UNIVERSIDAD DE LA LAGUNA

Análisis automático de textos en español utilizando NLTK

por

José Manuel Hernández Hernández

Un trabajo presentado para el
Grado en Ingeniería Informática

en la
Escuela Superior de Ingeniería y Tecnología

5 de septiembre de 2016

Dña. Coromoto León Hernández, con N.I.F 78.605.216-W, profesora Titular de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas, como tutora

C E R T I F I C A

Que la presente memoria titulada:

"*Análisis automático de textos en español utilizando NLTK*"

ha sido realizada bajo su dirección por D. **José Manuel Hernández Hernández**,

con N.I.F. 54.112.122-E.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firma la presente en La Laguna a 5 de septiembre de 2016.

"If you talk to a man in a language he understands, it will go to his head. If you talk to him in his own language, it will go to his heart."

Nelson Mandela

Abstract

The aim of this work is the development of a study case in Spanish in which Natural Language Processing techniques are applied, to observe the effectiveness of the NLP tools available for this language.

The study case chosen for this work is the classification of the Final Degree Projects made avaliable to the public in the Institutional Repository of the University of La Laguna.

Keywords: *NLP, Natural Language Processing, NLTK, Natural Language Toolkit, Python, CasperJS, JavaScript*

Resumen

La finalidad de este trabajo es el desarrollo de un caso de estudio en lenguaje español sobre el cual aplicar técnicas de Procesamiento del Lenguaje Natural, para constatar la eficacia de las herramientas PLN disponibles para este idioma.

El caso de estudio escogido para desarrollar en este trabajo fue la clasificación de los Trabajos de Fin de Grado puestos a disposición del público en el Repositorio Institucional de la Universidad de La Laguna.

Palabras clave: *PLN, Procesamiento del Lenguaje Natural, NLTK, Natural Language Toolkit, Python, CasperJS, JavaScript*

Agradecimientos

A mi tutora, Coromoto León Hernández, por su dirección, optimismo y apoyo incondicional a la hora de encaminar el proyecto.

Al personal docente de la Escuela Superior de Ingeniería y Tecnología, por su labor educativa.

A mis amigos y compañeros de la facultad, por el apoyo recibido.

A mi familia, y mis padres, por su paciencia, cariño y por hacer posible mi educación.

Índice

Índice de figuras

Lista de Listados

Índice de cuadros

Capítulo 1

Introducción

El Procesamiento del Lenguaje Natural (PLN) es una disciplina de la Computación que concierne al tratamiento de información desestructurada, presentada en texto o audio y expresada mediante Lenguaje Natural. Se entiende por lenguaje natural cualquier lenguaje utilizado para la comunicación humana en el día a día, en contraposición a un lenguaje artificial establecido, como por ejemplo los lenguajes de programación. Es un área de interés para el campo de la Interacción Persona-Computador (*Human-Computer Interaction*), así como los de Aprendizaje Automático (*Machine Learning*) e Inteligencia Artificial (*Artificial Intelligence*).

La necesidad del Procesamiento del Lenguaje Natural es aparente en los sistemas de Interacción Persona-Computador, dado que en este campo se busca el diseño de interfaces cómodas y funcionales para los usuarios humanos a la hora de utilizar ordenadores, ya sean tradicionales, portátiles, móviles, etc. El Procesamiento del Lenguaje Natural se posiciona como una capacidad deseable en los ordenadores, para permitir la comunicación natural entre el usuario y el ordenador, y aumentar tanto la comodidad y conveniencia como las posibilidades de una interacción exitosa.

En cuanto al campo del Aprendizaje Automático, no sólo nos referimos al hecho de que éste campo es necesario para el desarrollo de técnicas PLN, sino que el campo del Procesamiento del Lenguaje Natural a su vez puede ser útil para el Aprendizaje Automático en uno de sus mayores problemas. Uno de los principales cuellos de botella hoy en día a la hora de la construcción de sistemas de Aprendizaje Automático es la insuficiente cantidad de datos estructurados. El Procesamiento del Lenguaje Natural permite la conversión de los datos desestructurados en lenguaje natural en datos estructurados con los cuales un sistema informático pueda aprender.

Sólo por esto es por extensión de importancia para el campo de la Inteligencia Artificial, ya que los recientes avances en dicho campo se basan en el Aprendizaje Automático y por tanto se beneficia de las mejoras que el PLN pueda aportar al mismo. No es necesario concretar la importancia del PLN en el campo de la Inteligencia Artificial, ya que las primeras definiciones de una máquina inteligente (el Test de Turing) presuponen una capacidad de procesamiento e interpretación del lenguaje humano.

En otros campos de la ciencia se ha hablado extensamente de la importancia del lenguaje como factor decisivo para la producción de inteligencia tal y como la definimos, tanto así que existe una rama de la ciencia, la neurolinguística, centrada exclusivamente en el estudio de este fenómeno.

—

Para facilitar la fluidez en la lectura del documento, se hace uso de términos y abreviaturas que pueden ser propias del área de conocimientos a tratar o simplemente sean de nueva introducción, por lo que se proporciona un glosario como apéndice auxiliar con el fin de esclarecer su significado.

En el siguiente apartado, se hace una revisión de las áreas temáticas y herramientas en las que se subdivide el campo del Procesamiento del Lenguaje Natural. El resto del Capítulo 1 está dedicado a las herramientas consideradas para la realización del caso de estudio.

1.1. Estado del arte

El estado actual del campo tiene numerosas áreas temáticas que se pueden subclasificar en herramientas conceptuales y aplicaciones. Una lista extensa de éstas se define en el número 56 de la Revista de Procesamiento de Lenguaje Natural [5]:

Modelos lingüísticos, matemáticos y psicolingüísticos del lenguaje, refiriéndonos al PLN enfocado como un sistema de reglas.

Lingüística de corpus, metodología en la que se estudia el lenguaje a través de ejemplos de textos reales producidos en el mundo real.

Desarrollo de recursos y herramientas lingüísticas, en aquéllos trabajos que expanden las herramientas PLN disponibles para su futuro uso.

Gramáticas y formalismos para el análisis morfológico y sintáctico, refiriéndonos al PLN enfocado como un sistema de reglas lógicas aplicado al análisis morfosintáctico.

Semántica, pragmática y discurso. El estudio tradicional del lenguaje natural para cada idioma por parte de lingüistas nos permite abstraer las reglas para su uso y comprensión.

Lexicografía y terminología computacional, que busca una sistemática colección y explicación de todas las palabras (o más estrictamente, unidades léxicas) de un lenguaje.

Resolución de la ambigüedad léxica, siendo un ejemplo la determinación del sustantivo concreto dada una referencia indirecta.

Aprendizaje automático en PLN, es decir la aplicación de algoritmos de aprendizaje no supervisado para sistemas de esta índole.

Generación textual monolingüe y multilingüe. Esto es, la expresión en lenguaje natural de información estructurada, permitiendo a un ordenador generar a tiempo real respuestas más accesibles a un usuario.

Traducción automática, refiriéndonos a sistemas capaces de realizar la traducción de textos naturales a otro lenguaje natural con poca o nula supervisión humana.

Reconocimiento y síntesis del habla, atendiendo a problemática como la determinación de palabras homófonas a través del contexto.

Extracción y recuperación de información monolingüe, multilingüe y multimodal, en la que se aprovecha información multimedia recibida a tiempo real para analizar la intención del emisor.

Sistemas de búsqueda de respuestas, alternativas a los históricos sistemas expertos, en los que se pretende responder a preguntas del usuario sobre un ámbito reducido.

Resumen automático, por el cual obtenemos el resumen de un texto intentando conservar la información relevante.

PLN para lenguas con recursos limitados, como es el caso de lenguas indígenas o autóctonas.

Aplicaciones industriales del PLN, entre las más importantes la generación de información para la toma de decisiones en empresas, o Inteligencia Empresarial (*Business Intelligence*).

Sistemas de diálogo, sistemas que se comunican con el usuario utilizando lenguaje natural, de manera similar a mantener una conversación.

Análisis de sentimientos y opiniones, actualmente un área del PLN con gran popularidad y que puede considerarse como una de sus aplicaciones industriales de mayor interés.

Minería de texto, el procesamiento estadístico de grandes cantidades de texto con la intención de extraer información de alta calidad.

Implicación textual y paráfrasis, con la que podemos visualizar información textual de diversas maneras, así como conocer las enunciaciones que se derivan de ellas.

Se tuvieron en cuenta estas áreas temáticas a la hora de decidir el enfoque que tendría el trabajo, tomando en cuenta variables como factibilidad, tiempo disponible y utilidad, entre otras.

1.2. Ámbito del problema

Se decidió realizar un trabajo sobre Procesamiento de Lenguaje Natural, siendo este el primer punto acordado antes de proceder a la definición del problema a tratar para la realización del trabajo.

1.3. Definición del problema

El caso de estudio escogido para este trabajo puede ser subdividido en dos problemas: el primero es la obtención del texto a clasificar, ya sea desde un archivo de texto en local o a través de Internet.

El segundo problema es el más importante y el cual es objetivo principal de este trabajo: el procesamiento del texto una vez obtenido.

Se ha decidido presentar las secciones de manera cronológicamente equivalente a la que se produce en cualquier problema de Procesamiento de Lenguaje Natural.

1.3.1. Obtención del texto

Hoy en día existen muchas fuentes de las cuales obtener texto a la hora de realizar procesamiento de texto: se puede obtener de Internet, puede ser creado o incluso puede obtenerse de un medio físico como pueden ser los libros o la voz de un usuario.

La obtención del texto a procesar puede realizarse de muchas maneras, existiendo varias opciones para cada fuente de texto que se puede presentar.

En el caso de Internet, existen tanto librerías para lenguajes de programación como programas dedicados a la extracción de texto. Normalmente funcionan de manera similar a un navegador de Internet, utilizando el protocolo HTTP para enviar peticiones a servidores web y recibir contenidos de las páginas web, que luego son desglosados mediante *parsers* para obtener los datos deseados.

En nuestro caso, el objetivo es la clasificación de los Trabajos de Fin de Grado. El Repositorio Institucional de la Universidad de La Laguna [1] es un Servicio Web que contiene una gran cantidad de documentos relacionados con la actividad docente de la Universidad, entre ellos los Trabajos de Fin de Grado.

1.3.2. Procesamiento del texto

El procesamiento de texto es la parte del problema que toma una mayor importancia, ya que es la más complicada y objeto de este trabajo. Para nuestro caso de estudio, nos concentramos en un subproblema del campo; la clasificación de textos. La clasificación de textos es una de las de mayor relevancia actual.

El estudio previo de las opciones a considerar a la hora escoger las herramientas para realizar el proyecto ha ocupado una gran parte del tiempo dedicado. Esto es en parte debido a la gran cantidad de herramientas disponibles para la tarea a realizar: existen decenas de herramientas para el Procesamiento del Lenguaje Natural, y cientos para la extracción de texto a partir de fuentes en Internet. Para realizar el trabajo se han considerado varias herramientas, teniendo en cuenta puntos como la facilidad de uso, la adaptabilidad de la herramienta, su capacidad de procesamiento, las posibilidades de ser utilizada con entradas de texto en español, etc.

1.4. Herramientas de PLN

Se consideró un número de herramientas para el Procesamiento del Lenguaje Natural, las cuales fueron evaluadas según las ventajas y desventajas que proporcionarían al desarrollo del proyecto teniendo en cuenta los objetivos que se deseaban alcanzar.

La herramienta utilizada en este caso es la biblioteca NLTK (*Natural Language Toolkit*) desarrollada en el lenguaje de programación Python, pero es necesario conocer las decisiones que llevaron a esta elección.

A continuación se muestran las herramientas consideradas para el Procesamiento del Lenguaje Natural en la realización de este trabajo.

1.4.1. Stanford CoreNLP

Se trata de una herramienta de Procesamiento de Lenguaje Natural implementada en Java por la Universidad de Stanford [4]. Fue una de las herramientas de mayor importancia a considerar, debido a la familiaridad que se tenía con la metodología de trabajo a la hora de utilizarlo. Actualmente cuentan con modelos lingüísticos para el chino, inglés, francés, alemán, y español, el idioma objetivo de este trabajo. Una de las principales ventajas de esta herramienta es su implementación en Java, un lenguaje de más bajo nivel que Python o Ruby, lo que le proporciona una mayor capacidad de cómputo de datos.

1.4.2. Apache Lucene y Solr

Una de las grandes ventajas del sistema Solr es que posee una API (*Application Programming Interface*) flexible con la que se puede interactuar mediante los lenguajes de programación Ruby y Python, así como también es posible comunicarnos mediante el paso de mensajes JSON (JavaScript Object Notation), de manera similar a muchas arquitecturas servidor/cliente en la red [2].

1.4.3. Apache OpenNLP

Esta herramienta utiliza una aproximación diferente a la que usa el Stanford CoreNLP. Se trata de un sistema desarrollado en Java, que permite ser utilizado como una biblioteca Java. Además de esto, posee una interfaz de programación en línea (*scripting*) que puede también usarse mediante la línea de comandos [3]. A pesar de encontrarse algo desfasada en comparación con el resto de opciones, permanece como una opción robusta y rápida de implementar.

1.4.4. GATE

El GATE (*General Architecture for Text Engineering*) es un sistema de software libre con una gran capacidad de procesamiento de texto [8]. Fue desarrollado inicialmente por un equipo base de 16 programadores [9], empezando en 1995 como parte de un proyecto del EPSRC (Engineering and Physical Sciences Research Council), una organización basada en el Reino Unido con el objetivo de financiar la investigación científica en el país. Dado el largo tiempo que lleva esta herramienta en desarrollo, que abarca más de dos décadas, cuenta con un amplio abanico de funcionalidades de procesamiento de texto.

1.4.5. NLTK, Natural Language Toolkit

El NLTK (*Natural Language Toolkit*) es una biblioteca de Procesamiento de Lenguaje Natural que utiliza el lenguaje de programación Python [6]. NLTK es software libre, lo que permite a estudiantes y al personal académico realizar estudios con la herramienta sin necesidad de realizar una inversión económica. Esta herramienta es también de código abierto, lo que lo hace ideal para expandir sus funcionalidades en caso de necesitarlo. El hecho de estar implementada como una biblioteca Python reduce la curva de aprendizaje, y la acerca al mundo académico, cuya mayor parte de integrantes se encuentra familiarizado con este lenguaje de programación.

1.5. Herramienta de PLN elegida

Una razón para la elección de NLTK como herramienta es el gran soporte que tiene, debido a las dimensiones de su comunidad de usuarios. Es una de las herramientas de Procesamiento de Lenguaje Natural de mayor aceptación en el ámbito científico.

Otra razón importante es el apoyo que proporciona el libro *Natural Language Processing with Python*[6]. Este libro tiene por autores a los creadores del NLTK, haciéndolo idóneo para comprender y utilizar todas las funcionalidades que aporta esta biblioteca.

1.6. Lenguaje de programación escogido

El código de las herramientas de Procesamiento de Lenguaje Natural desarrolladas en Java y C++ es compilado a un código máquina de bajo nivel, lo que le confiere una mayor rapidez de cómputo.

Ésta amplia capacidad de procesamiento en comparación a herramientas para Python o Ruby es aprovechable ya sea en aumentar la cantidad de datos a procesar o reducir el tiempo que lleva el procesamiento de los mismos.

Sin embargo, el que una herramienta esté diseñada para un lenguaje de éstas características tiene como consecuencia la reducción de flexibilidad inmediata de la herramienta, ya que el desarrollo de software en los lenguajes fuertemente tipados conlleva una mayor cantidad de tiempo a la hora de especificar la relación, jerarquía y tipado de los datos, con el fin de aprovechar el aumento en velocidad que ofrecen.

Es por esta razón que se optó por lenguajes de guiones (*scripting*), como Python o Ruby, que sin perder generalidad en cuanto a la resolución de problemas, y a pesar de tener

una velocidad de cómputo más reducida, permiten el desarrollo de soluciones adaptadas a los problemas que surgen con una mayor flexibilidad y rapidez, ofreciendo además una mayor cantidad de funcionalidad estándar para el procesamiento de texto.

Se decidió utilizar Python para complementar la elección de herramienta de PLN que se hizo, en este caso la biblioteca NLTK para Python, como se mencionó anteriormente (1.4.5).

A continuación, en el Capítulo 2, se expone la problemática concreta del caso de estudio así como las soluciones implementadas con las herramientas escogidas. El Capítulo 3 expone los resultados alcanzados y realiza una comparativa, mientras que el Capítulo 4 extrae conclusiones sobre el trabajo realizado, y plantea futuras líneas de trabajo. Finalmente, el Capítulo 5 desglosa los recursos utilizados en la elaboración de este proyecto.

Capítulo 2

Caso de estudio

En este capítulo procederemos a explicar el caso de estudio escogido para el desarrollo del trabajo, así como la problemática asociada y las soluciones implementadas para el desarrollo de la misma.

2.1. Descripción del caso de estudio

El caso de estudio escogido consiste en la clasificación de los Trabajos de Fin de Grado (TFGs) que se encuentran en el Repositorio Institucional de la ULL, partiendo de la información que se puede recabar sobre los mismos. El objetivo de este caso de estudio es el de obtener un sistema de clasificación de TFGs que pueda servir como suplemento a la hora de asignar un área temática a un TFG cuando éste es inscrito a la plataforma.

2.2. Opciones de clasificación

La clasificación de textos es un subproblema del campo del Procesamiento del Lenguaje Natural, entre muchos otros como aquéllos discutidos en el capítulo anterior (2). Fue escogido debido a la facilidad y rapidez de su implementación con las herramientas disponibles a día de hoy, entre ellas el NLTK. Existen varias formas de realizar la clasificación de un texto, todas claramente atendiendo a las características del mismo, pero utilizando diferentes enfoques. En las siguientes secciones pasamos a describir los enfoques más utilizados para ello.

2.2.1. Enfoques estadísticos

Esta clase de enfoques hacen uso de las técnicas de Aprendizaje Automático, que hoy en día son las que gozan de mayor aceptación y crecimiento. Utilizan técnicas estadísticas para realizar estimaciones sobre un conjunto de datos del que se desconoce su clase, y se procede a determinar comparando las características del elemento actual con aquéllas de elementos previos de los que se conoce su clase. El éxito de este tipo de técnicas se basa en las características que sean detectadas y seleccionadas como relevantes para el problema de clasificación que se tenga.

Tomemos como ejemplo un hipotético estudio sobre los hábitos alimenticios de un grupo de animales. El color del pelaje de un animal no es relevante a la hora de inferir su dieta, pero en cambio la clase de dentadura que posea sí lo es. Por tanto, en este caso se deberá ignorar el color del pelaje pero tomar en cuenta el tipo de dentadura de los animales.

Para los casos de aprendizaje supervisado, existen elementos ya clasificados de manera correcta, ya sea por parte de otro programa o de manera manual. Estos elementos son utilizados para conformar un subconjunto de entrenamiento.

En estos casos de aprendizaje supervisado, es importante cuidarse de no sobreajustar el clasificador a la clasificación de los problemas de entrada. El problema del sobreajuste es un problema difícil de resolver objetivamente, y suele manejarse mejor cuando se tiene un cierto nivel de experiencia en tratar con él.

En el caso del aprendizaje no supervisado, no se poseen de datos de entrenamiento, por lo que normalmente recae sobre el algoritmo la diferenciación de los elementos en diferentes clases "a posteriori", es decir, clases que no habían sido definidas antes de la tarea de clasificación.

2.2.2. Naive Bayes

Los clasificadores de Bayes ingenuos (*Naive Bayes classifiers*) son clasificadores utilizados ampliamente en la actualidad para problemas de clasificación de textos, tanto en ámbitos académicos como industriales.

Los caracteriza una gran simplicidad, que conlleva la ventaja de que son de fácil comprensión y por tanto de fácil uso.

A pesar de esta simplicidad, los clasificadores de Bayes ingenuos son bastante eficaces y tienen una gran cantidad de aplicaciones prácticas.

Los clasificadores de Bayes ingenuos se basan en un modelo probabilístico, el Teorema de Bayes.

$$p(C_k|x) = \frac{p(C_k)p(x|C_k)}{p(x)} \tag{2.1}$$

Según este teorema, la probabilidad de un suceso C_k, dada una situación x viene determinada por el número de veces que se ha producido el suceso C_k en esa situación x, dividido por el número de veces total que se ha encontrado esta situación x (Eq. 2.1).

Aplicados al problema de la clasificación, nos permite inferir la posibilidad que tiene un elemento de pertenecer a cada una de las clases consideradas, teniendo en cuenta cada una de las características de este elemento como un suceso a priori.

Los clasificadores de Bayes ingenuos reciben su nombre debido a que toman en cuenta la contribución de cada característica hacia la posibilidad de una clasificación de manera independiente, en contraposición a correlacionar varias características para realizar una estimación más precisa a la hora de clasificar un elemento.

2.2.3. Enfoques analíticos

Los enfoques analíticos son aquéllos que intentan realizar un análisis morfosintáctico sobre el texto de entrada, obteniendo información relativa a las construcciones sintácticas de las oraciones y la morfología de las palabras utilizadas. Esta clase de enfoques se basa en el análisis morfosintáctico clásico, que lleva realizándose durante siglos de forma manual, como herramienta docente para dar a conocer las bases de un lenguaje natural.

Sin embargo, se trata de enfoques de mayor complejidad en comparación con los enfoques estadísticos, debido a que requieren la implementación de conocimientos sobre la gramática, sintaxis y semántica del lenguaje objetivo. Por ejemplo se deben de conocer todas las conjugaciones posibles de todos los verbos del lenguaje objetivo, y atender al contexto de las oraciones para utilizar la acepción correcta de palabras ambigüas.

Por otra parte, las herramientas de PLN deben contener corpus morfosintácticos y semánticos para cada uno de los idiomas, además de requerir para cada uno de ellos diferente para paliar las idiosincracias de cada uno de los lenguajes.

Debido a estas razones, y con motivo de atenerse al tiempo disponible, se ha decidido no adentrarnos en esta clase de análisis, que sin embargo resulta bastante prometedor.

2.3. Trabajos de Fin de Grado en la ULL: ¿Qué revela el análisis automático de texto sobre los mismos?

En nuestro caso de estudio, se utilizó como fuente el Repositorio Institucional de la Universidad de La Laguna (RIULL), conteniendo ésta los documentos que se deseaban explotar. Se ha realizado un desarrollo en base a aproximaciones, cada una de ellas con resultados que han sido comparados entre sí para constatar su eficacia.

2.3.1. Obtención de datos

El primer paso para realizar el Procesamiento del Lenguaje Natural consiste en obtenerlo. Normalmente se busca extraer información de una fuente para poder explotarla y obtener el texto a procesar. En nuestro caso de estudio, la obtención de información se centró en los Trabajos de Fin de Grado (TFG), usando como fuente de información el Repositorio Institucional de la Universidad de La Laguna (RIULL). El procedimiento utilizado para la obtención de información referente a estos documentos fue el uso de una herramienta de *web scraping* sobre la página web de la RIULL, cuyo uso se explica a continuación.

2.3.2. Obtención de datos con CasperJS

El proceso de obtención de datos se basó en el uso de la herramienta software CasperJS [7] para simular una navegación y recabar datos de las páginas web del RIULL de manera automatizada. La herramienta actúa de la misma manera que un navegador de uso personal, enviando peticiones HTTP, recibiendo información y prosiguiendo de la manera que se le haya indicado. En este caso se implementó la recogida de datos partiendo de la página en la que se muestran los TFG por orden de publicación, y se realiza la paginación hasta haber recuperado los enlaces individuales para cada uno de los TFG.

Es entonces cuando se procede a la extracción de los metadatos de cada página individual de TFG, obteniendo los datos relevantes de cada TFG como su título, descripción y clasificación.

2.3.3. Extracción de la información

CasperJS trabaja mediante la ejecución de un *script*(guión) en lenguaje JavaScript, que le es proporcionado como entrada.

Antes que nada, se debe crear una instancia del navegador CasperJS, incorporando la librería *casper* y pasando las opciones de creación como argumento.

```
1 var casper = require('casper').create({
2 //  verbose: true,
3   logLevel: "debug"
4 });
```

LISTADO 2.1: Creación del navegador CasperJS.

En este caso se le pasa el argumento *verbose* para indicar que se desea que muestre mensajes informativos al realizar las acciones que se le pasen, mientras que el argumento *logLevel* sirve para especificar el detalle que precisamos de esos mensajes.

```
1 // Enlaces hacia las paginas de los TFGs.
2 var links = [];
3 // Offset navegado hasta el momento en el catalogo de TFGs.
4 var offset = 0;
```

LISTADO 2.2: Estructuras de datos para la navegación por la página.

El navegador obtendrá los enlaces a cada página individual de TFG, los cuales deberá mantener en memoria hasta la finalización de la ejecución del programa de extracción de datos. Es por ello que se define la variable *enlaces* que los contendrá. Por otra parte, al realizar el paginado se debe conocer la posición del navegador de manera inmediata, por lo que se mantiene y actualiza en la variable *offset*.

```
1 var titles = [];
2 var descriptions = [];
3 var tags = [];
```

LISTADO 2.3: Estructuras de datos para las características de cada TFG.

Las características que se han tomado como relevantes en la primera aproximación han sido el título, la descripción y las etiquetas que se han dado a cada TFG en el repositorio. Estas se mantendrán en variables separadas.

```
1 // En cualquier pagina de TFGs recientes de un grado,
2 // obtener los enlaces a las paginas de los TFGs.
3 function scrapeLinks() {
4     var links = $('li.ds-artifact-item a');
5     return Array.prototype.map.call(links, function(e) {
6         return e.getAttribute('href');
7     });
8 }
```

LISTADO 2.4: Función de obtención de enlaces individuales.

Durante la paginación, cada una de las páginas muestra enlaces individuales hacia cada uno de los Trabajos de Fin de Grado. Para ello, estando el navegador en cualquier página del catálogo, obtenemos los elementos *href* y con ellos todos los enlaces de la vista actual.

```
1 // Estando en el catalogo, obtiene el enlace a
2 // la siguiente pagina del catalogo si la hay.
3 function scrapeNextPageLink(){
4   if ($('.next.pull-right.disabled').length > 0) return null;
5   return $('a.next-page-link')[0].href;
6 }
```

LISTADO 2.5: Función de paginado.

Durante la paginación, se debe conocer si existe una página subsecuente a aquélla en la que se encuentra el navegador actualmente. Para ello inspeccionamos si el botón que permite paginar a la siguiente página está bloqueado en la vista. De ser este el caso, nos encontramos en la última página. De otro modo, conseguimos el nuevo enlace del elemento de paginación.

```
/**
* Obtiene todos los enlaces de los TFGs de un Grado.
* Se presupone que Casper se encuentra en
* la primera pagina de TFGs recientes del Grado.
*/
function casperLinksOnThisBachelor(){
    var newLinks = casper.evaluate(scrapeLinks);
    links = links.concat(newLinks);
    var nextLink = casper.evaluate(scrapeNextPageLink);

    casper.echo('nextLink: ' + nextLink);
    if (nextLink) {
        // casper.thenClick(nextLink);
        casper.thenOpen(nextLink);
        casper.then(casperLinksOnThisBachelor);
    } else {
        casper.echo("END");
    }
}
```

LISTADO 2.6: Función de obtención de todos los enlaces de TFG.

Aplicamos las anteriores funciones en lo que viene a ser el bucle principal del programa, una función recursiva que se detiene cuando el navegador ha llegado a la última página.

FIGURA 2.1: Vista en la que se muestran los metadatos de un TFG.

Show simple item record

(Título del TFG)

dc.contributor.author	(Nombre del autor)	es_ES
dc.date.accessioned	2016-08-01T08:35:11Z	
dc.date.available	2016-08-01T08:35:11Z	
dc.date.issued	01/08/2016 09:30	es_ES
dc.identifier.uri	http://riull.ull.es/xmlui/handle/915/2910	
dc.title	(Título del TFG)	es_ES
tfg.interested.identificationNumber	(DNI del autor)	es_ES
tfg.interested.name	(Nombre del autor)	es_ES
tfg.name	(Título del TFG)	es_ES
tfg.codTitle	G026	es_ES
tfg.codCenter	1353	es_ES
tfg.documentId	697144	es_ES
tfg.securityCode	TEJOOHRIcVQ=	es_ES
tfg.nameTitle	GRADO EN INGENIERÍA INFORMÁTICA	es_ES

FIGURA 2.2: Modelo DOM subyacente a la vista.

```
<h2 class="page-header first-page-header">(Título del TFG)</h2>
<div class="ds-table-responsive">
<table class="ds-includeSet-table detailtable table table-striped table-hover">
<tbody>
<tr class="ds-table-row odd ">
<td class="label-cell">dc.contributor.author</td>
<td class="word-break">(Nombre del autor)</td>
<td>es_ES</td>
</tr>
<tr class="ds-table-row even ">
<td class="label-cell">dc.date.accessioned</td>
<td class="word-break">2016-08-01T08:35:11Z</td>
<td></td>
</tr>
<tr class="ds-table-row odd "></tr>
<tr class="ds-table-row even ">
<td class="label-cell">dc.date.issued</td>
<td class="word-break">01/08/2016 09:30</td>
<td>es_ES</td>
</tr>
<tr class="ds-table-row odd ">
<td class="label-cell">dc.identifier.uri</td>
<td class="word-break">http://riull.ull.es/xmlui/handle/915/2910</td>
<td></td>
</tr>
<tr class="ds-table-row even ">
<td class="label-cell">dc.title</td>
```

```
1 // En una pagina del TFG, obtener el valor del campo 'str' (si lo hay
    ).
2 function scrapePageValue(str){
3     var datum = $('td.label-cell:contains(' + str + ') + td')[0].
    innerHTML;
4     if (datum === null || datum === undefined || datum === "")
5         datum = "---";
6     return datum;
7 }
```

LISTADO 2.7: Función de obtención de metadato por nombre de etiqueta.

Esta función se basa en la vista de la página para obtener el texto perteneciente a una etiqueta. Por ejemplo, para el caso de la Fig.2.2, si se la función recibe `author` como argumento, extraerá del DOM el valor `(Nombre del autor)`.

```
1  // En una pagina del TFG, obtener el valor de varios campos 'str' (si
       los hay).
2  function scrapePageValues(str){
3      var data = $('td.label-cell:contains(' + str + ') + td');
4      var data2 = [];
5      for (var i = 0; i < data.length; i++){
6        data2.push(data[i].innerHTML);
7      }
8      if (data2.length < 1) data2.push("---");
9      return data2;
10 }
```

LISTADO 2.8: Función de obtención de metadatos por nombre de etiqueta similar.

Definimos además una función similar a la anterior, con la diferencia de que inspecciona la página para encontrar múltiples datos definidos bajo el mismo nombre. Esto es importante dado que existen metadatos que están definidos bajo campos de mismo nombre, por ejemplo: dos campos de "tema" con distintos valores, para expresar que un TFG forma parte de múltiples temas.

```
1  function completeLink(currentValue, index, array){
2    return 'http://riull.ull.es'.concat(currentValue).concat('?show=
       full');
3  }
```

LISTADO 2.9: Concatenación de la URL absoluta con el enlace relativo.

Esta función se utiliza para poder navegar a todos los enlaces relativos que se obtienen de la extracción de la vista de navegación.

2.3.4. Almacenamiento de la información

Se decidió guardar la información obtenida del proceso de extracción de texto en archivos locales en el disco duro del ordenador utilizado para la realización del trabajo.

El formato utilizado para los archivos de texto es CSV (*Comma Separated Values*), es decir, archivos de texto planos- Se trata de un formato sencillo en el cual cada elemento se encuentra en una línea diferente, y las características de dichos elementos separados por comas.

La razón por la que se utilizó éste formato se debe a la mayor comodidad que aporta en un caso de estudio restringido, con una cantidad de datos relativamente pequeña. Otra

razón por la que se eligió el formato CSV es la disponibilidad de librerías Python que trabajan en este formato, incrementando la facilidad y rapidez de desarrollo del sistema de almacenamiento.

2.3.5. Clasificación del texto

```
1  def cargar_instancias_de_csv(ruta):
2      """
3      Cargamos las instancias de un archivo CSV.
4
5      :param ruta: Ruta del archivo csv que se leer .
6      :return: Una lista con cada una de las filas del csv.
7      """
8      instancias = []
9      with open(ruta) as csvfile:
10         reader = csv.DictReader(csvfile)
11         for row in reader:
12             instancias.append((row['Clasificaci n'], row['T tulo'],
       row['Descripci n']))
13     return instancias
```

Listado 2.10: Función para cargar instancias de un archivo CSV.

Esta función se utiliza para la carga en memoria de las instancias a clasificar.Las instancias ya deben de haber sido descargadas en un CSV con el formato apropiado, tal y como se explicó en el apartado 2.3.3.

```
1  def cadena_de_palabras(instancias, indices_de_textos):
2      """
3      Creamos una larga cadena con todas las palabras
4      separadas entre si por espacios.
5
6      :param instancias: Listas que representan a los elementos a
       clasificar.
7      :param indices_de_textos: Lista con los indices de los elementos
       de las
8      instancias que se van a extraer.
9      :return: Una lista con cada una de las filas del csv.
10     """
11     larga_cadena = ''
12     for row in instancias:
13         for indice in indices_de_textos:
14             larga_cadena += ' '
15             larga_cadena += row[indice]
16     return larga_cadena
```

Listado 2.11: Función que recopila un conjunto de palabras comunes

Esta función obtiene las características textuales y las guarda en un conjunto comunal de palabras para luego determinar las más utilizadas. En este caso, toman el texto tanto del título de un TFG como de su descripción si la hay, y los suma al dicho conjunto de palabras comunes mencionado.

```
1  def palabras_alfanum(lista_palabras):
2      """
3      Eliminamos del conjunto global de palabras
4      aquellas cadenas que no sean alfanumericas.
5      :param lista_palabras: Una lista de palabras
6      :return: Una lista con las cadenas que son alfanumericas.
7      """
8      alfanum = [];
9      for palabra in lista_palabras:
10         if re.match('\w+', palabra):
11             alfanum.append(palabra)
12     return alfanum
```

LISTADO 2.12: Función que retorna palabras alfanuméricas

Esta función realiza una criba del conjunto de palabras obtenido en la función `cadena_de_palabras`, descartando aquéllas palabras que contengan algún carácter no alfanumérico.

```
1  def clasificaciones_invalidas(instancias,clases_invalidas=["null"],
       ind_clase=0):
2      """
3      Separamos las instancias con una clasificacion valida de las que
       no la tienen.
4
5      :param instancias: Instancias a separar.
6      :param ind_clase: Indice de la instancia que contiene su
       clasificaci n.
7      :param clases_invalidas: Valores de clasificaci n inv lidos.
8      :return: En orden: las instancias inv lidas , y las instancias
       v lidas.
9      """
10     invalidas = [];
11     for c in instancias:
12         if (c[ind_clase] in clases_invalidas):
13             invalidas.append(c)
14             instancias.remove(c)
15     return invalidas, instancias
```

LISTADO 2.13: Función que separa las instancias con clasificación inválida.

Esta función recibe un conjunto de instancias, y las separa según la clase a la que pertenezca. A éste efecto, se proporciona como argumento una lista de las clases que se

consideran inválidas. La utilidad de esta función es que nos permite descartar instancias que no hayan sido clasificadas correctamente en el Repositorio Institucional de la ULL.

```
1  def rasgos_del_texto(texto, listado_palabras):
2      """
3      A partir de un texto, creamos un hash con las caracteristicas
4      de dicho texto. En este caso si contiene cada una de las palabras
       .
5
6      :param texto: Un texto a clasificar
7      :param listado_palabras: Un listado de palabras globales que
       sirve
8          para definir las caracteristicas del texto.
9      :return: Un diccionario con las caracteristicas del texto.
10     """
11     if(not texto):
12         print("Texto nulo")
13         return None
14     palabras_del_texto = set(texto)
15     caracteristicas = {}
16     for p in listado_palabras:
17         caracteristicas['contiene({})'.format(p)] = (p in
       palabras_del_texto)
18     return caracteristicas
```

Listado 2.14: Función que extrae las características de un texto.

Esta función recibe un texto y entrega las características del mismo. En este caso consideramos como características el hecho de que el texto contenga o no cada una de las palabras más comunes consideradas. Para ello, también se le proporciona esta lista de palabras comunes a esta función.

```
1  def listado_palabras(instancias, ind_texto=1, ind_clase=0, invalidos
       =["null"]):
2      """
3      :param ruta: Ruta del CSV con las instancias a clasificar.
4      :param n: Numero de palabras mas comunes a tener en cuenta.
5      Cargar las instancias de un CSV, ...
6      """
7      # Creamos una larga cadena con todas las palabras
8      # separadas entre si por espacios.
9      conjunto_palabras = clas.cadena_de_palabras(instancias, [2])
10
11     # De esa larga cadena, obtenemos una lista de palabras
12     # gracias al tokenizador, que sabe donde cortar.
13     conjunto_palabras = nltk.word_tokenize(conjunto_palabras)
14
15     # print(conjunto_palabras)
16
17     # Eliminamos del conjunto global de palabras
18     # aquellas cadenas que no sean alfanumericas.
19     conjunto_palabras = clas.palabras_alfanum(conjunto_palabras)
20
21     # Hacemos una distribucion de frecuencia de las palabras en el
       corpus.
22     listado_palabras = nltk.FreqDist(w.lower() for w in
       conjunto_palabras)
23
24     return listado_palabras
```

LISTADO 2.15: Función que retorna un listado de las palabras comunes.

Función que retorna un listado de las palabras comunes a todas las instancias a clasificar. Hace uso de varias de las funciones anteriormente expuestas para procesar el conjunto de palabras que retorna.

```
1  def clasificacion_binaria_palabras(distr_frec, instancias, n,
       clases_invalidas):
2      """
3      :param distr_frec: Ruta del CSV con las instancias a clasificar.
4      :param n: Numero de palabras mas comunes a tener en cuenta.
5      :param clases_invalidas: Las instancias que las tengan no estan
       clasificadas.
6      """
7
8      # Escogemos las N palabras mas comunes.
9      mas_comunes = list(distr_frec)[:n]
10
11     invalidas, validas = clas.clasificaciones_invalidas(instancias,
       clases_invalidas)
12
13     # A partir de un texto, creamos un hash con las caracteristicas
14     # de dicho texto. En este caso si contiene cada una de las
       palabras comunes.
15     inst_validas = [(clas.rasgos_del_texto(d, mas_comunes), c) for (c
       ,t,d) in validas]
16     inst_invalidas = [(clas.rasgos_del_texto(d, mas_comunes), c) for
       (c,t,d) in invalidas]
17     half = math.floor(len(instancias) / 2)
18     train_set, test_set = inst_validas[:half], inst_validas[half:]
19     # train_set, test_set = inst_validas, inst_invalidas
20     classifier = nltk.NaiveBayesClassifier.train(train_set)
21
22     resultado = {}
23     resultado['precision'] = nltk.classify.accuracy(classifier,
       test_set)
24     # print("print(nltk.classify.accuracy(classifier, test_set)):" +
       str(resultado['precision'])
25     return resultado
```

LISTADO 2.16: Función que clasifica una instancia en base a las palabras comunes que contiene

Función que clasifica las instancias en base a las N palabras más comunes de entre las palabras que se encuentran en el conjunto de las instancias. La clasificación se lleva a cabo utilizando un clasificador bayesiano ingenuo que toma como características la presencia o ausencia de cada palabra común en la instancia a clasificar.

```
# Cargamos las instancias de un archivo CSV.
instancias = clas1.cargar_instancias_de_csv("tfg.csv")

listado_palabras = clas2.listado_palabras(instancias)

clases_invalidas = ["null", "Inform tica", "Logicales de ordenadores
    ",
    "Logicales de ordenador"]

str = []
plot_dat_x = []
plot_dat_y = []
for i in range(100,2000,100):
    print(i)
    instancias2 = list(instancias)
    res = {}
    plot_dat_x.append(i)
    res = clas2.clasificacion_binaria_palabras(listado_palabras,
    instancias2, i, clases_invalidas)
    res['pal_comunes'] = i
    print(res.get('precision'))
    plot_dat_y.append(res.get('precision'))
    str.append(res)
print(str)

plt.plot(plot_dat_x, plot_dat_y)
plt.ylabel('N mero de palabras comunes consideradas')
plt.ylabel('Precisi n del clasificador')
plt.show()
```

LISTADO 2.17: Batería de pruebas de clasificación

Éste es el código que ejecuta la batería de pruebas de clasificación. Se pasan como clases inválidas las expuestas, por considerarse demasiado amplias. Además se proporciona el valor de corte de N palabras comunes, que iniciará en 100, y cambiará en incrementos de 100 hasta alcanzar las 2000 palabras comunes. Esto se ejecuta con el objetivo de determinar el comportamiento del clasificador para distintos valores de corte. Finalmente se genera y muestra una gráfica de la precisión obtenida por el clasificador.

Capítulo 3

Resultados y discusión

En este capítulo se expone los resultados alcanzados y se realiza una comparativa entre los métodos utilizados en base a dichos resultados. Además de esto, se destacan los problemas encontrados en las diferentes fases del proceso, y cómo fueron remediados.

3.1. Conclusiones sobre la extracción

La extracción de información fue implementada utilizando selectores de jQuery, para la obtención de los elementos que contienen la información que nos interesa de la vista presente.

Es evidente que este acercamiento es ciertamente inestable, y aumenta la necesidad de mantener el código, debido a que se basa en la jerarquía HTML y estilos CSS de la página que nos es devuelta en una respuesta HTTP. Esta vista puede ser modificada en cualquier momento por los administradores o propietarios de una página web, rompiendo la dependencia y con ella la extracción automatizada.

En el caso de realizar la extracción de texto en otras páginas web, o incluso la misma página si en un futuro es modificada su vista, deberá modificarse el programa para adaptarse a las nuevas características y disposición que tomen los elementos que contengan la información, siendo esto incómodo y poco óptimo.

Sin embargo, es la única forma de extraer información de una página, servicio, o aplicación web que no disponga de una API (*Application Programming Interface*) que presente la información de manera independiente a la vista o interfaz de usuario.

Otra ventaja del uso de APIs es que evita al interesado descargar la página al completo, de manera que evita descargar archivos pesados que puedan existir en la vista como imágenes, anuncios y etc.

A día de hoy, la gran mayoría de páginas y aplicaciones web no poseen una API subyacente, en gran parte debido a la falta de necesidad que se tiene de ello, ya que no están pensadas para obtener la información mediante métodos programáticos.

Es ideal que todo servicio o página web disponga de una API como punto de entrada de datos a una aplicación, pero en su defecto también es factible que en la vista se realice un etiquetado semántico de los elementos, ya sea mediante HTML o CSS, proporcionando cierta autonomía de la vista de la página.

A pesar de esto, en la mayoría de los casos los administradores y propietarios de las páginas web no tienen incentivos ni necesidad de suplir esta necesidad, por lo que se deberá seguir utilizando estas técnicas de extracción de datos o *web scraping*.

3.2. Conclusiones sobre la clasificaión

El primer acercamiento de clasificación se realizó utilizando los metadatos disponibles para cada uno de los TFGs, concretamente el título y el abstracto. Se utilizó el campo de "tema"para determinar la clasificación que se le había dado a cada TFG en el RIULL.

Se realizó una batería de pruebas, variando el número de palabras comunes a considerar. Se empezó utilizando las 100 palabras más comunes, aumentando esto en incrementos de 100 hasta llegar a un clasificador que considera las 2000 palabras más comunes entre los textos a clasificar.

Se tiene aquí un histograma de las palabras de mayor frecuencia entre las instancias de la primera aproximación del caso de estudio, que se corresponden con los metadatos de los TFG del Grado en Ingeniería Informática. Como se puede apreciar, la información es muy pobre, dado que las palabras más comunes son preposiciones, encabezadas por `de`, que aparece 187 veces. La palabra `aplicación` es la primera palabra significativa y aparece 18 veces, seguida de `sistema` con 17 apariciones. Las palabras relevantes se repiten muy pocas veces en el corpus creado, con lo cual es difícil realizar inferencias en bases a ellas.

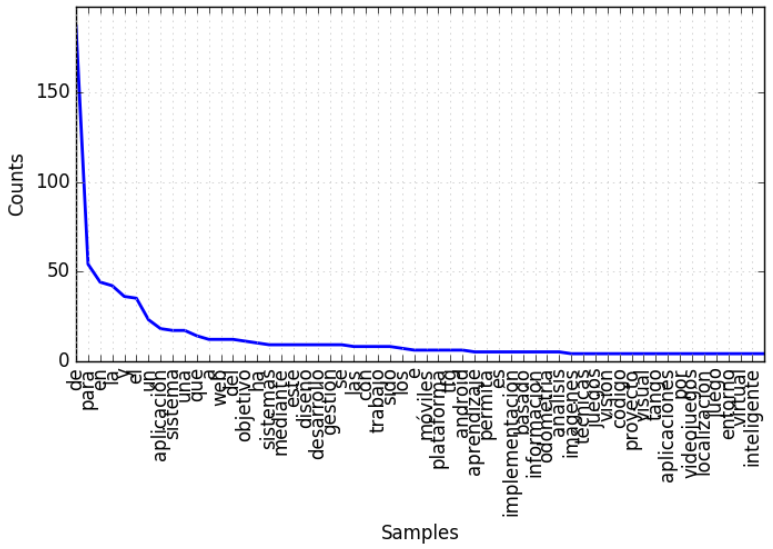

FIGURA 3.1: Palabras de mayor frecuencia entre las instancias de la primera aproximación.

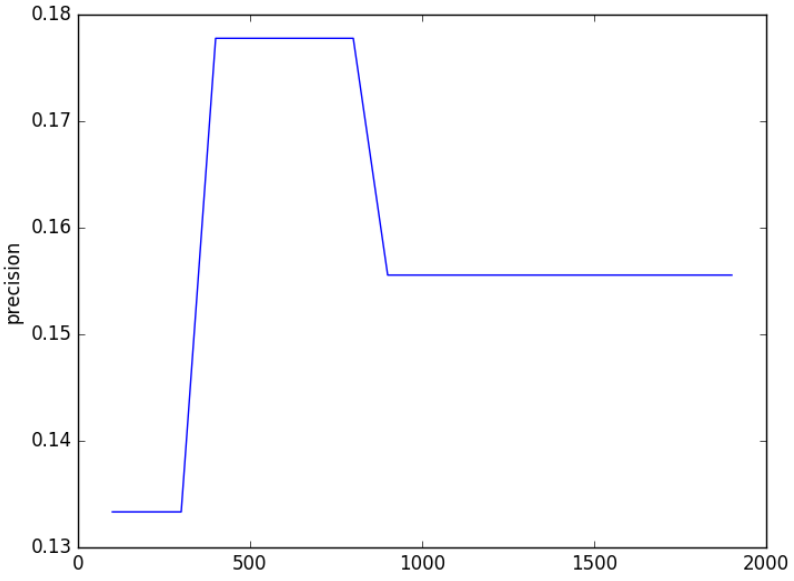

FIGURA 3.2: Precisión de la clasificación Naive Bayes usando metadatos de los TFG de Ing. Informática.

Como vemos, la precisión del clasificador oscila entre 0.13 y 0.18, lo que indica que el clasificador acierta entre el 13 % y 18 % de las veces. Estos resultados son bastante pobres, y esto se debe a la poca información que proporcionan los metadatos existentes sobre los textos. La falta de información se debe a tres factores: en primer lugar, los títulos no suelen sobrepasar las 20 palabras, lo que hace difícil la clasificación por conteo

de palabras comunes. El segundo problema es que en muchos casos, la descripción o abstracto del TFG no se encuentra como metadato del TFG en el RIULL, debido a ser un campo opcional. Finalmente, se tiene un número elevado de clases, muchas de ellas conteniendo a un único TFG, lo que reduce las posibilidades de caracterizar una clase y aumenta las probabilidades de cometer una asignación errónea.

Por otra parte, podemos observar cómo decrementa la precisión del clasificador a partir de un número de palabras dado. Podemos inducir de esto que existe un límite de palabras a considerar óptimo, tras el cual se consideran demasiadas características de los elementos como para dar una respuesta fiable.

El segundo acercamiento realizado consistió en la obtención de los metadatos de todos los Trabajos de Fin de Grado en el RIULL. Esto resultó en un incremento considerable de la cantidad de instancias, pasando de los 91 TFG de Ing. Informática a 350 documentos. Al igual que en el caso más reducido de los TFG de Ing. Informática, la gran mayoría de los Trabajos de Fin de Grado no contienen un abstracto como metadato, provocando que nos limitemos a utilizar únicamente sus títulos en su mayor parte.

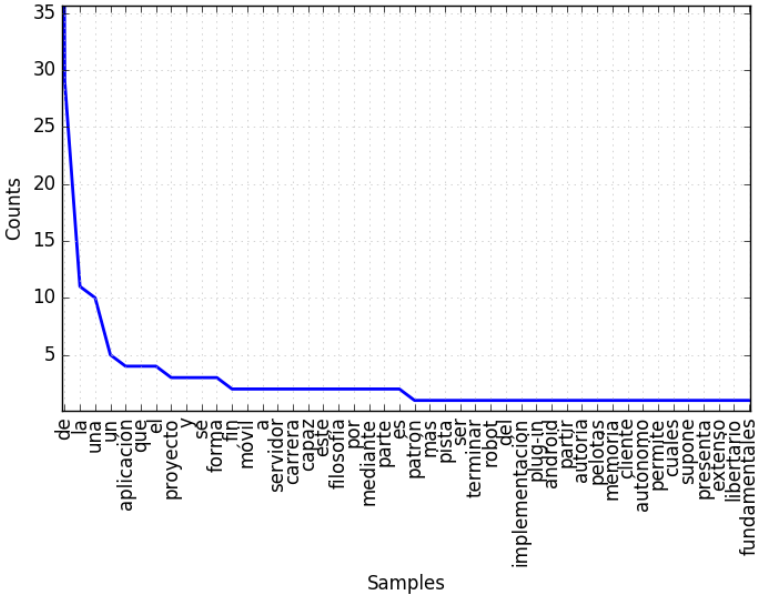

FIGURA 3.3: Palabras de mayor frecuencia entre las instancias de la segunda aproximación.

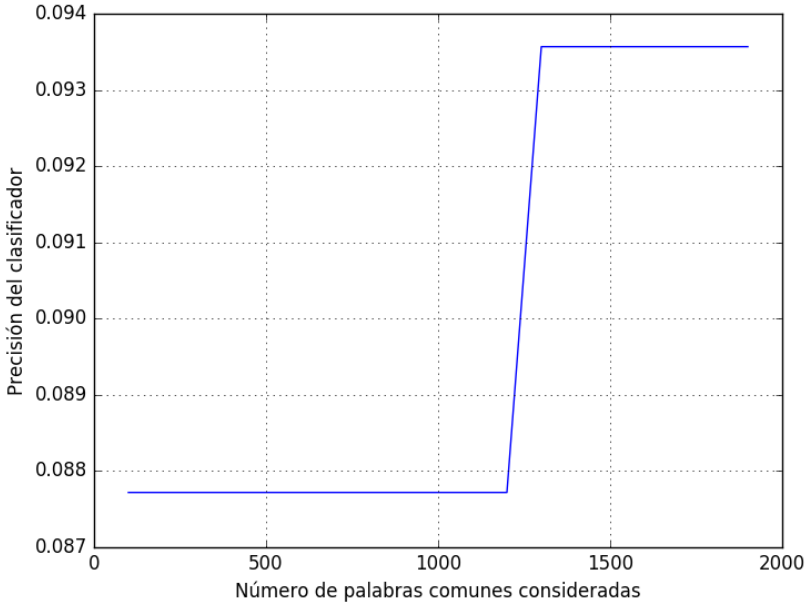

FIGURA 3.4: Precisión de la clasificación Naive Bayes usando metadatos de todos los TFG del RIULL.

Como podemos apreciar, el leve incremento en la cantidad de datos disponibles no evita que esta aproximación tenga menos éxito que la anterior en base al subproblema escogido. Mientras el primer clasificador oscilaba entre el 13 % y 18 % de acierto, éste clasificador se mueve entre el 8 % y 10 % de efectividad. Este clasificador tiene una menor eficacia que el del anterior caso, pero hay que tener en cuenta que hace frente a un problema de mayor dificultad: en la aproximación anterior tratábamos con 14 clases, mientras que en éste caso se dan 28 clases distintas.

3.3. Problemas encontrados

3.3.1. Obtención del texto

Uno de los problemas a la hora de realizar la extracción de texto en la página del Repositorio Institucional de la ULL se producía cuando se realizaba un número excesivo de consultas a la página. Esto provocaba que las peticiones enviadas desde la IP del ordenador de pruebas fuesen ignoradas por el servidor. Se puede conjeturar que esto sea resultado de un mecanismo de defensa del servidor institucional para contrarrestar ataques DoS (*Denial of Service*).

3.3.2. Clasificación del texto

Para la clasificación de los textos, uno de los problemas más importantes fue el exceso de clases en comparación con el número de instancias. Muchas de estas clases contenían a un único elemento, lo que dificultaba el proceso de clasificación al no obtener suficiente información determinante para caracterizar cada tipo de clase.

Como se comentó en el apartado 3.2, en muchos casos, la descripción o abstracto del TFG no se encuentra como metadato de los TFG de la RIULL, debido a que no es obligatorio rellenar este campo para dar de alta un TFG en la plataforma. Esto redujo drásticamente la capacidad de clasificación sobre los metadatos de los TFGs en comparación con el caso en que todos los documentos tuviesen un resumen o abstracto asignado en el servicio.

Las bajas tasas de acierto resultan poco alentadoras, pero se deben a la presencia de clases con pocas entidades representativas, así como la falta de información individual sobre cada TFG en forma de metadatos. Esto se puede paliar ignorando las clases que no posean un número mínimo de instancias, centrándonos en clasificar aquéllas clases que podamos caracterizar correctamente gracias a su número elevado de integrantes. La otra solución es la obtención de más información característica sobre cada instancia individual, lo que se consigue extrayendo información textual de los documentos infijos (extensión PDF) aparte de sus metadatos.

3.3.3. Realización de la memoria

Por defecto *LaTeX* no permite el uso de tildes y caracteres españoles, por lo que fue necesario incluir el paquete `inputenc` y `babel` en la configuración del proyecto, con las opciones `utf8` y `spanish`, respectivamente.

A la hora de mostrar el código del trabajo realizado, el primer método considerado fue la realización de capturas del código en el editor de texto, para insertarlas en el documento como imágenes. Esto conllevaba una falta de uniformidad, debido al diferente tamaño de las imágenes, así como dificultad a la hora de cambiar la apariencia de las mismas.

Como alternativa, se utilizó un paquete *LaTeX* llamado `listings`, que permite la introducción de código con formato, y proporciona funcionalidad que ayuda a su lectura como el resaltado (*highlighting*) de palabras reservadas. Tuvo que ser modificado para permitir el resaltado de código JavaScript, cuya sintaxis no se incluía por defecto, para resaltar los listados apropiados.

Capítulo 4

Conclusiones

4.1. Conclusiones y Trabajos Futuros

Como líneas futuras de trabajo, se considera la extracción de texto de los Trabajos de Fin de Grado en formato infijo (extensión PDF), con el fin de obtener una cantidad suficiente de palabras para caracterizar cada una de las instancias a clasificar.

Por otra parte, se tiene en cuenta la posibilidad de descargar datos de TFGs de diferentes disciplinas a la Informática, para aumentar la diferenciación entre instancias y esperando que la capacidad de clasificación incremente de manera considerable.

Finalmente, se puede utilizar un clasificador estadístico diferente al clasificador bayesiano ingenuo, o también considerar otros métodos de definición y extracción de características.

Las técnicas de Procesamiento del Lenguaje Natural tienen hoy en día un gran número de aplicaciones prácticas, gracias a la disponibilidad de una gran cantidad de datos en Internet y a la creciente capacidad de procesamiento tradicional y en la nube (*cloud computing*), dando soporte a las técnicas de aprendizaje estadístico que gozan de gran aceptación en la actualidad.

La herramienta que goza de mayor popularidad actualmente tanto para el Procesamiento del Lenguaje Natural como otras ramas de la Computación es el Aprendizaje Automático, gracias a la mayor cantidad de información y capacidad de procesamiento disponibles.

Esta herramienta ha permitido un incremento considerable de las aplicaciones prácticas del Procesamiento del Lenguaje Natural durante la última década. Esto ha creado nuevos

puestos de trabajo en grandes compañías y pequeñas *startups* de nueva creación que han decidido tomar la oportunidad que estas nuevas tecnologías brindan.

Esta inversión en el área del Procesamiento del Lenguaje Natural también se ve reflejada en el ámbito académico, y producirá durante las próximas décadas nuevas herramientas y aplicaciones para la sociedad, en sinergia con los campos de la Interacción Persona-Computador y la Inteligencia Artificial.

4.2. Conclusions and Future Work

For future lines of work, we will consider the extraction of the Final Degree Projects in infix format (PDF extension), in order to obtain a sufficient amount of words so as to characterize each of the instances that are to be classified.

We also take into account the possibility of downloading data from Final Degree Projects of fields different from Computer Science, in order to increase the differentiation between instances and in hopes of increasing the classification capacity considerably.

Finally, a classifier different from the Naive Bayes classifier can be used, or different methods of feature definition and extraction can be considered.

At present, Natural Language Processing techniques have a wide range of practical applications, thanks to the availability of a vast amount of data on the Internet and the increasing processing capabilities of computers, both locally and in cloud computing, supporting the statistical learning techniques that are widely accepted to date.

The tool which has garnered the most popularity in recent years, in the field of Natural Language Processing as well as other fields of Computer Science, is Machine Learning, thanks to the higher amount of information and computational power currently available.

This tool has allowed for a significant increase of the practical applications of Natural Language Processing during the last decade. This has created new jobs in large companies and newly-created *startups* that have seized the opportunity that these new technologies bring.

This investment in the area of Natural Language Processing is also reflected in the academical world, and in the next decades it will provide new tools and applications for society, in synergy with the fields of Human-Computer Interaction and Artificial Intelligence.

Capítulo 5

Presupuesto

Aquí se expone el presupuesto equivalente al trabajo realizado. No se produjeron gastos de infrastructura puesto que los servicios de Internet y red eléctrica utilizados fueron el doméstico y el de dominio público. No se dan costes de equipo puesto que el ordenador utilizado fue el de uso personal, mientras que el software utilizado es libre y por lo tanto no se incurrió en gastos.

Cuadro 5.1: Presupuesto

Descripción	Costo por unidad	Cantidad	Costo total
Horas de trabajo	9€	300	2700€
Total:	—	—	2700€

Apéndice A

Abreviaturas

- **API**: ***A****pplication* ***P****rogramming* ***I****nterface*
- **CSS**: ***C****ascading* ***S****tyle* ***S****heets*
- **CSV**: ***C****omma* ***S****eparated* ***V****alues*
- **DOM**: ***D****ocument* ***O****bject* ***M****odel*
- **HTML**: ***H****yper****T****ext* ***M****arkup* ***L****anguage*
- **HTTP**: ***H****yper****T****ext* ***T****ransfer* ***P****rotocol*
- **NLTK**: ***N****atural* ***L****anguage* ***T****ool****K****it*
- **PDF**: ***P****ortable* ***D****ocument* ***F****ormat*
- **PLN**: **P**rocesamiento del **L**enguaje **N**atural
- **RIULL**: **R**epositorio **I**nstitucional de la **U**niversidad de **L**a **L**aguna
- **TFG**: **T**rabajo de **F**in de **G**rado

Apéndice B

Glosario

API: *Application Programming Interface*, interfaz que permite la interacción con un sistema informático de manera programática mediante código, sin necesidad de interactuar de forma manual a través de una interfaz de usuario.

Metadatos: Datos que acompañan a un archivo o documento informático, contextualizando su creación y denotando sus características, sin llegar a formar parte del contenido del documento en sí.

Parser: Programa que recibe información de manera secuencial y tiene como cometido desglosar la información que se le pasa como entrada en datos estructurados en la memoria de un sistema.

Web scraping: Procedimiento mediante el cual se obtiene información de una página web, obteniéndose a través del protocolo HTTP, y luego extrayendo la información relevante mediante expresiones regulares o *parsers* de DOM o HTML.

Bibliografía

[1] Universidad de La Laguna. Repositorio Institucional de la Universidad de La Laguna. 2010.

[2] Apache Software Foundation. Apache Lucene. 2004.

[3] Apache Software Foundation. Apache Open NLP. 2010.

[4] Mihai Surdeanu John Bauer Jenny Finkel Steven J. Bethard Manning, Christopher D. and David McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[5] Sociedad Española para el Procesamiento del Lenguaje Natural. Revista SEPLN nº56. 2016.

[6] Ewan Klein Steven Bird and Edward Loper. *Natural Language Processing With Python*. 2009.

[7] CasperJS Team. CasperJS Website. 2011.

[8] The GATE Project Team. General Architecture for Text Engineering.

[9] The GATE Project Team. The GATE Project Team.