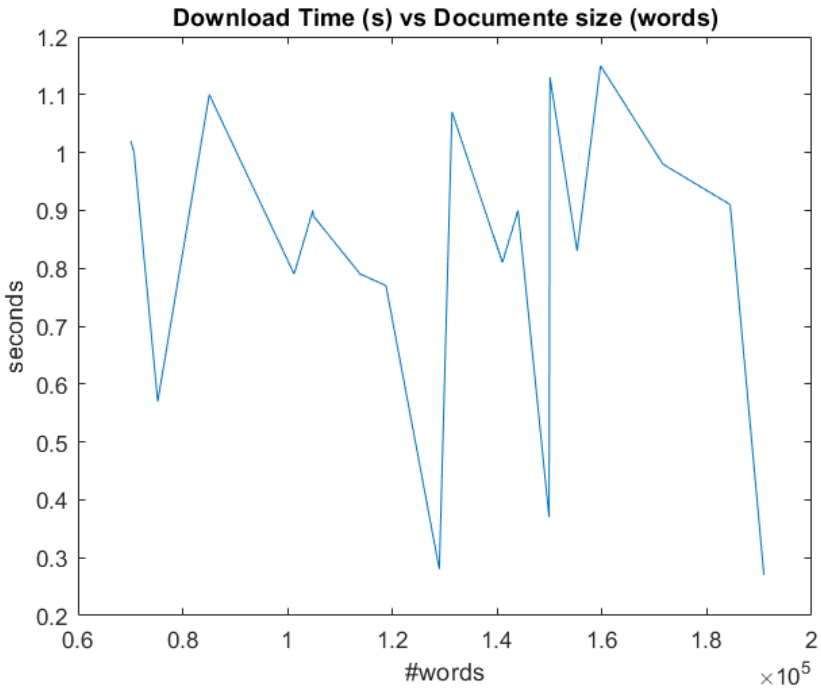


Figura 13.1: *Gráfica tiempo de descarga. Elaboración propia mediante* `Matlab`

Como podemos observar, el tiempo para descargar una sentencia no es proporcional al tamaño de esta. Dado que la velocidad de transferencia con un servidor está influenciado de muchos factores, como la carga del servidor en aquel momento, su estado funcionamiento, si se está realizando actualizaciones o reparaciones...

Textserver

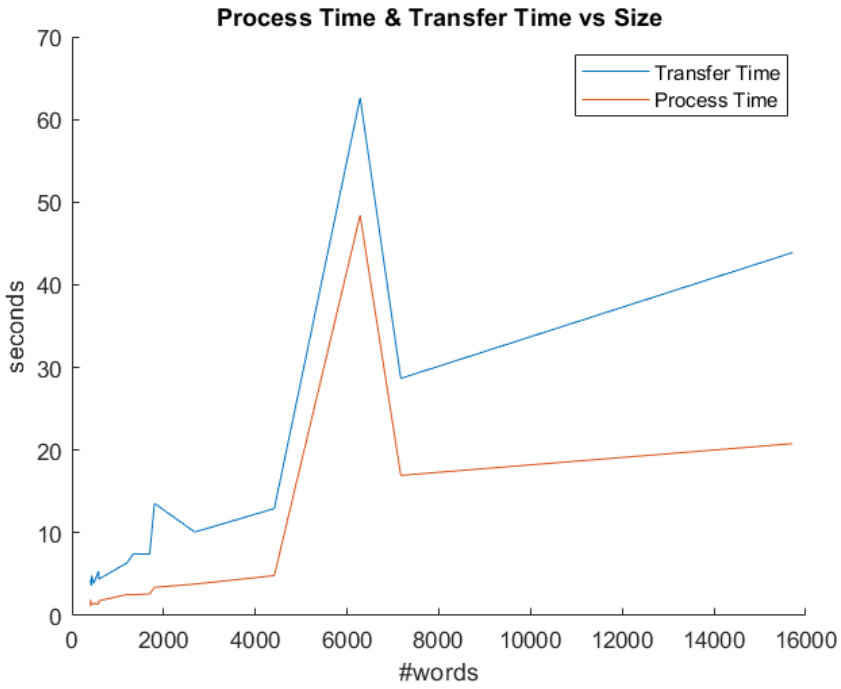


Figura 13.2: *Gráfica tiempo de procesado con Textserver. Elaboración propia mediante* `Matlab`

En este caso, tanto el tiempo de procesado como el tiempo de transferencia del resultado de Textserver es proporcional (casi lineal) al tamaño de entrada. En la gráfica se observa un pico irregular, pero lo más probable es que se debe a la inestabilidad de la conexión al servidor.
El tiempo de procesado está incluido en el tiempo para recibir la salida de Textserver (tiempo de transferencia), por lo que en la gráfica vemos que la línea azul siempre está encima que la naranja.

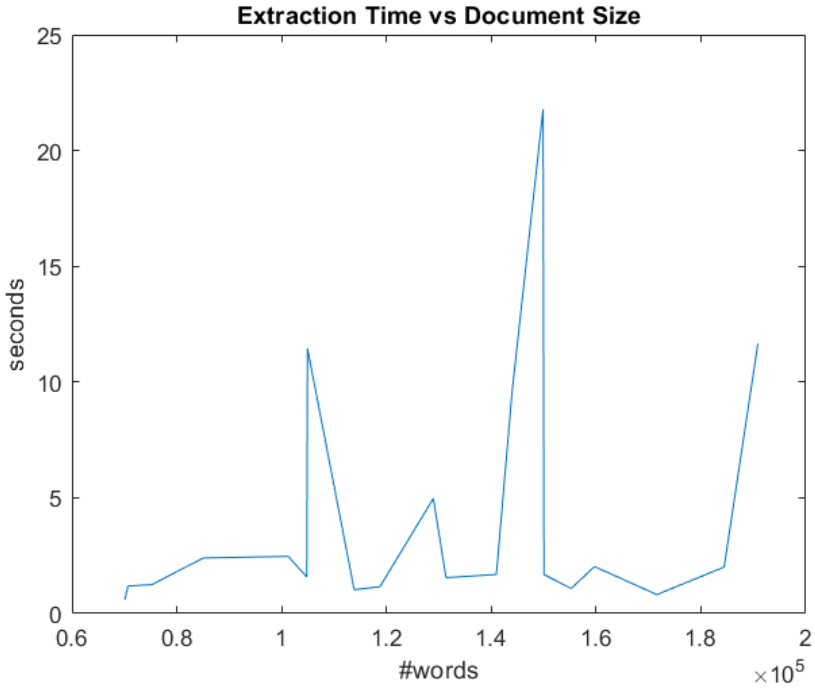


Figura 13.3: *Gráfica tiempo de extracción de información. Elaboración propia mediante* `Matlab`

En general el tiempo de extracción no se ve muy afectado por el tamaño de entrada, que por lo general en la práctica son aproximadamente 2 segundos. El uso de expresión regular hace que el tiempo de cómputo sea bastante inestable, dado que se implementa con un NFA en Python, en caso de que no encuentre un `match` se hará `backtracking` sobre todas las posibilidades. Llevando así el tiempo de cómputo a otro nivel, en la gráfica vemos varios picos que probablemente se deba a esta razón.

13.1.1 Precisión de los campos extraídos

De las 20 sentencias a analizar, se clasificará el resultado en 4 categorías. TP (true positive) si la información extraída por el programa coincide con la real. TN (true negative) si no se ha podido extraer un campo y realmente tampoco aparece en la sentencia. FN (false negative) si no se ha podido encontrar ese campo mientras que si

aparece en la sentencia. FP (false positive) si se ha extraído una información erróneo.

Los campos de número de identificación, tipo de procedimiento y fecha de resolución, no se tendrá en cuenta es este *test* ya que se extraen directamente del html, por lo que no debería haber errores. Incluirlos en el calculo puede perjudicar al resultado del *test*.

Campo de información	**TP**	**TN**	**FP**	**FN**
Fecha de inicio	7	7	0	6
Tribunal origen	16	0	1	3
Localidad	2	13	0	5
Ley aplicada	18	0	0	2
Demandante	17	1	1	1
Tipo de demandante	14	2	3	1
Fallo	20	0	0	0
Costas procesales	16	0	2	2
Total	110	23	7	20

Cuadro 13.1: *Resultado test de rendimiento sobre 20 sentencias. Elaboración propia*

Entonces mediante esta tabla, podemos estimar el rendimiento como informaciones correctas sobre el total:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = 0{,}83$$

Sin embargo, esta métrica es adecuada para tipo de datos balanceados. Que no es caso para las sentencias judiciales que analizamos, por ejemplo todas las sentencias usadas en este *test* son de tipo `No favorable`, por lo que no es nada fiable decir que el sistema es 100 % correcto identificando el campo Fallo, dado que no se tiene muestras suficientes para Fallo `Favorable`.

Dicho esto, podemos usar *f-score* para estimar el rendimiento de sistema. Siendo la media armónica sobre *Precision* y *Recall*, penalizando valores extremos, cosa que lo hace útil para *datasets* no balanceados.

$$Precision = \frac{TP}{TP + FP} = 0{,}94 \quad y \quad Recall = \frac{TP}{TP + TN} = 0{,}85$$

$$F_1\_score = (\frac{Recall^{-1} + Precision^{-1}}{2})^{-1} = 2 * \frac{Precision * Recall}{Precision + Recall} = 0{,}89$$

Capítulo 14

Conclusiones

14.1 Conclusión del proyecto

En este proyecto hemos podido desarrollar un *software* que cumpla con los objetivos descritos en el capítulo de Alcance, así como también finalmente realizar un análisis del rendimiento del sistema.

Para empezar, para la obtención de las sentencia, se usa el API de vLex para obtener un listado de identificadores de las sentencias, mediante el cual permite el acceso y descarga de la sentencia en formato *html*.

En segundo lugar, existen campos de información que aparece en posiciones fijas de la sentencia. Entonces la extracción de estas es trivial usando *parsers* de html.

Terceramente, para la extracción del resto de información, se usa expresiones regulares para buscar patrones e identificar fragmentos del texto que potencialmente puede ser el campo que queremos extraer. Para eliminar posibles falsos positivos, se combina expresión regular con entidades clasificadas por analizadores lingüísticos (en este caso en *cloud*), de tal modo asegurar que el fragmento de texto es efectivamente una entidad como persona o organización.

Finalmente, se ha realizado un estudio sobre la eficiencia y correcteza del sistema, un *f-score* de 0.89 indica un resultado decente. Dado que los problemas de extracción de información es sensible al contexto de los documentos, y no estamos basando en un analizador lingüístico generalista.

14.2 Trabajo futuro

Del proyecto existen varios aspectos que podemos mejorar y remarcar como línea de trabajo futuro que debido a la limitación del tiempo del TFG no se pudo desarrollar.

A pesar de que se ha conseguido un resultado decente en los experimentos, como se había comentado, la fase de *testing* se realiza sobre un subconjunto reducido de 20 sentencias, por lo que puede haber bastante imprecisión a la hora de estimar el rendimiento. Lo ideal sería probar con un subconjunto mayor de sentencias para aumentar la fiabilidad del resultado de *testing*.
Dicho esto, la precisión del sistema se basa principalmente en dos componentes: la clasificación de entidades y la expresión regular. El primero se puede mejorar usando modelos de *machine learning* entrenados sobre un corpus de contexto jurídico en lugar de un clasificador basado en reglas. Las expresiones regulares se basa en observar empíricamente patrones de las sentencias, entonces para obtener más cobertura sobre las expresiones del lenguaje, se ha de analizar un conjunto mayor de sentencias.

En el sistema usamos para cada proceso de extracción tres threads que corresponde a las tres secciones de la sentencia. Sin embargo, los computadores de hoy en día disponen de un número de threads mucho mayor, esto lo podemos aprovechar para analizar varias sentencias a la vez y de allí conseguir un aumento de velocidad.

Para este proyecto se ha seguido lo que se pedía en la memoria *Polisjuris*. Sin embargo, podemos extender los campos de información a extraer. Como por ejemplo también sería interesante tener información sobre los jueces y magistrados que interponen en el fallo de la sentencia, de forma que posteriormente se puede hacer un análisis sobre la probabilidad de fallo favorable de un juez.

14.3 Integración de conocimiento

La realización del presente proyecto ha sido útil para reflejar conocimientos adquirido en varias disciplinas desarrolladas durante la carrera:

1. PRO1,PRO2[1]: ya que tener una buena base en programación es esencial en el desarrollo de cualquier tipo de software.

[1]Programación 1, 2

2. SO[2], PAR[3]: en el uso y gestión de sistemas multiproceso.

3. IDI[4]: se ha podido aprovechar toda la teoría sobre el diseño de interfaz que se enseñó en esta asignatura a pesar que este proyecto usa un *framework* diferente (`pySimpleGUI` frente a `Qt`).

4. BD[5]: ha sido de gran ayuda, sin haber cursado esta asignatura habría dedicado un tiempo extra para aprender a como manejar bases de datos.

5. LP[6]: el proyecto está totalmente programado en Python, y es en esta asignatura dónde tube un primer contacto con este lenguaje.

6. TC[7]: a la teoria sobre los lenguajes y el uso de expresión regular. Una técnica potente para buscar patrones en textos, muy usada en este proyecto para la extracción de información.

7. MD[8], IA[9]: aunque en el proyecto no se ha podido aplicar métodos de aprendizaje automático, el conocimiento aprendido en estas asignaturas ha sido esencial para poder reflexionar sobre la alternativa de construir un clasificador de entidades mediante modelos de machine learning.

8. CAIM[10]: en las técnicas de procesamiento de textos, así como tokenizar, lematizar o métodos de calcular la similitud entre textos.

14.4 Competencias técnicas

A continuación se explicará como se ha alcanzado las competencias técnicas definidas al comienzo del TFG.

1. **CCO1.1: Evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan llevar a su resolución, y**

[2]Sistemes Operatius
[3]Paralelisme
[4]Interacció i Disseny d'Interfícies
[5]Bases de Dades
[6]Llenguatges de Programació
[7]Teoría de Computació
[8]Mineria de Dades
[9]Inteligencia Artificial
[10]Cerca i Anàlisi d'Informació Massiva

recomendar, desarrollar e implementar la que garantice el mejor rendimiento de acuerdo con los requisitos establecidos. [En profundidad].

En la sección [12.4.7] se ha hecho un análisis sobre la complejidad del proceso de extracción de información, así como una estimación del coste temporal del proceso de análisis lingüístico y expresión regular. Sin embargo las tareas de NLP se llevan a cabo en servidores remotas, por lo que no ha sido posible hacer una mejora sobre su rendimiento.

2. **CCO1.3: Definir, evaluar y seleccionar plataformas de desarrollo y producción hardware y software para el desarrollo de aplicaciones y servicios informáticos de diversa complejidad. [Bastante].**

   En el proyecto se han integrado diferentes frameworks y librerías, pues la selección de estas no ha sido trivial, en el capítulo 11 (Tecnologías aplicadas) se ha realizado comparaciones con alternativas y la explicación del porqué de su uso.

3. **CCO2.1: Demostrar conocimiento de los fundamentos, los paradigmas y de las técnicas propias de los sistemas inteligentes, y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen estas técnicas en cualquier ámbito de aplicación. [Bastante].**

   A pesar de que no se ha podido usar técnicas de *machine learning* en proyecto, se ha analizado como alternativa su integración para la implementación del clasificador de entidades, que sin una base teórica sobre este campo no sería posible.

4. **CCO2.2: Capacidad para adquirir, obtener, formalizar y representar el conocimiento humano de una forma computable para la resolución de problemas mediante un sistema informático en cualquier ámbito de aplicación, particularmente en los que están relacionados con aspectos de computación, percepción y actuación en ambientes o entornos inteligentes. [Bastante].**

   La tarea de extracción de información de un documento requiere al programa un comportamiento asimilable a la inteligencia humana. Pues, no sería posible

su implementación sin tener conocimientos sobre técnicas de *Natural Language Processing*.

5. **CCO3.1: Implementar código crítico siguiendo criterios de tiempo de ejecución, eficiencia y seguridad. [Bastante].**

   Dado de la imposibilidad de mejorar la eficiencia algorítmicamente, en el proyecto se ha integrado *threads* para paralelizar los procesos y conseguir un aumento en tiempo de ejecución.

14.5 Normativa y regulación

Los programas informáticos cada vez están ganando más protagonismo en la sociedad actual, sin embargo no gozan de la misma protección que otras figuras de propiedad industrial, como patentes y modelos de utilidad. De forma que La Ley 24/2015 de Patentes, de 25 de Julio, lo excluye expresamente en el ámbito de protección.

Por otra parte, existe la Ley de Propiedad Intelectual (Real Decreto Legislativo 1/1996, de 12 de abril) bajo la cual reconoce el software como bien inmaterial producto de creatividad humana, gozando el autor de derecho de explotación sobre su obra de creación (`copyright`). Dicho esto, el proyecto del actual TFG está protegido bajo este reglamento.

En cuanto a las librerías usadas en el proyecto están bajo licencias de software libre (`open-source`) como `MIT license` o `GNU Affero General Public License`. Donde el autor concede al usuario el derecho de ejecutar, modificar, estudiar o mejorar el producto.

La Ley Orgánica de Protección de Datos (Ley Orgánica 15/1999, de 13 de diciembre), una ley para proteger la privacidad de los usuarios, es muy frecuente en todo tipo de software. Sin embargo, en este proyecto no tiene aplicación, dado que para usar el servicio del producto no se le pide ningún tipo de dato personal al usuario.

Bibliografía

[1] "Aplicación de técnicas de inteligencia artificial al análisis de las decisiones judiciales: actores y temas en conflicto en los tribunales españoles (Polisjuris)". En: (2019).

[2] Michael Collins y Yoram Singer. *Unsupervised Models for Named Entity Classification*. URL: https://www.aclweb.org/anthology/W99-0613.pdf.

[3] *Conceptos Jurídicos: Jurisprudencia*. [Online; accessed 25-02-2020]. 2017. URL: https://www.conceptosjuridicos.com/jurisprudencia/.

[4] Robert Dale. *Law and Word Order: NLP in Legal Tech*. [Online; accessed 25-02-2020]. 2018. URL: https://web.stanford.edu/class/cs224n/reports/custom/15845830.pdf.

[5] Carla Hernandez Díaz. "Extracción de información de textos legales y notas de prensa". En: *Universidad La Laguna* (2017).

[6] *Energía en España*. [Online; accessed 09-03-2020]. URL: https://es.wikipedia.org/wiki/Energ%C3%ADa_en_Espa%C3%B1a#Energ%C3%ADas_renovables.

[7] *FreeLing User Manual*.

[8] Jeffrey E.F. Friedl. *Mastering Regular Expressions: Understand Your Data and Be More Productive*. O'Reilly Media, 2006.

[9] Profesores de GEP. *Módulo 2.4 Gestión económica*. 2019.

[10] *Glassdoor Job Search — Find the job that fits your life*. [Online; accessed 08-03-2020]. URL: https://www.glassdoor.es/index.htm?countryRedirect=true.

[11] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[12] Barbara Korousi Seljak Gorjan Popovski Stefan Kochev y Tome Eftimov. *A Rule-based Named-entity Recognition Method for Food Information Extraction*.

[13] Mohan Gupta. *A Review of Named Entity Recognition (NER) Using Automatic Summarization of Resumes*. [Online; accessed 20-05-2020]. 2018. URL: https://towardsdatascience.com/a-review-of-named-entity-recognition-ner-using-automatic-summarization-of-resumes-5248a75de175.

[14] Ricard Gavaldá José Luis Balcázar Ramon Ferrer-i-Cancho. "Transparencia de CAIM: Information Retrieval Models". En: 2018.

[15] Ricard Gavaldá José Luis Balcázar Ramon Ferrer-i-Cancho. "Transparencia de CAIM: Locality Sensitive Hashing". En: 2018.

[16] Ricard Gavaldá José Luis Balcázar Ramon Ferrer-i-Cancho. "Transparencia de CAIM: The Information Retrieval (IR) process". En: 2018.

[17] Mark S. Krass. *Learning the Rulebook: Challenges Facing NLP in Legal Contexts*. [Online; accessed 22-02-2020]. 2018. URL: https://towardsdatascience.com/law-and-word-order-nlp-in-legal-tech-bd14257ebd06.

[18] Susan Li. *Named Entity Recognition with NLTK and SpaCy*. [Online; accessed 25-05-2020]. 2018. URL: https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da.

[19] *PyCurl Quick Start*. [Online; accessed 20-03-2020]. 2020. URL: http://pycurl.io/docs/latest/quickstart.html.

[20] Lluís Marquez Rafel Cases. *Teoria de la computació: Llenguatges regulars e incontextuals*. UPC.

[21] sigmoider. *Get started with NLP (Part I)*. [Online; accessed 20-02-2020]. 2017. URL: https://medium.com/@gon.esbuyo/get-started-with-nlp-part-i-d67ca26cc828.

[22] Edward Loper Steven Bird Ewan Klein. *Natural Language Processing with Python*.

[23] *Sueldos para Project Manager*. [Online; accessed 25-02-2020]. URL: https://www.glassdoor.es/Sueldos/san-francisco-project-manager-sueldo-SRCH_IL.0,13_IM759_KO14,29.htm?countryRedirect=true.

[24] *Sueldos para Software Developer*. [Online; accessed 25-02-2020]. URL: https://www.glassdoor.es/Sueldos/software-developer-sueldo-SRCH_KO0,18.htm?countryRedirect=true.