Universidad Carlos III de Madrid

Escuela Politécnica Superior

Categorización de textos científicos mediante Aprendizaje Automático

Luna Fidalgo Manchón

Doble Grado en Ingeniería Informática y Administración de Empresas

Tutor: Raquel Fuentetaja

22 de junio de 2016

Resumen

El presente documento se corresponde con el Trabajo de Fin de Grado elaborado para finalizar el Doble Grado de Ingeniería Informática y Administración de Empresas, siendo este el referido al primer grado.

En él, se ha recogido el proceso llevado a cabo para el desarrollo de un software que genere automáticamente las palabras clave necesarias para etiquetar un artículo científico a partir de su resumen. Para ello, ha sido necesario, en primer lugar, conocer el Estado de la Cuestión, es decir, investigar qué era exactamente un problema de clasificación multietiqueta y de qué técnicas se disponían para resolverlo. Además, también fue necesario averiguar cómo se podía transformar un texto para poder generar de él una serie de atributos que explicaran su contenido.

Una vez se tenían los conocimientos suficientes, se pudo pasar al análisis, diseño e implementación del sistema, fases que han sido debidamente documentadas, y puestas a prueba en la evaluación. Esto se ha realizado mediante la generación de cuatro módulos: extracción de la información, elaboración de los conjuntos de entrenamiento, la construcción de los modelos y la evaluación.

En cuanto a las pruebas de comportamiento realizadas, se detectó que la labor de aprendizaje se dificultaba enormemente a medida que las etiquetas estaban muy relacionadas, pues sus atributos no eran lo suficientemente diferentes cómo para poder diferenciarlos con claridad.

Este trabajo se cierra con un capítulo de conclusiones, donde se ha realizado un repaso de los aspectos más importantes detectados durante el desarrollo, así como las limitaciones del sistema y las líneas futuras de investigación.

Palabras clave: aprendizaje automático, clasificación multietiqueta, *text mining*.

"No sirve de nada ir deprisa si no sabes dónde vas.
Lo importante es caminar en la dirección correcta."

Índice general

Índice de figuras

Índice de tablas

Glosario

AA - Aprendizaje Automático.
IA - Inteligencia Artificial.
MLC - *Multi-Label Classification* o Clasificación Multietiqueta.
SLC - *Single-Label Classification* .
IEEE - *Institute of Electrical and Electronics Engineers*.
API - *Application Programming Interface*.
BR - *Binary Relevance*.
LP - *Label Power Set* .
ECC - *Classifier Chains*.
RAkEl - *Random k-LabEl Pruned Sets*.
BPNN - *Back Propagation Neural Network*.

Capítulo 1

Introducción

1.1. Preámbulo

Hoy en día, aunque mucha gente no tenga noción de ello, el Aprendizaje Automático está en numerosos aspectos de nuestras vidas. Ejemplo de ello puede ser la detección automática de *spam* en el correo electrónico o la recomendación de productos en base a los gustos de una persona en una página web. Aunque esto pueda parecer sencillo, pues se ve muy a menudo, en realidad no lo es tanto: hay todo un proceso, donde se involucran complejos algoritmos y años de desarrollo, para lograrlo.

Este proceso es aún más complicado cuando se trata con el lenguaje natural: para un ser humano puede resultar fácil leer un texto y decir una serie de palabras que definan su contenido. Sin embargo hay todo un campo de estudio para que dicha generación se realice de forma automática. De hecho, será habitual que se establezcan varias palabras, como las referidas a este trabajo: clasificación multietiqueta, aprendizaje automático y *text mining*. Esta apreciación, que parece trivial, ha supuesto un nuevo área de desarrollo, donde se ha tenido en cuenta el cariz multietiqueta que tienen numerosos datos. Este trabajo, precisamente, se centra en ello: se profundizará sobre el tratamiento automático de textos que pueden contar con varias etiquetas de contenido.

1.2. Motivación

El principal motivo por el cual se quiere desarrollar este trabajo consiste en investigar sobre la clasificación multietiqueta de textos. Además, pasar del mero conocimiento al desarrollo, es decir, de estudiar qué técnicas existen para resolver ese problema a ponerlas en práctica, supone un aliciente.

Por otro lado, más allá del simple interés por conocer más a fondo estas técnicas, se cree puede servir como punto de partida de cualquier técnica que se base en este tipo de clasificación, sea cual sea su ámbito. Aunque el tema de este trabajo, como se puede deducir del título, tenga como base los textos científicos, lo aprendido en él sobre el tratamiento de lenguaje natural, donde se puede extraer información de ellos de forma automática, puede tener muchos más usos, por ejemplo:

1. Reconocimiento de patrones en imágenes, teniendo en cuenta que puede haber varios en una misma imagen, por ejemplo, una foto en la que se puede visualizar el mar y la montaña. (Boutell et al., 2004), (Li et al., 2011).

2. Aplicaciones médicas. Por ejemplo, un modelo de diagnóstico donde se puede detectar más de un síndrome a la vez (Liu et al., 2012) o el análisis del genoma humano (Clare y King, 2001).

3. Otras aplicaciones: relación entre las emociones y la música (Li y Ogihara, 2003), predicción de resolución de disputas en relaciones público-privadas (Chou, 2012)

1.3. Objetivos

El objetivo principal de este trabajo se corresponde con diseñar, implementar y evaluar un software que genere automáticamente las palabras clave necesarias para etiquetar un artículo científico a partir de su resumen (*abstract*). Para ello, se utilizarán técnicas de Aprendizaje Automático, concretamente, técnicas de clasificación multietiqueta.

Tomando como base dicho objetivo principal, se han descrito una serie de objetivos secundarios que se pretenden alcanzar a la hora de desarrollar este proyecto:

1. Investigar las peculiaridades de la clasificación multietiqueta: tanto de los métodos para clasificar, como de las métricas de evaluación.

2. Conocer las diferentes tecnologías existentes para desarrollar este tipo de clasificación.

3. Familiarizarse con el trato del lenguaje natural en tareas de Aprendizaje Automático.

4. Desarrollar un proyecto de principio a fin, a partir de una idea propia, comprendiendo las fases involucradas hasta lograrlo.

1.4. Estructura

El presente documento está dividido en las siguientes secciones, de las cuales se explicará brevemente su contenido:

Capítulo 1: Introducción. Mediante este capítulo se busca establecer una preámbulo donde se explique, en líneas generales, por qué se ha decidido hacer este proyecto y qué se pretende hacer.

Capítulo 2: Estado de la cuestión. Para poder llegar a desarrollar el software mencionado, es necesario primero entender cómo se puede hacer. Mediante el análisis presente en este capítulo se trata de adquirir las competencias necesarias para poder desarrollar una clasificación multietiqueta de textos. Tanto a nivel teórico, estudiando las técnicas existentes en la literatura, como a nivel práctico, mediante las herramientas existentes para llevarlo a cabo.

Capítulo 3: Descripción del trabajo realizado. En este capítulo se exponen las fases por las cuales ha pasado el proyecto de acuerdo a una determinada metodología: desde el análisis, pasando por el diseño, y su posterior implementación.

Capítulo 4: Evaluación. Una vez se ha desarrollado el sistema, es momento de ponerlo en práctica. En este capítulo se mencionarán las pruebas realizadas para corroborar el correcto funcionamiento del sistema, así como las pruebas de comportamiento.

Capítulo 5: Planificación. 5 En esta sección se ofrecerá el plan temporal llevado a cabo para desarrollar el presente trabajo. Mediante él, se podrá observar qué tareas principales existen, la relación entre ellas y la temporización.

Capítulo 6: Presupuesto. A fin de situar este proyecto en el contexto socioeconómico actual, se ha incluido un presupuesto detallado, donde se recogen los costes en los que ha incurrido en su realización.

Capítulo 7: Conclusiones. Para finalizar este TFG, en este capítulo se puede encontrar aquellas reflexiones finales acerca de él: tanto las limitaciones del sistema desarrollado como posibles líneas futuras de desarrollo.

Capítulo 8: Bibliografía. Durante la realización de este proyecto, se ha consultado numerosas fuentes, las cuales han sido debidamente mencionadas en el texto y recogidas en esta sección.

Apéndice A: Resumen de contenidos en inglés. Tal y como figura en la normativa del Grado de Ingeniería Informática del Plan 2011, se debe incluir en el TFG una parte que demuestre la competencia en inglés. Para ello, se ha incluido esta sección, donde se recoge una introducción, un resumen de las partes más relevantes del sistema y las conclusiones finales.

Capítulo 2

Estado de la cuestión

2.1. Aprendizaje Automático

En primer lugar, se debe mencionar que el Aprendizaje Automático (AA) conforma una rama de la Inteligencia Artificial (IA). Por tanto, para definir qué es el AA, es necesario hacer un breve repaso sobre qué es la IA.

Las definiciones de IA se suelen poder organizarse en cuatro categorías (Russell y Norving, 1995), las cuales pueden proporcionar información acerca de sus diferentes interpretaciones:

1. Sistemas que piensan como humanos.
2. Sistemas que actúan como humanos.
3. Sistemas que piensan racionalmente
4. Sistemas que actúan racionalmente.

De cualquier forma, la IA trata de dotar a un sistema la capacidad de discurrir. El nacimiento de la Inteligencia Artificial podría encontrar sus indicios mucho antes, pero fue en 1950 cuando Alan Turing lanzó la siguiente pregunta: ¿Pueden las máquinas pensar? (Turing, 1950), a fin de proponer *"The imitation game"*, una analogía por la cual pretendía plantear la posibilidad de que un humano no pudiera reconocer si estaba tratando con una máquina o con otro humano. De esta manera, se inició el planteamiento de la Inteligencia Artificial, término que fue acuñado por

John McCarthy y que se formalizó en el verano de 1956 durante una Conferencia en la Universidad *Darmouth College* (Estados Unidos) (McCarthy et al.), evento clave para su impulso.

Ahora bien, el AA trata de ir más allá, pues se fundamenta en el intento de que el sistema no solo discurra, sino que sea capaz de *aprender*. De esta manera, debería ser capaz de, a partir de un conjunto de ejemplos que le provean de información de la cuestión en concreto, resolver dicho problema de forma automática, es decir, sin la intervención directa de un ser racional. Incluso, este sistema debería ser capaz de mejorar su comportamiento tomando como base la experiencia adquirida. Dicho con otras palabras, y tomándolas del libro *Aprendizaje Automático* (Millán et al., 2006), *"El objetivo del AA consiste en dotar a los computadores la capacidad de adquirir conocimientos automáticamente para poder resolver nuevos problemas o mejorar su comportamiento a partir de la experiencia"*. Además, se defiende la idea de que, en un futuro, casi la totalidad de las máquinas necesitarán el componente de adaptación o de aprendizaje mencionado, por lo que la necesidad futura del AA será clave.

2.1.1. Tareas representativas del AA

Existen tres tareas representativas de los problemas a los que se enfrenta el AA: la clasificación, la agrupación y la caracterización. Tomando las definiciones que se pueden encontrar en (Millán et al., 2006).

1. Clasificación: búsqueda de una relación de correspondencia entre las observaciones y las clases a través de un conjunto de reglas de inferencia. Con otras palabras, consiste en, partiendo de un conjunto conocido de clases, asignar a nuevos casos una de las categorías existentes.

2. Agrupación o *clustering*: partiendo de un conjunto de ejemplos de los cuales no se conoce su clase, encontrar las clases más probables en las que se pueden agrupar dichos ejemplos.

3. Caracterización: obtención de la descripción simbólica de un conjunto a partir de sus componentes. De esta manera, se pueden observar los rasgos que más caracterizan la pertenencia de un ejemplo a un conjunto.

Existe también otra tarea donde se utiliza el AA: el aprendizaje por refuerzo. En él, se trata de otorgar a un agente la capacidad de decidir qué hacer ante una situación indeterminada, tratando de maximizar o minimizar un objetivo. Además, hay dos características principales en este tipo de aprendizaje (Sutton y Barto, 1998): al no especificar al agente qué decisiones tomar en cada momento, éste deberá decidir qué acciones le otorgan una mayor recompensa, basándose en dicho objetivo, a través del método de prueba y error. La segunda característica se basa en que es posible que sus acciones no tengan un efecto inmediato en la siguiente situación. Por ejemplo, al tratar de salir de un laberinto, no se sabe hasta que se ha alcanzado la salida que las decisiones previas fueron las adecuadas.

Clasificación

Como se ha podido deducir, la tarea que atañe a este trabajo se correspondería con aquellas de clasificación, pues se trata de, a partir de un conjunto de textos, determinar qué etiquetas definen su contenido.

A la hora de llevar a cabo una tarea de clasificación es necesario contar, en primer lugar, con un conjunto de ejemplos de los cuales se conozca lo suficiente para poder extraer una serie de atributos, así como el valor de una clase que los represente. A continuación, se utilizará un algoritmo de clasificación, cuya misión será generalizar los atributos que describen cada clase, para así poder inferir la clase de un nuevo ejemplo. En las tareas de clasificación habituales, es normal aplicar algún tipo de conocimiento para determinar los atributos. Por ejemplo, si se deseara generar un clasificador que determinara si un paciente padece una enfermedad o no, se podría contar con la opinión de un experto que explicara cuales son los síntomas propios o los aspectos a tener en cuenta. Sin embargo, en la tarea que se trata de resolver en este trabajo, se detectan varias diferencias que es preciso comentar.

En primer lugar, la selección de atributos es diferente al tratarse de un sistema cuya entrada son textos, y de los cuales se quiere averiguar su contenido. Debido a ello, no se puede determinar de antemano qué atributos se van a utilizar para categorizar cada texto, sino que se deberá aplicar algún tipo de mecanismo para ello. Por tanto, los atributos dependerán enteramente de qué textos se elijan para entrenar el clasificador. Con otras palabras, no es útil utilizar información exógena,

es decir, información provista por una fuente externa, sino se dependerá enteramente de la información endógena que se logre extraer, es decir, de la información que reside propiamente en el texto. (Sebastiani, 2002). Por su parte, se profundizará mas en esta cuestión en la Sección 2.2

En segundo lugar, tradicionalmente se ha asignado una sola clase a cada instancia. De esta forma, un clasificador podría diferenciar entre varias clases, pero no podría determinar la pertenencia de un ejemplo a más de una. Esto significaría, por ejemplo, que un clasificador no podría determinar si un texto trata sobre biología y matemáticas a la vez. Sin embargo, se han desarrollado técnicas cuyo objetivo es poder manejar varias clases a la vez, lo que se denomina multietiqueta. Esto, se explicará más a fondo en la Sección 2.3.

2.1.2. Aplicaciones del Aprendizaje Automático

Para citar algunas de las aplicaciones del AA, pues son muy numerosas, se ha utilizado una división encontrada en el libro *Aprendizaje Automático* (Millán et al., 2006), según la cuál se dividen las motivaciones por las que se considera útil el estudio del AA, pudiéndose obtener de ella una proyección hacia las diferentes aplicaciones:

1. Técnicas: construcción de software de forma automática.

2. Comerciales: personalización de servicios y productos para el ciudadano.

3. Científicas: estudio del aprendizaje humano y análisis automático de grandes cantidades de datos científicos.

Debido a la naturaleza de este trabajo, se prestará especial atención en la última categoría, más concretamente en el *Big Data Mining*, cuyo significado se concretará a continuación.

Se podría definir *Big Data* como un término usado para identificar set de datos que tienen un gran tamaño y complejidad, por lo que no pueden ser tratados por las herramientas comunes de minería de datos. De ahí surge, por tanto, el concepto de *Big Data mining*, que se correspondería como la capacidad de extraer información útil de dichos grandes set de datos que debido a su volumen, variedad y velocidad, antes era imposible de hacer (Fan y Bifet, 2012). Existen 5 V's que se relacionan

con este último término, representando los factores que han de tenerse en cuenta al tratar con estos datos:

1. Volumen: el tamaño de los datos se incrementa a cada segundo.
2. Variedad: los tipos de datos son muy diferentes, como texto, audio, video, etc.
3. Velocidad: los datos se generan continuamente, pero lo realmente interesante es poder obtener información útil en tiempo real.
4. Variabilidad: existen cambios en la estructura de los datos y cómo los usuarios interpretan dichos datos.
5. Valor: existe un valor de negocio asociado a ello, pues se pueden tomar decisiones basándose en las conclusiones obtenidas.

2.2. Análisis de textos

Como se ha mencionado, una de las principales diferencias en el clasificador que se trata de desarrollar se fundamenta en la entrada del mismo. De esta manera, no se parte de unos atributos ya definidos con anterioridad, sino que se irán descubriendo a partir del tratamiento de los diferentes resúmenes escogidos. Se debe, por tanto, realizar un análisis de dichos textos a fin de determinar qué palabras podrían servir para definir su contenido. Esta tarea se puede encuadrar en la Minería de textos (*Text Mining*), donde se trata de alcanzar algún tipo de conocimiento útil a través de una colección de documentos. Su similitud con la Minería de datos (*Data Mining*) es evidente, pues parten de un mismo objetivo, con la salvedad de que uno trata con textos y, el otro, con datos. No obstante, el proceso para lograrlo, en líneas generales, puede ser el mismo. Por ello, se va a presentar, en primer lugar, el proceso KDD (*Knowledge Discovery in Databases*), para luego pasar a mencionar las peculiaridades de ese método cuando se trata de *text mining*.

2.2.1. Proceso KDD

El KDD podría definirse como el proceso organizado de identificar patrones válidos, nuevos, útiles y comprensibles de grandes y complejos conjuntos de datos. La

Minería de Datos aunque parece recoger un concepto parecido, supone en realidad el núcleo del proceso KDD, pues implica la inferencia de algoritmos para explorar los datos, desarrollando un modelo y descubriendo patrones desconocidos. Dicho modelo es utilizado para entender fenómenos presentes en los datos, analizarlos y predecirlos (Grigorios Tsoumakas y Vlahavas, 2005). De esta manera, el proceso KDD estaría formado por nueve pasos iterativos e interactivos, los cuales son presentados a continuación.

PASO 1. Comprender el dominio de aplicación

Mediante este paso preparatorio, se trata de generar la comprensión suficiente para saber qué decisiones se deben tomar en el resto de pasos del proceso. De esta manera, se deberán fijar los objetivos que se buscan lograr mediante el proyecto en particular.

PASO 2. Seleccionar y crear un conjunto de datos sobre el que se busca generar algún tipo de conocimiento

Al fijar en el paso anterior los objetivos, indirectamente se está guiando la forma y el lugar de donde obtener los datos. Este paso involucra todo el proceso de búsqueda de datos, detectando cuales están disponibles y de qué forma se podrían obtener, así como la fijación de los atributos que se deben recoger. Como se explicó, el proceso KDD es iterativo, por lo que se podrá volver sobre este punto para recolectar los datos de nuevo si fuera necesario.

PASO 3. Preprocesamiento y limpieza

En este paso, se busca mejorar la fiabilidad de los datos mediante la aplicación de diferentes técnicas de limpieza, como por ejemplo eliminar datos erróneos o tratar con atributos de los cuales no se disponga el valor. Esta fase, dependiendo de la naturaleza del proyecto, puede ocupar muy poco tiempo, o presentarse como una de las mayores tareas a desarrollar.

PASO 4. Transformación de los datos

Mediante este paso, se trata de preparar y desarrollar un buen conjunto de datos para las tareas de minería posteriores. De esta forma, se aplican métodos relacionados con la reducción de la dimensión de los datos, o la transformación de ciertos atributos, como puede ser de un valor nominal a numérico, lo que puede provocar una reducción considerable del tiempo de ejecución del algoritmo. Esta fase, aunque puede ser muy específica de cada proyecto en particular, es vital para el éxito de todo el proceso KDD.

PASO 5. Elección de la tarea minería de datos

Una vez se han completado los cuatro pasos anteriores, es momento de pasar a las tareas propias de la minería de datos. En primer lugar, se debe fijar qué se quiere hacer mediante dicha minería, aunque esto ya debería haberse encaminado cuando se determinaron los objetivos. Por ejemplo, en este paso se puede determinar si se debe llevar a cabo una clasificación o una regresión. También es momento de decidir si se busca predecir (tratar de saber qué va a ocurrir con datos no clasificados anteriormente) o describir (analizar qué ocurre con los datos).

PASO 6. Elección del algoritmo de minería de datos

Una vez se ha decidido la estrategía, es momento de decidir las técnicas necesarias para lograrla. En este paso, se trata de valorar las diferentes posibilidades que proveen los algoritmos existentes, a fin de detectar cuál se adapta mejor a las circunstancias del proyecto. También es momento de fijar bajo qué circunstancias se considera más apropiada su utilización, como puede ser la aplicación de validación cruzada o la evaluación con un conjunto de test separado.

PASO 7. Ejecución del algoritmo de minería de datos

Habiendo elegido ya el algoritmo, es el turno de su aplicación. En este paso, se trata de probar y ajustar los diferentes parámetros del algoritmo, a fin de que se prueben las diferentes variaciones de un mismo algoritmo, obtiéndose el que mejor se adapta a los datos.

PASO 8. Evaluación e interpretación

En este paso, se deben evaluar e interpretar los patrones obtenidos respecto a los objetivos marcados en el paso 1. En este punto, se puede valorar el impacto que han tenido las decisiones tomadas en la parte de preprocesamiento, por ejemplo el añadir algún atributo y repetir todo el proceso desde ese paso. Se busca, asimismo, centrarse en la comprensibilidad y utilidad del modelo generado.

PASO 9. Utilización del conocimiento descubierto

Una vez se ha alcanzado este punto, se considera que se puede incorporar el conocimiento adquirido, mediante el uso del modelo desarrollado.

2.2.2. Tratamiento de textos

Existen dos problemas principales a la hora de tratar con datos, en este caso documentos: el primero, es que la mayoría de los datos están en un formato que una máquina no puede usar. Por lo tanto, se debe aplicar algún tipo de transformación para lograrlo. El segundo problema mencionado reside en cómo representar y almacenar la información extraída (Rodrigues et al., 2015). Estos dos problemas son mencionados en el proceso KDD, aunque de forma indirecta: en el paso 2 y 3 se hace referencia a la selección de atributos, y al preprocesamiento y limpieza, respectivamente.

Cuando se trata con textos y lenguaje natural, se debe prestar especial atención a esta parte pues, potencialmente, todas las palabras podrían ser un atributo. Sin embargo, esto presentaría un problema grave en términos de complejidad. Por ello, antes de poder generar los atributos, es necesario realizar algún tipo de transformación a los textos de entrada.

Por un lado, como ya se mencionó, no se pueden elegir previamente los atributos debido a la naturaleza de los datos. El método para determinar los atributos se explicará en la Sección 2.2.3.

Por otro lado, el preprocesamiento y limpieza difieren del tratamiento habitual de los datos (Feldman y Sanger, 2007). Normalmente, estas tareas se basan en detectar ejemplos erróneos o campos vacíos, pero en este caso se deben transformar

los textos. Esto se puede realizar mediante técnicas de preprocesamiento en *text mining*, más concretamente en técnicas relacionadas con el procesamiento del lenguaje natural (*natural language processing* o NLP). Dichas técnicas, tratan de procesar los textos utilizando el conocimiento general acerca del lenguaje natural. No obstante, las técnicas que se deben aplicar en la tarea que atañe a este trabajo son básicas, pues normalmente se determinan como tareas principales de NLP las relacionadas con identificar aspectos morfosintácticos de las frases, información gramatical o de contexto. Por ejemplo, en este trabajo sería interesante aplicar técnicas orientadas a transformar las palabras de los textos a fin de que se unifiquen aquellas que representan un mismo concepto, además de retirar aquellas palabras que no aportan nada a la tarea de clasificación. De esta manera, se trata de no diferenciar entre palabras como: robots, Robots, robótica, y de eliminar palabras que no recogen contenido: la, aquel, ello.

2.2.3. Análisis de frecuencia de palabras

Como ya se ha mencionado, uno de los problemas principales se deriva de la selección de los atributos necesarios para la tarea de aprendizaje. Antes de presentar las diferentes técnicas para ello, se considera útil presentar la notación que se va a utilizar. La aplicación de las técnicas presentadas se realiza sobre un conjunto de documentos, ($D = \{d_1, d_2, ..., d_n\}$), formados cada uno por un conjunto de términos o palabras ($D_i = (t_1, t_2, ..., t_n)$).

La aproximación más básica consiste en medir el número de ocurrencias de cada palabra, es decir, su frecuencia (*Term Frequency*), lo que se denotaría como $tf_{t,d}$, donde la t se refiere al término o palabra, y la d al documento donde aparece. Si se dispusiera los siguientes documentos:

Documento 1. María es muy ágil.

Documento 2. Pedro, aunque también es ágil, es más rápido.

Se generaría una lista con las palabras que aparecen en alguno de los dos textos, para cada una se podría calcular la frecuencia, como se observa en la Tabla 2.1.

María	$tf_{1,1} = 1$	$tf_{1,2} = 0$
es	$tf_{2,1} = 1$	$tf_{2,2} = 2$
muy	$tf_{3,1} = 1$	$tf_{3,2} = 0$
ágil	$tf_{4,1} = 1$	$tf_{4,2} = 1$
Pedro	$tf_{5,1} = 0$	$tf_{5,2} = 1$
aunque	$tf_{6,1} = 0$	$tf_{6,2} = 1$
también	$tf_{7,1} = 0$	$tf_{7,2} = 1$
más	$tf_{8,1} = 0$	$tf_{8,2} = 1$
rápido	$tf_{9,1} = 0$	$tf_{9,2} = 1$

Tabla 2.1: Ejemplo *Term Frequency*

Si se expresara en forma de vector de información, quedaría de la siguiente manera:

Documento 1. $[1, 1, 1, 1, 0, 0, 0, 0, 0]$

Documento 2. $[0, 2, 0, 1, 1, 1, 1, 1, 1]$

La forma presentada de cálculo de frecuencia podría variar, pues aquí se ha recogido la frecuencia bruta, pero es común utilizar la frecuencia normalizada, para así evitar la primacía de los textos largos sobre los cortos. Ésta se calcularía como el cociente entre la frecuencia bruta de un término entre el número de términos del documento, como se muestra en la Ecuación 2.1.

$$ntf_{t,d} = \left(\frac{tf_{t,d}}{tf_{max}(d)} \right) \tag{2.1}$$

Sin embargo, la utilización de la frecuencia de los términos adolece de un problema: asume que todas las palabras son igual de relevantes. Por ejemplo, si se dispusiera de una base de datos de artículos sobre Inteligencia Artificial, y se quisiera saber qué documento o documentos se refieren al término *aprendizaje supervisado*, se encontraría un problema: el término aprendizaje aparecería en numerosas ocasiones, por lo que podría dificultar la determinación de la relevancia de los documentos que sí trataran sobre aprendizaje supervisado.

Para solucionar esto, se puede introducir un factor de atenuación, cuyo objetivo

es reducir el efecto de los términos que aparecen demasiado frecuentemente en una colección. Dicho factor se denomina *inverse document frequency* o *idf* y se calcula como se muestra en la Ecuación 2.2.

$$idf_t = \log\left(\frac{N}{df_t}\right) \tag{2.2}$$

Donde:

- N: número total de documentos.
- df_t: *Document Frequency* o frecuencia del documento. Se refiere al número de documentos que contienen el término t.

De esta manera, el valor de *idf* de un término poco común será alto, mientras que será bajo si el término es frecuente.

Tomando el ejemplo anterior, df_t y idf_t tendrían los valores mostrados en la Tabla 2.2.

`María`	$df_1 = 1$	$idf_1 = \frac{2}{1} = 2$
`es`	$df_2 = 2$	$idf_2 = \frac{2}{2} = 1$
`muy`	$df_3 = 1$	$idf_3 = \frac{2}{1} = 2$
`ágil`	$df_4 = 2$	$idf_4 = \frac{2}{2} = 1$
`Pedro`	$df_5 = 1$	$idf_5 = \frac{2}{1} = 2$
`aunque`	$df_6 = 1$	$idf_6 = \frac{2}{1} = 2$
`también`	$df_7 = 1$	$idf_7 = \frac{2}{1} = 2$
`más`	$df_8 = 1$	$idf_8 = \frac{2}{1} = 2$
`rápido`	$df_9 = 1$	$idf_9 = \frac{2}{1} = 2$

Tabla 2.2: Ejemplo *Document Frequency* y *Inverse Document Frequency*

Ahora, se podrían combinar las definiciones de la frecuencia del término y la frecuencia inversa del documento, que se correspondería con dicho factor atenuante, para calcular el peso otorgado a cada término de cada documento, lo que se denomina TF-IDF (*term frequency - inverse document frequency*). Se puede observar en la Ecuación 2.3.

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t \tag{2.3}$$

De esta manera, el valor de $TF-IDF_{t,d}$ asignado a un término t en un documento d será:

- Alto cuando t aparece muchas veces, pero en un número reducido de documentos.
- Bajo cuando un término aparece pocas veces en un documento, o aparece en muchos documentos.
- Bajo cuando un término ocurre en prácticamente todos los documentos.

Para la elaboración de esta sección se ha repasado lo expuesto en: (Salton y McGill, 1986, pp. 117-119).

2.3. Clasificación multietiqueta

A la hora de resolver un problema de aprendizaje supervisado multietiqueta, existen dos tareas principales: la clasificación multietiqueta (MLC) y el *label ranking*. En el primero de ellos, se trata de dividir la decisión en dos conjuntos: las etiquetas predichas y las no predichas. En el segundo, sin embargo, la decisión se fundamenta en una jerarquía donde se recogen todas las clases, de más relevante a menos para una instancia en particular (Grigorios Tsoumakas y Vlahavas, 2005, 667-668). En este trabajo, donde se trata de establecer una serie de etiquetas para cada resumen, se ha decidido estudiar las técnicas correspondientes al primer grupo: MLC.

Como ya se ha explicado, la característica principal de la clasificación multietiqueta, tomando las siglas en inglés MLC correspondientes a *multilabel classification*, consiste en que los ejemplos están asociados a un conjunto de etiquetas, frente a la clasificación tradicional (*single-label classification* o SLC) donde los ejemplos solo están asociados a una etiqueta perteneciente a un conjunto de etiquetas (Nasierding y Kouzani, 2012b).

De esta manera, se podría dar una descripción formal de un problema cualquiera multietiqueta, donde $L = \{\lambda_j : j = 1...q\}$ denota un conjunto finito de etiquetas referente a una tarea de clasificación multietiqueta, y $D = \{(x_i, Y_i), i = 1..m\}$ denota un

conjunto de ejemplos de entrenamiento multietiqueta, siendo x_i el vector de atributos; y donde $Y_i \subseteq L$ hace referencia al conjunto de etiquetas del iésimo ejemplo. Frente a él, un conjunto de entrenamiento SLC se expresaría como $D = \{(x_i, Y), i = 1..m\}$

Para resolver un problema de MLC existen dos enfoques principales: la transformación del problema y la adaptación del algoritmos (Tsoumakas y Katakis, 2007). En el primero de ellos, se transforma la tarea de aprendizaje en una o más tareas de SLC, mientras que en el segundo se desarrolla un algoritmo que trate explícitamente con problemas de MLC. A continuación, se va a realizar un repaso de aquellas técnicas que se encuentran en cada uno de los dos enfoques, pues su estudio es necesario para determinar la mejor solución a la hora de desarrollar el MLC de este trabajo.

2.3.1. Transformación de problemas

Existen numerosas técnicas cuyo objetivo es transformar un problema de MLC de tal forma que pueda ser solucionado por los métodos tradicionales de SLC, algunas de las cuales se van a presentar a continuación. Estas son: *Support Vector Machine*, *Binary Relevance*, BR+,*Classifier Chain*, *Label Powerset* y RAkEL.

Support Vector Machine

Uno de los primeros acercamientos para resolver problemas multietiqueta se basa en la Máquina de Soporte Vectorial (*Support Vector Machine* o SVM). La SVM fue presentada en 1995 como una nueva forma de resolver problemas de clasificación binaria (Cortes y Vapnik, 1995), donde se trata de separar ambas clases mediante separadores lineales o hiperplanos. Se puede encontrar más información en: (Cortes y Vapnik, 1995) (Suárez, 2014).

Ante la necesidad de discriminar entre más de dos clases, la SVM fue adaptada para la clasificación multietiqueta, teniendo en cuenta dos perspectivas: (Hsu y Lin, 2002):

1. *One-against-all*: este método se basa en la construcción de k modelos SVM, uno por cada clase.

2. *One-against-one*: en este caso, se construyen $\frac{k(k-1)}{2}$ clasificadores SVM, donde cada uno es entrenado con los datos de dos clases.

Nótese que en dichos enfoques, se podría utilizar cualquier modelo binario, no solo una máquina SVM, aunque es habitual su utilización. De hecho, el modelo *Binary Relevance*, presentado a continuación, es una aplicación del enfoque *one-against-all*. Asimismo, *Label PowerSet* podría ser una aplicación del segundo enfoque.

Binary Relevance

Dado un conjunto de etiquetas $L = \{l_1, l_2, ..., l_n\}$, el método denominado como *binary relevance* (BR o relevancia binaria en español) (Cherman et al., 2011) consiste en descomponer un problema de MLC en n problemas SLC, uno por cada etiqueta del conjunto E. De esta forma, se deberá entrenar n clasificadores binarios a partir de un conjunto de entrenamiento común, el cual solo variará de un clasificador a otro en el valor del atributo referente a la clase, pues deberá ser q en aquellas instancias referentes a la etiqueta en cuestion, y $\neg q$ en los restantes. A la hora de determinar el conjunto de etiquetas de una nueva instancia, ésta deberá pasar por cada uno de los n clasificadores, siendo el resultado la agregación de la salida de ellos. Su implementación, por tanto, resulta sencilla, lo que constituye una de las ventajas de su utilización.

Una de las desventajas de este método, como puede deducirse, está relacionada con el número de etiquetas de las cuales se disponga: a medida que n aumente, el número de clasificadores aumentará en la misma proporción. Por ello, resulta solo apropiado para problemas que tenga un conjunto pequeño de L. No obstante, se debe apreciar una gran limitación del BR, pues no tiene en cuenta la relación que pueda existir entre etiquetas, ya que al transformar el conjunto de entrenamiento, esta información se pierde. Existe una variación de este método llamada BR+ (Cherman et al., 2011), la cual trata de resolver este aspecto. Para ello, incorpora como atributos de cada ejemplo el valor del resto de las etiquetas, de tal forma que a cada instancia se le debería incluir $(n - 1)$ atributos. De esta manera, se estaría incluyendo la posibilidad de que el clasificador tuviera en cuenta, si es que lo hubiera, una relación entre las etiquetas. El problema resulta al intentar clasificar un nuevo ejemplo, pues habría $(n - 1)$ atributos de los cuales se desconocería el valor.

Classifier Chain

El método cadena de clasificadores (*Classifier Chain* o CC) (Read et al., 2011) genera n clasificadores binarios, uno por cada etiqueta, de igual forma que en el BR. Sin embargo, cada clasificador está conectado con el anterior, de tal forma que añade como atributo la predicción del clasificador anterior. De esta manera, se comienza con el primer clasificador binario para la etiqueta e_1, que generará, para cada ejemplo, su clasificación. Se añadiría, por tanto, un atributo más y, por consiguiente, un valor más a cada ejemplo, que indicaría la predicción del primer clasificador. Así, el segundo clasificador, diseñado para diferenciar la segunda etiqueta, tendrá como entrada los mismos atributos que el primer clasificador, pero con la predicción para la primera etiqueta. De esta forma, se tendría en cuenta la relación entre etiquetas.

De la utilización de este método se derivan dos problemas: el primero de ellos, se basa en la elección del orden de las etiquetas para construir la cadena de clasificadores, pues esto tendrá un impacto en la información que el clasificador puede sacar de la relación entre etiquetas. Se debería, por tanto, probar diferentes combinaciones generadas de forma aleatoria, evaluando el rendimiento en cada una de ellas, lo que se puede hacer generando un conjunto de cadenas de clasificadores (*Ensemble Classifier Chain o ECC*). El segundo problema, por su parte, trata del aumento de la complejidad al añadir n atributos.

Label PowerSet

El tercer método se llama *Label Powerset* (LP) (Cherman et al., 2011) y éste aplica una transformación al conjunto de etiquetas con valor positivo que identifica cada ejemplo del conjunto de datos. De esta manera, si un ejemplo tuviera las etiquetas l_1 y l_3, éstas serían recodifcadas, pasando a ser una clase nueva denominada, por ejemplo, $y_{1,3}$. En nuestro caso particular, si un texto tratara de biología y matemáticas, pasaría a tener una sola etiqueta, denominada: biología_matemáticas. A la hora de identificar un nuevo ejemplo, el resultado del clasificador sería un conjunto de etiquetas, en lugar de una sola.

La desventaja principal de la utilización de este método se deriva de la codificación de las variables, pues está estrechamente ligado a cuántas combinaciones haya entre ellas. De hecho, en el peor de los casos en que se dieran todas las combinaciones

posibles de las etiquetas del conjunto L, se tendrían que recodificar 2^n etiquetas. Otro problema que puede surgir al utilizar *Label Powerset* resulta del número de ejemplos de cada clase, pues seguramente estará desbalanceado, habiendo pocos ejemplos de alguna de ellas. Aunque en este caso sí se guarda la relación entre las etiquetas, de hecho, se está forzando a que haya una relación al codificarlo de esta manera, su utilización está muy penalizada por la complejidad exponencial que presenta.

RAkEL

Debido a dicha complejidad, se propuso un nuevo algoritmo, denominado RAkEL (*RAndom k-labELsets* (Tsoumakas y Vlahavas, 2007)), que funciona de la siguiente forma: se realiza una codificación de las variables de la forma anteriormente descrita para generar un clasificador LP. Sin embargo, en lugar de utilizar todas las etiquetas codificadas para entrenar un único clasificador, se escogen k etiquetas, y se entrena un clasificador. Esta operación se repite las veces que se considere necesario, obteniéndose m clasificadores binarios. A la hora de clasificar una nueva instancia, ésta deberá pasar por los m clasificadores construidos, cada uno de los cuales generará una serie de clases que considera le corresponden. Para obtener su clase final, se determinará un umbral, que representará el número mínimo de clasificadores en los que debe de figurar como positiva una clase para considerarla como clase final de la instancia. De esta manera, se estaría solucionando el problema de la complejidad y se estaría manteniendo la ventaja de tener en cuenta las relaciones entre etiquetas. La principal dificultad asociada a la aplicación de este algoritmo se deriva de una correcta elección de tres parámetros: el número de etiquetas (k), el número de clasificadores a entrenar (m) y el umbral.

2.3.2. Adaptación de algoritmos

Son numerosos los algoritmos tradicionales que han sido adaptados para tratar con multietiqueta. Se considera que comentar qué cambios se han realizado en dichos algoritmos es profundizar demasiado para este trabajo. No obstante, algunos de ellos son: C4.5, *AdaBoost*, BP-MLL (*Back-propagation algorithm for multilabel learning*), MPP (*multi-class multi-label perceptron).* Se puede encontrar más información en (Tsoumakas y Katakis, 2007) y (Grigorios Tsoumakas y Vlahavas, 2005).

2.4. Evaluar un clasificador

A la hora de evaluar el desempeño de un clasificador, es necesario establecer una técnica para decidir sobre qué datos se van a aplicar las métricas de evaluación. Una vez se han explicado dichas técnicas, se pasará a analizar las diferentes métricas existentes para evaluar un clasificador multietiqueta, donde algunas de ellas son transformaciones a partir de las métricas básicas, por lo que también es necesario su explicación.

De esta manera, se explicará en primer lugar las diferentes transformaciones que puede sufrir el conjunto de datos para la evaluación; en segundo lugar, las métricas básicas, y, finalmente, los tres tipos utilizados para la evaluación multietiqueta: métricas basadas en ejemplos, métricas basadas en etiquetas y métricas basadas en *ranking* (Nasierding y Kouzani, 2012a) (Sorower, 2010).

2.4.1. Selección del conjunto de test

Una vez se dispone de los datos, se debe plantear la cuestión de qué parte será utilizada para entrenar, y qué parte para evaluar el sistema. Se van a citar cuatro formas de realizarlo: *holdout*, *random subsampling*, validación cruzada y *leave-one-out*.

Holdout

Mediante el método *holdout* se divide el conjunto de datos en dos partes: entrenamiento y test, representándo cada uno de ellos un porcentaje variable del conjunto inicial. Es común otorgar un porcentaje mayor al conjunto de entrenamiento. De esta manera, se entrenaría el modelo con el conjunto de entrenamiento, y se utilizaría para la evaluación el de test, del cual se tendrían dos clases: la real y la predicha por el clasificador.

El problema principal de este método es que no otorga la suficiente veracidad, pues el rendimiento del clasificador puede venir determinado porque casualmente se haya escogido una división buena. Además, los datos que se utilicen para el conjunto de test no se estarían utilizando para entrenar, por lo que se desaprovecharían datos de entrenamiento.

Random subsampling

Random subsampling surge como una forma de solventar el problema del método *holdout*. Este método consiste en dividir el conjunto de datos en k divisiones, para luego escoger de forma aleatoria algunas de esas partes para entrenar, y otras para evaluar el desempeño. Además, esta prueba se puede repetir varias veces, obteniéndose la media de todas ellas.

Validación cruzada

En el caso de la validación cruzada de k-iteraciones (k-*folds*), se divide el conjunto de datos en k particiones, utilizando $k-1$ para el entrenamiento y la una restante para el test. Esta partición se repetirá k veces, siendo cada vez un conjunto diferente el que se utilice como test, asegurándose de que todas las particiones formen parte del conjunto de entrenamiento y del test. La precisión final será calculada como la media, a partir del valor obtenido en cada una de las iteraciones. Se ha incluido un ejemplo en la Tabla 2.3 para facilitar la explicación.

	particiones				
Prueba 1	E	E	E	E	**T**
Prueba 2	E	E	E	**T**	E
Prueba 3	E	E	**T**	E	E
Prueba 4	E	**T**	E	E	E
Prueba 5	**T**	E	E	E	E

Tabla 2.3: Ejemplo validación cruzada con $k = 5$.

Cabe señalar que el proceso anteriormente mencionado solo serviría para valorar el desempeño del clasificador. De esta forma, el proceso de entrenamiento se llevaría a cabo con todos los datos disponibles.

Leave-one-out

Leave-one-out es una variación de la validación cruzada, donde k = número de ejemplos. De esta manera, en un conjunto de datos con N ejemplos, se efectuarán N experi-

mentos, dejando en cada uno de ellos solo uno de los ejemplos para la evaluación. Se puede visualizar un ejemplo en la Tabla 2.4.

	Número total de ejemplos				
Prueba 1	E	E	...	E	**T**
Prueba 2	E	E	...	**T**	E
Prueba 3	E	E	...	E	E
...	...	...	...	...	...
Prueba N	**T**	E	E	E	E

Tabla 2.4: Ejemplo *leave-one-out*.

2.4.2. Métricas de evaluación

Métricas básicas

Las métricas que se suelen utilizar para clasificación de una etiqueta (Van Asch, 2013), se basan en la matriz de confusión (Tabla 2.5):

		valores predichos	
		positivo	negativo
valores reales	positivo	true positive (tp)	false negative (fn)
	negativo	false positive (fp)	true negative (tn)

Tabla 2.5: Matriz de confusión.

De esta manera, para la clasificación de una clase A, se obtendría:

1. tp: instancias pertenecientes a la clase A, clasificadas correctamente como clase A.

2. tn: instancias no pertenecientes a la clase A, que no se clasifican como clase A.

3. fp: instancias no pertenecientes a la clase A pero que se clasifican como clase A.

4. fn: instancias pertenecientes a la clase A, pero que no se clasifican como clase A.

Sobre ella, se calculan las siguientes métricas explicadas a continuación.

La precisión (*precision*), descrita en la Ecuación 2.4, indica la proporción de etiquetas verdaderas acertadas, respecto a la totalidad de etiquetas predichas como positivas. En otras palabras, mide que las instancias clasificadas como clase A sean realmente de esa clase, aunque puede haber instancias de la clase A clasificadas como otra clase.

$$precision = \frac{tp}{tp + fp} \tag{2.4}$$

La métrica *recall*, descrita en la Ecuación 2.5, indica la proporción de etiquetas verdaderas acertadas respecto a las instancias pertenecientes a la clase. En otras palabras, mide que las instancias clasificadas como clase A se clasifiquen como clase A, aunque puede haber otras instancias clasificadas como clase A sin serlo.

$$recall = \frac{tp}{tp + fn} \tag{2.5}$$

La métrica *accuracy*, descrita en la Ecuación 2.6, representa la proporción de instancias correctamente clasificadas (pertenecientes y no pertenecientes) respecto al total.

$$accuracy = \frac{tp + tn}{tp + tn + fn + fp} \tag{2.6}$$

A fin de establecer una métrica que recoja la precisión y el *recall*, se puede calcular F_β, recogido en la Ecuación 2.7:

$$F_\beta = (1 + \beta^2) \times \frac{precision \times recall}{(\beta^2 \times precision) + recall} \tag{2.7}$$

Siendo común que el valor de β sea igual a 1, generándose en tal caso la medida F_1.

Una vez presentadas las métricas básicas, se pueden pasar a mencionar las diferentes métricas existentes para evaluar clasificadores multietiqueta.

Métricas basadas en ejemplos

La idea de las métricas basadas en ejemplos parte de lo siguiente: una vez se han clasificado todos los ejemplos, se realiza una comparación para cada uno de ellos entre las etiquetas predichas y las reales. Después, se realiza la media de todos los ejemplos para obtener la métrica que represente al modelo en cuestión.

Para realizar dicha comparación entre etiquetas, hay diferentes métodos, siendo uno de los más utilizados la distancia de Hamming o *Hamming-loss*, que realiza una comparación binaria. De esta forma, se realizaría una diferencia de conjuntos entre las etiquetas reales y las predichas, es decir, se aplicaría el operador XOR. Por ello, se estarían teniendo en cuenta tanto una etiqueta predicha de forma incorrecta, como una etiqueta real no predicha. Dicha métrica tendría su mínimo en 0, lo que significaría que las etiquetas predichas y las reales son iguales, y su máximo en 1, que representaría el caso contrario, donde no se ha acertado ninguna etiqueta.

Su expresión matemática sería ha sido recogida en la Ecuación 2.8.

$$\textit{Hamming-loss} = \frac{1}{N}\sum_{i=1}^{N}\frac{Y_i \triangle Z_i}{M} \tag{2.8}$$

donde:

- N: número total de ejemplos.
- Y_i: conjunto de etiquetas reales de un ejemplo i.
- Z_i: conjunto de etiquetas predichas de un ejemplo i.
- M: número total de etiquetas.
- $\triangle$: operador XOR de conjuntos.

Métricas basadas en etiquetas

Las métricas basadas en etiquetas descomponen el proceso de evaluación en evaluaciones separadas para cada etiqueta, para luego calcular la media de todas ellas.

Las métricas que se utilizan normalmente para evaluar clasificadores binarios (explicadas en la sección métricas básicas) pueden ser utilizadas aquí, variando su forma de calcular según el enfoque.

Existen dos enfoques: el enfoque micro (*micro-averaging*) y el enfoque macro (*macro-averaging*). En el primero de ellos, el micro, se construye la matriz de confusión de todas las instancias, y luego se calcula la media global. En contraste, en el enfoque macro, se calcula individualmente para cada etiqueta, y luego se obtiene la media global.

$$B_{macro} = \frac{1}{q}\sum_{\lambda=1}^{q} B(tp_\lambda, fp_\lambda, tn_\lambda, fn_\lambda) \tag{2.9}$$

$$B_{micro} = B\left(\sum_{\lambda=1}^{q} tp_\lambda, \sum_{\lambda=1}^{q} fp_\lambda, \sum_{\lambda=1}^{q} tn_\lambda, \sum_{\lambda=1}^{q} fn_\lambda\right) \tag{2.10}$$

Donde:

- B: medida de evaluación binaria.
- tp: *true positive*, tn: *true negative*, fp: *false positive*, fn: *false negative*.
- λ: una etiqueta del conjunto de etiquetas $L = \{\lambda_j : j = 1..q\}$.

Métricas basadas en ranking

Las métricas basadas en ranking están orientadas a evaluar uno de los enfoques mencionados: *label ranking*, el cual no es objeto del presente trabajo. Sin embargo, se han querido mencionar dos métricas habituales en este tipo de enfoque: *one-error* y *average-precision*. Se puede encontrar más información sobre ellas en (Nasierding y Kouzani, 2012b) y (Sorower, 2010, pp.15-16).

2.5. Tecnologías aplicadas

Para el desarrollo del presente trabajo se requería lo siguiente: una fuente científica de donde sacar los resúmenes de los textos, que contara con algún método que facilitara la adquisición automática de los mismos. También se necesitaba una herramienta de procesador de textos, para así normalizarlos; así como una herramienta

que permitiera aplicar algún método de análisis de frecuencias de palabras. Finalmente, se requería de algún instrumento que permitiera generar los clasificadores. La valoración de alternativas y la elección final se comenta a continuación. Además, se ha incluido la herramienta que ha sido utilizada para generar la documentación.

2.5.1. BibSonomy

Para la selección de los resúmenes con los cuales entrenar los clasificadores, se valoraron diferentes páginas web que cumplieran con dos requisitos fundamentales: debían albergar una cantidad suficiente de textos científicos y debían estar categorizados por etiquetas. Por ejemplo, las páginas webs de los periódicos nacionales, como puede ser El País, tiene categorizados los artículos por etiquetas, pudiendo un artículo estar en varias categorías, como podría ser banca y economía. Aunque esa era la idea, faltaba el requisito del carácter científico que se buscaba otorgar a este trabajo. Además, se necesitaba una descripción breve del mismo, que permitiera la escalabilidad del problema. De esta forma, se encontró *BibSonomy*, (BibSonomy), que es un sistema para organizar y compartir marcadores (*bookmarks*) y publicaciones científicas. Además, dicha página ofrece una API, mediante la cuál se puede acceder a una librería escrita en Java, que permite la interacción directa con la página web desde el código. Esto permite el acceso y la recolección automática de numersos textos, algo necesario para obtener un número suficiente de ejemplos de entrenamiento en un tiempo razonable.

Eclipse

Para la utilización de la API anteriormente mencionada, se decidió emplear el entorno de desarrollo Eclipse (Eclipse), con el que se estaba familiarizado, por lo que resultaba sencilla su utilización. Más concretamente, se utilizó Eclipse IDE para desarrolladores de Java, en la versión: Mars.1 Release (4.5.1).

Eclipse permite la incorporación de la biblioteca que provee *BibSonomy* a un proyecto, para así poder utilizar los métodos que se conectan con la página web mencionada. Además, es apto para albergar la información recogida en una estructura de datos que permita su modificación, así como la escritura en un fichero externo donde guardar dichos datos.

2.5.2. NLTK

NLTK (o *Natural Language Toolkit*), es decir, herramienta del lenguaje natural) es una plataforma desarrollada en el lenguaje de programación Python, la cuál permite el procesamiento del lenguaje natural. Dicha herramienta cuenta, entre otros, con numerosos métodos para normalizar textos, pudiendo eliminar mayúsculas, palabras frecuentes, y otras acciones similares.

En este TFG se ha utilizado NLTK para tratar con los ejemplos que se obtenían directamente de la página web anteriormente mencionada, más concretamente para eliminar todo tipo de símbolos, mayúsculas o números; para eliminar textos en otros idiomas (solo se van a tratar con textos que estén escritos en inglés), para eliminar las palabras más frecuentes de un idioma (*stop words*) y para llevar a las palabras a su raíz.

2.5.3. Scikit-Learn

Scikit-learn (Pedregosa et al., 2011) es una librería de AA desarrollada en Python, que provee herramientas para la minería de datos y su análisis.

En este TFG ha sido utilizada para efectuar el análisis de la frecuencia de palabras mediante la elaboración de la matriz TF-IDF. Una de las razones por las cuales se eligió esta librería se corresponde con su fácil acoplamiento con la herramienta utilizada para preprocesar los textos, NLTK, la cuál también está escrita en Python.

2.5.4. Meka

Meka (Read et al., 2016) es una extensión multiquetiqueta basada en Weka (Mark Hall, 2009), la cual es, a su vez, una herramienta de aprendizaje automático. De esta manera, gracias a Meka, se pueden realizar las operaciones básicas que se pueden desarrollar en Weka, así como utilizar los algoritmos específicos para clasificación multietiqueta (tanto de transformación de problemas como de adaptación de algoritmos) propios de la clasificación multietiqueta. Además, una vez se realiza la ejecución, muestra al usuario numerosas métricas de evaluación, vitales para la valoración de los diferentes modelos obtenidos. Una de las grandes ventajas de Meka es que ofrece una interfaz gráfica, lo que facilita su uso.

En el caso del presente trabajo, se ha utilizado Meka para clasificar los textos obtenidos de la página web *BibSonomy*. Dichos textos, fueron preprocesados con técnicas de *text mining*, para obtener después una serie de atributos que representaban las palabras más representativas de cada uno, a través de la matriz TF-IDF, generándose así el documento de entrada a Meka. Además, se ha utilizado para analizar los diferentes modelos obtenidos.

Para su utilización, el documento que será utilizado para entrenar el modelo deberá estar en formato ARFF (*Attribute-Relation File Format*), el cual está dividido en dos secciones:

1. La cabecera: se debe especificar el nombre de la relación, seguido del comando -C y el número de etiquetas o clases que se van a diferenciar. A continuación, debe figurar una lista con los atributos y sus tipos, que pueden ser: *numeric, nominal, string* y *date*. Si el atributo tiene un conjunto finito de valores, pueden ser expresados entre llaves, por ejemplo: $\{0, 1\}$ para etiquetas binarias. Es habitual que la especificación de las clases figure en primer lugar de los atributos.

2. Los datos, donde se incluirá el valor de cada atributo para cada ejemplo siguiendo el orden expresado en la cabecera.

Asimismo, los algoritmos que ofrece se pueden dividir en varias categorías:

- Métodos de Relevancia Binaria, tales como *binary relevance, classifier chains, ensemble classifier chains* entre otros.

- Métodos *Label Power Set*: *Label powerset, RAkEL* y otros.

- Métodos por parejas: son aquellos que llevan a cabo comparaciones por pares (*pairwase*).

- Otros métodos: en esta categoría se encuentran, entre otros, métodos basados en redes de neuronas, lo que se correspondería con la técnica de adaptación de algoritmos tradicionales a multietiqueta.

Además, Meka cuenta con dos modos: el *experimenter* y el *explorer*. El primero de ellos, permite realizar una serie de pruebas con diferentes clasificadores a la vez,

así como seleccionar el número de repeticiones y el sistema de evaluación que se quiere utilizar. Además, cuando se ejecuta, permite guardar los resultados en un formato muy útil para su posterior interpretación. El segundo de ellos, está más orientado a realizar pruebas concretas, pues cada vez que se quiera probar un clasificador, se tiene que seleccionar a mano y no de forma agregada como el anterior. Asimismo, los resultados sólo se podrían guardar uno por uno, lo que alargaría el tiempo de recogida de los resultados.

Otras alternativas

Además de Meka, existen otras herramientas que soportan la clasificación multietiqueta. En este TFG se ha valorado la utilización de dos de ellas: la librería en Python que provee Scikit-learn, y la librería en Java denominada Mulan (Tsoumakas et al., 2011).

Scikit-learn Implementa las estrategias denominadas como *one-against-all*, *one-against-one*, y *Error-Correcting Output Coding* (ECOC). Una de las ventajas de la utilización de esta librería es que está escrita en Python, el mismo lenguaje utilizado para el preprocesamiento y limpieza de los datos, por lo que no habría sido necesario migrar los datos a otra plataforma.

Mulan Incluye diferentes algoritmos cuyo objetivo es resolver un problema multietiqueta, tanto diferenciando entre una serie de etiquetas que son relevantes o no, como generando una clasificación ordenada de las etiquetas más relevantes, así como una combinación de ambas. Esta librería estaba escrita en Java, y su incorporación a un proyecto, por ejemplo en el entorno Eclipse, no es del todo sencillo.

Ambas librerías, como su nombre indica, no cuentan con una interfaz gráfica que facilite la interacción del usuario con el sistema, sino que se debe implementar en el código los métodos que ellos proveen. Esto supone una dificultad añadida si no se tiene experiencia previa con este tipo de lenguajes o herramientas, lo cual era el caso. Por otro lado, de ello también se deriva una ventaja, la generación de varios modelos a la vez es mucho más rápida que si se debiera hacer mediante el uso de una interfaz, donde normalmente solo se puede generar un modelo cada vez. Aunque

dicha flexibilidad fue tenida en cuenta a la hora de elegir qué software utilizar, se consideró que Meka ofrecía las herramientas necesarias para una buena resolución del problema planteado en este TFG, aunque la evaluación de los modelos fuera más tediosa.

2.5.5. LaTeX

Para la elaboración de la documentación se decidió utilizar LaTeX, un sistema de preparación de documentos. La principal ventaja de su utilziación se deriva de que permite separar el contenido de la forma, permitiéndo al autor centrarse en en lo qué quiere decir, y dejando que sea LaTeX quien se ocupe de quede de una forma profesional y presentable. Por otro lado, este sistema tiene una curva de aprendizaje elevada, es decir, si no se tiene experiencia previa con él, hay un período de tiempo que deberá dedicarse a investigar cómo escribir ciertos comandos o cómo estructurar el documento.

Como alternativa a él, se podría haber utilizado Microsoft Word, una herramienta similar al procesamiento de texto, pero se detectaron dos inconvenientes: el sistema estaba completamente desarrollado en Ubuntu (Linux), por lo que se debería haber utilizado o bien una máquina virtual o bien realizar la memoria en otro equipo. Es cierto que Ubuntu ofrece una alternativa parecida a ella, LibreOffice Writer, pero se considera que tiene demasiadas limitaciones. El segundo inconveniente se deriva del trato de la bibliografía, pues ésta se debería haber hecho en una herramienta a parte o bien utilizar la que ofrece Word, cuya utilización es bastante engorrosa.

Además, como LaTeX es una herramienta de código abierto, son muchas las personas que han contribuido a que su utilización sea más sencilla, sobre todo para gente principiante. En este trabajo se ha utilizado una plantilla, denominada *ezthesis*, que permite emular con facilidad el formato propio de una tesis doctoral. Aunque en proyecto no se corresponde con una tesis, se consideró que el formato podía ser similar. Su utilización a permitido reducir enormemente el tiempo de acondicionamiento para utilizar LaTeX, así como conseguir el formato profesional requerido. Además, incluye un archivo orientado a la bibliografía, mediante el cual se pueden introducir las referencias sin ningún tipo de problema.

2.5.6. Mención al Marco Legal

A la hora de escoger las diferentes tecnologías utilizadas en este trabajo, se revisaron los aspectos legales referentes a su uso. Por ello, todas herramientas utilizadas para el desarrollo del sistema son de software libre. En las páginas web de donde fueron descargadas así se manifiesta.

La única parte de este sistema donde se utiliza información externa se corresponde con la extracción de los datos de la página web BibSonomy. En ella, existe una sección referida a los términos de uso, donde se explica al usuario que BibSonomy no tiene ningún interés en el perfil privado de sus usuarios, aunque se entiende que ellos le otorgan el uso de los materiales que publican. Asimismo, se expresa que las publicaciones de carácter publico podrán ser accedidas por otros usuarios, como lo es en este caso.

Capítulo 3

Descripción del trabajo realizado

En esta sección se ha tratado de explicar el trabajo que ha sido desarrollado para lograr el sistema. En primer lugar, se presentará la metodología que ha servido de base para desarrollar las siguientes fases, las cuales son: el análisis, el diseño, la implementación y la evaluación, aunque ésta última fase será presentada en una sección aparte.

3.1. Metodología

Antes de comenzar a explicar el trabajo realizado, es necesario mencionar qué metodología se ha aplicado en este proyecto a fin de completar su desarrollo: el Modelo de Ciclo de Vida. Dado que al comienzo del proyecto no se tenía del todo claro cómo se iba a poder desarrollar, fue necesario avanzar y retroceder, a fin de que se fuera averiguando cómo podían resolverse determinados aspectos. Esta filosofía está representada en el Modelo de Desarrollo Incremental, el cual está basado en el Modelo Iterativo basado en Casacada. Éste último parte de las fases que componen el Modelo en Cascada, es decir, análisis, diseño, codificación y pruebas, pero siendo repetidas en diferentes ciclos. El segundo modelo citado, de desarrollo incremental, se basa en esa idea, pero en cada ciclo se va desarrollando una parte del sistema (de Calidad del Software de INTECO, 2009). De esta manera, al dividir el problema principal en partes más pequeñas, se logra centrar la atención en la parte que se está desarrollando sin preocuparse, todavía, por las demás. Además, como en cada ciclo se van

repasando las fases anteriores, es posible añadir modificaciones si fuera necesario. Se puede ver una representación gráfica en la Figura 3.1. Dado que este documento representa la versión definitiva, no se podrán observar dichas iteraciones ni cómo se ha ido desarrollando el sistema, pero su desarrollo se basó en dicha metodología

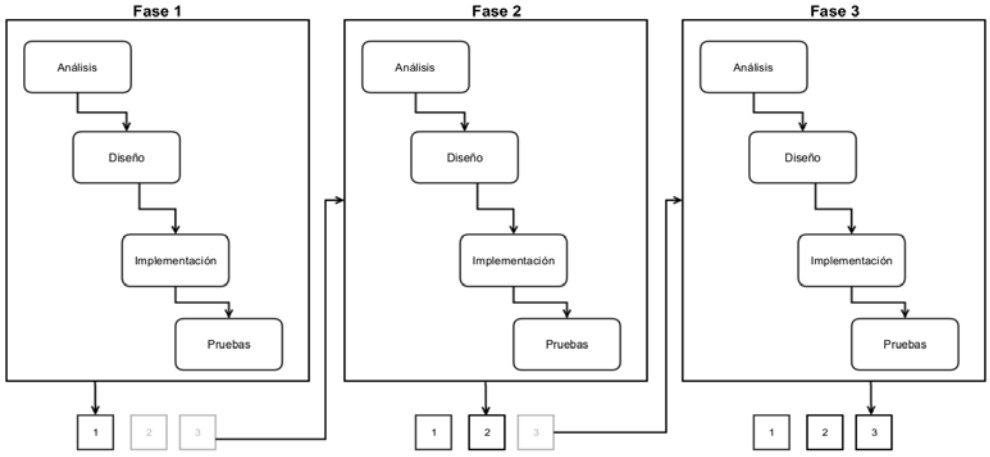

Figura 3.1: Metodología Modelo de Desarrollo Incremental

3.2. Análisis

Como ya se ha explicado, el proyecto a desarrollar consiste en la clasificación multietiqueta de textos de carácter científico. El objetivo final, por tanto, será que, a partir del resumen de un texto, se pueda concluir una serie de etiquetas que definan su contenido. No era objetivo de este trabajo desarrollar una interfaz para facilitar a un posible usuario una interacción con el sistema, ni tampoco que estuviera integrado en una sola plataforma. Se busca, por tanto, realizar una investigación acerca de algunos de los diferentes métodos existentes para resolver este tipo de problemas, y concluir qué acercamiento resulta el más idóneo para ello. Debido a ello, se utilizarán diferentes softwares, siendo la salida de uno la entrada de otro, lo que se denominará 'sistema'.

De esta forma, las diferentes soluciones que se desarrollaron varían, principalmente, en dos aspectos: el software elegido para desarrollar los clasificadores, y la

elección de los clasificadores en sí. Dichas soluciones se comentarán en la tercera fase de este capítulo, es decir, en la implementación. En esta, sin embargo, se pasarán a recoger aquellos aspectos que, independientemente del software o clasificador elegido, sea necesario que soporten. Esto se recogerá mediante los requisitos del sistema.

Después de ellos, se presentarán los casos de uso, que son utilizados para mostrar la interacción de un usuario o programa externo, denominado actor, con el sistema.

3.2.1. Requisitos del sistema

Tomando como base las funcionalidades requeridas, los requisitos se dividen en dos categorías:

1. Requisitos funcionales: son aquellos que representan las acciones fundamentales que debe desarrollar el sistema, aceptando y procesando los *inputs* y procesando y generando los *outputs*.

2. Requisitos no funcionales: son aquellos que describen cómo debe hacer el sistema lo anteriormente descrito.

Para representar cada requisito, se ha decidido utilizar las recomendaciones presentadas en el estándar IEEE 830 (IEEE, 1998), adaptándolos a las características del presente trabajo. De esta forma, se ha desarrollado la siguiente tabla (3.1), que albergará una serie de campos que se comentarán a continuación:

`Identificador`	
`Nombre`	
`Prioridad`	
`Necesidad`	
`Descripción`	

Tabla 3.1: Ejemplo representación de requisitos del sistema

Los campos contenidos son:

1. Identificador: representa la identificación unívoca de cada requisito. Para ello, se denominará cada requisito de la siguiente forma:

- Requisitos funcionales: RF-XX, donde XX se corresponderá con la numeración de los requisitos funcionales.
- Requisitos no funcionales: RNF-XX, donde XX se corresponderá con la numeración de los requisitos nfuncionales.

2. Nombre: otorga más información sobre qué trata el requisito, pero sin entrar en tanto detalle como la descripción.

3. Prioridad: representa la importancia asociada al requisito, existiendo tres categorías: alta, media y baja.

4. Necesidad: es una forma de clasificar los requisitos, pudiendo ser:

   - Esencial: el sistema no se considerará aceptable si no cumple esos requisitos.
   - Condicional: el sistema se verá mejorado si se implementan estos requisitos, pero seguirá siendo válido sin ellos
   - Opcional: representan aquellos requisitos que suponen un añadido sobre el sistema inicial.

5. Descripción: recoge la explicación extendida del requisito.

Requisitos funcionales

A continuación, se presentan los requisitos funcionales del sistema.

`Identificador`	RF-1
`Nombre`	Ejemplos de entrenamiento
`Prioridad`	Alta
`Necesidad`	Esencial
`Descripción`	El sistema recogerá los ejemplos de entrenamiento de forma automática.

Tabla 3.2: Requisito Funcional - 1

Identificador	RF-2
Nombre	Ejemplos de entrenamiento
Prioridad	Alta
Necesidad	Esencial
Descripción	El sistema realizará las búsquedas de los ejemplos a través de etiquetas que representen de qué trata su contenido.

Tabla 3.3: Requisito Funcional - 2

Identificador	RF-3
Nombre	Preprocesamiento y limpieza
Prioridad	Alta
Necesidad	Esencial
Descripción	El sistema preprocesará y limpiará cada ejemplo.

Tabla 3.4: Requisito Funcional - 3

Identificador	RF-4
Nombre	Frecuencia de palabras
Prioridad	Alta
Necesidad	Esencial
Descripción	El sistema aplicará algún tipo de análisis de frecuencia de palabras

Tabla 3.5: Requisito Funcional - 4

`Identificador`	RF-5
`Nombre`	Métodos de clasificación
`Prioridad`	Alta
`Necesidad`	Esencial
`Descripción`	El sistema utilizará varios métodos de clasificación multietiqueta

Tabla 3.6: Requisito Funcional - 5

`Identificador`	RF-6
`Nombre`	Métricas de evaluación
`Prioridad`	Alta
`Necesidad`	Esencial
`Descripción`	El sistema generará varias métricas de evaluación de algoritmos

Tabla 3.7: Requisito Funcional - 6

Requisitos no funcionales

Los requisitos no funcionales son:

`Identificador`	RNF-1
`Nombre`	Número de etiquetas
`Prioridad`	Alta
`Necesidad`	Esencial
`Descripción`	El sistema contará con un número limitado de etiquetas.

Tabla 3.8: Requisito No Funcional - 1

`Identificador`	RNF-2
`Nombre`	Número de etiquetas
`Prioridad`	Alta
`Necesidad`	Esencial
`Descripción`	El sistema soportará búsquedas de ejemplos pudiendo combinar varias etiquetas de contenido a la vez.

Tabla 3.9: Requisito No Funcional - 2

3.2.2. Casos de uso

Los casos de uso describen el comportamiento del sistema[1] bajo difrentes condiciones, según éste va respondiendo a las peticiones de uno de los *stakeholders*, relacionado con un objetivo en particular (Cockburn, 1999). En dicha definición, se encuentran varios conceptos que es necesario explicar:

- *Stakeholder*: algo o alguien interesado en el comportamiento del sistema.
- Actor[2]: es un *stakeholder* que inicia una interacción con el sistema con el objetivo de completar un objetivo.

Para lograr la descripción de los casos de uso, se ha utilizado el proceso sugerido por (Cockburn, 1999). En él, después de fijar el alcance y los límites del sistema, algo que ya se ha hecho mediante los requisitos, se procede a averiguar los actores y los objetivos de los mismos.

Se quiere señalar que los casos de uso aquí recogidos se corresponden con una versión definitiva, en la que ya se ha incluido parte del diseño, que se estudiará en la siguiente sección. Se ha considerado más útil así porque de esta forma se puede profundizar más en qué herramientas concretas deben utilizar los actores y cómo.

A la hora de describir el caso de uso, se ha decidido utilizar la siguiente tabla (3.10), cuyos elementos se explican a continuación:

[1] A. Cockburn diferencia entre 'sistema'y 'sistema bajo construcción', aunque aquí se ha decidido unificar en un solo concepto.

[2] Ocurre lo mismo con la denominación de actor, ya que A. Cockburn diferencia entre 'actor'(que lo definiría como: algo o alguien con comportamiento.) y 'actor principal'. Se ha obviado dicha diferenciación porque se considera que no es necesario en el presente trabajo.

`Identificador`	
`Nombre`	
`Actor`	
`Objetivo`	
`Descripción`	
`Precondiciones`	
`Postcondiciones`	

Tabla 3.10: Ejemplo representación de casos de uso

Los diferentes campos son:

- Identificador: utilizado para identificar unívocamente a cada caso de uso. Se utilizarán las letras 'CdU-'seguido de un número que lo señale.
- Nombre: identificativo muy breve que es útil para saber de qué trata el caso de uso.
- Actor: describe el actor que inicia el caso de uso.
- Objetivo: representa el objetivo que quiere completar el actor al iniciar la interacción.
- Descripción: explicación detallada del proceso que recoge el caso de uso.
- Precondiciones: condiciones que deben ocurrir antes de que se produzca el caso de uso.
- Postcondiciones: estado del sistema que acaecerá una vez se ha completado el caso de uso.

Actores

En este caso, la única persona que interactuará con el sistema será el desarrollador, pues el presente trabajo no está orientado a desarrollar una aplicación de cara a posibles usuarios externos. Tampoco se considera que haya sistemas externos

que vayan a interactuar con él, pues, aunque se realiza una conexión con la página web *Bibsonomy*, realmente no es dicha página quién inicia el proceso, sino el propio sistema.

Dicho desarrollador interactuará con el sistema para lograr diferentes objetivos. Se han identificado tres objetivos, relacionados con el proceso de desarrollo del clasificador: el entrenamiento, la ejecución y la evaluación.

Descripción de casos de uso

`Identificador`	CdU - 01
`Nombre`	Extracción de la información y generación de los conjuntos de entrenamiento
`Actor`	Desarrollador
`Objetivo`	Preparar el entrenamiento del clasificador
`Descripción`	1. El desarrollador señala la lista de etiquetas que desea que el clasificador diferencie en el programa de Eclipse. A continuación, lo ejecuta, obteniendo el archivo con los resúmenes para cada etiqueta. 2. Después, ejecuta el programa en Python, el cual preprocesa los resúmenes y genera la matriz IF- IDF. La ejecución termina con la obtención de otro fichero, acorde con el formato de Meka.
`Precondiciones`	El desarrollador debe haber abierto Eclipse, así como haber obtenido un usuario y una contraseña que validen su conexión con *Bibsonomy*, y Python.
`Postcondiciones`	Generación de un archivo acorde con el formato de Meka.

Tabla 3.11: Caso de uso - 01

`Identificador`	CdU - 02
`Nombre`	Construcción de los modelos
`Actor`	Desarrollador
`Objetivo`	Realizar el entrenamiento del clasificador
`Descripción`	1. El desarrollador accede al modo *Experimenter* de Meka. 2. El desarrollador añade los clasificadores que quiere entrenar. 3. El desarrollador añade el (los) archivo(s) .arff que contiene(n) los conjuntos de entrenamiento. 4. El desarrollador configura las pruebas, estableciendo el número de repeticiones y la forma de evaluación. 5. El desarrollador aplica los cambios y ejecuta la prueba.
`Precondiciones`	- Se debe haber desarrollado el caso de uso: CdU - 01 - Se debe haber abierto Meka
`Postcondiciones`	Visualización en la aplicación de Meka de las métricas de evaluación en función del modelo elegido

Tabla 3.12: Caso de uso - 02

`Identificador`	CdU - 03
`Nombre`	Evaluación
`Actor`	Desarrollador
`Objetivo`	Elegir el mejor clasificador
`Descripción`	1. El desarrollador visualiza las diferentes métricas obtenidas en la aplicación Meka. 2. El desarrollador elige aquel que mejores valores haya obtenido, según los parámetros fijados. 3. El desarrollador guarda el modelo asociado a dichos valores.
`Precondiciones`	Se debe haber desarrollado el caso de uso: CdU - 02
`Postcondiciones`	Obtención del modelo que representa el mejor clasificador obtenido

Tabla 3.13: Caso de uso - 03

3.2.3. Diagrama de secuencia

La utilización de un Diagrama de Secuencia puede ser útil para ejemplificar el modelado de escenarios de uso, que refleja un paso a través de la lógica que está contenida en varios casos de uso (Zapata y Garcés, 2008). En este caso, se ha utilizado para facilitar los casos de uso anteriormente expuestos.

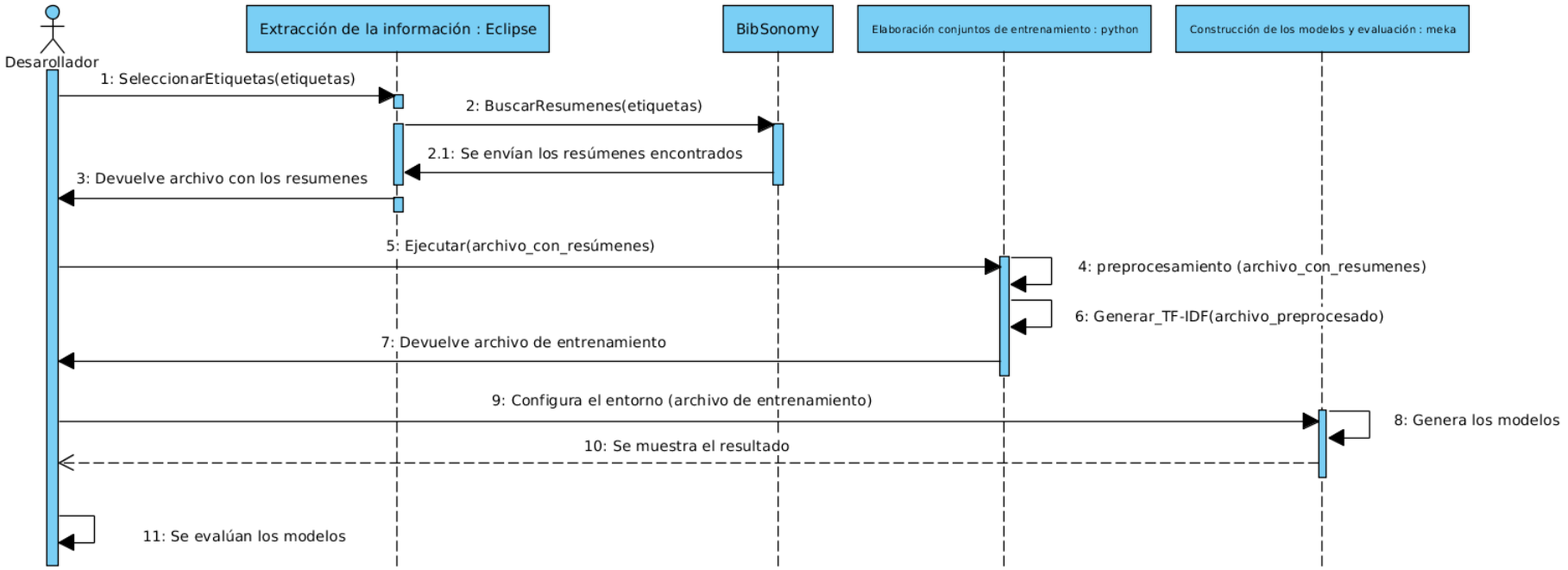

Figura 3.2: Diagrama de Secuencia.

3.3. Diseño

3.3.1. Arquitectura del sistema

La arquitectura del software de un sistema informático o programa es una estructura o estructuras del sistema, que comprende los elementos del software, sus propiedades externas, así como las relaciones entre ellos. Es, por tanto, una abstracción del sistema que suprime detalles que no afectan a su uso o a cómo interaccionan con otros elementos, y donde el comportamiento de cada elemento supone una parte de la arquitectura (Clements, 2003).

En cuanto a la estructura de la arquitectura elegida, se cree que la que más se adecúa al sistema que se quiere desarrollar es aquella que se estructura en módulos, que representan unidades de implementación y se relacionan entre sí mediante una relación de uso, en la que una requiere la presencia de una versión correcta de otra (Clements, 2003, pp. 35-40). En este caso, habría cuatro, cuyas responsabilidades serían las siguientes, y los cuales se pueden observar en la Figura 3.3:

- Módulo 1: extracción de la información. Este módulo se comunica con la página web *BibSonomy* a través de la API, y tiene como salida dos ficheros: uno con los resúmenes en forma de texto, y otro con las etiquetas de cada resumen.

- Módulo 2: elaboración del conjunto de entrenamiento. Este módulo es el encargado del procesamiento del texto y la extracción de las características. Como salida, generará la matriz TF-IDF, que recoge los valores de dichas características.

    - Módulo 2.1: generación fichero de entrenamiento. Tomando como base la matriz anteriormente citada y el software que se vaya a utilizar a continuación, en este caso Meka, se genera un determinado tipo de archivo de entrenamiento.

- Módulo 3: construcción de los modelos y entrenamiento. A partir del fichero de entrenamiento generado en el anterior módulo, se generarán los modelos elegidos, realizándose el entrenamiento de los mismos. Como salida de este módulo se obtendrían los modelos entrenados.

- Módulo 4: evaluación. En este módulo, se procederá a analizar el valor de las métricas para cada uno de los modelos entrenados. En este caso, la salida sería las métricas de evaluación.

Tal y como se comentó en la Sección 3.1, el desarrollo de este sistema estuvo compuesto por varios ciclos, desarrollándose en cada uno de ellos un módulo de los aquí referidos.

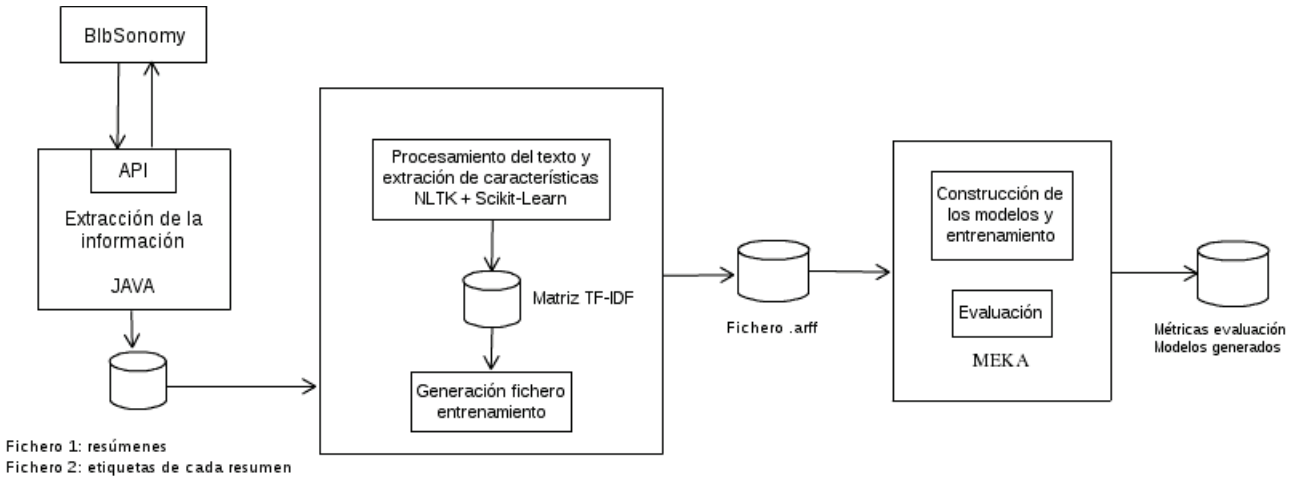

Figura 3.3: Representación de la Arquitectura del Sistema

3.4. Implementación

En esta sección se va a comentar la implementación de cada uno de los módulos identificados en la arquitectura del sistema, los cuales son: extracción de los datos, la elaboración del conjunto de entrenamiento, la generación de los modelos y la evaluación. Además, se ha incluido una sección previa, referente al acondicionamiento del entorno de trabajo para poder comenzar la implementación.

3.4.1. Preparación del entorno de trabajo

El primer paso para poder comenzar a desarrollar el sistema era preparar el entorno necesario para ello. En un primer momento, se empezó a utilizar como sistema

operativo Windows 10, pues se creyó que las herramientas que se iban a utilizar admitían dicho sistema. Con la API de *BibSonomy* no hubo problema, pues se ejecutaba en Eclipse, sin embargo, el problema se detectó cuando se comenzaron a utilizar las plataformas como *NLTK* o *Scikit-learn*, para las cuales se dificultaba enormemente su instalación en Windows. Por ello, se decidió migrar el sistema a una configuración Linux.

Surgieron dos alternativas: utilizar una máquina virtual, lo cual facilitaba la instalación, o realizar una partición del disco duro, de tal forma que el sistema contara con ambos sistemas operativos, Linux y Windows, a la vez. Dado que se intuía que el sistema requeriría muchos de los recursos, sobre todo en la fase del entrenamiento, se decidió llevar a cabo la segunda opción, que permitía explotar toda la capacidad de la máquina utilizada. No obstante, esto retrasó el desarrollo del proyecto.

Una vez realizada dicha partición e instalados los sistemas operativos indicados, se procedió a instalar las diferentes herramientas, con los problemas derivados de ello, pues la experiencia que se contaba era nula.

3.4.2. Extracción de la información

La primera tarea que se debía acometer se correspondía con la obtención de los datos. Como ya se comentó, se eligió la página web BibSonomy, la cual ofrece una API. En este proyecto, se utilizó la librería de Java que proveen, la cual sólo se debe incorporar a un proyecto de Java, para así disponer de sus métodos. Mediante esta API se debía obtener un fichero donde estuvieran recogidos los resúmenes de cada artículo buscado, así como las etiquetas asociadas a cada uno de ellos. Esto se decidió hacer mediante la escritura de dos ficheros separados, uno donde se escribieran los resúmenes y, otro, donde se escribieran las etiquetas, o clases, correspondientes a cada uno. De esta manera, el orden del fichero de etiquetas, sería una especie de índice, donde la posición en el mismo indicaría a qué resúmen hacía referencia.

Una vez creados ambos ficheros, se debían establecer las etiquetas sobre las que se quería buscar información asociada a ellos. Uno de los requisitos, recogido en la Tabla 3.9, expresaba que se debían poder realizar búsquedas de varias etiquetas a la vez. Esto significa que se debía, por ejemplo, poder buscar documentos que trataran sobre *biología* y sobre *algoritmos*, para que así tuviera sentido la clasificación multi-

etiqueta. Por ello, se establecieron dos arrays de contenido: uno donde figuraban las etiquetas principales y otro donde se almacenaban las secundarias. De esta manera, cuando se fuera a realizar una consulta para recoger los datos, se combinarían en orden las de un array y otro. Esto se ha tenido que hacer así porque en el API de BibSonomy las etiquetas (*tags*) identifican conceptos de una sola palabra, por lo que si solo se guardara en un array, como 'biología algoritmos', la búsqueda se realizaría únicamente mediante el primer *tag*. Si se quisiera hacer una búsqueda mediante una sola etiqueta, bastaría con dejar la posición del segundo array relativo a ella vacío. Por ejemplo:

1. Array de etiquetas principales: ['biología', 'matemáticas', 'algoritmo']
2. Array de etiquetas secundarias: ['', 'algoritmo', 'inteligencia']

Realizaría tres búsquedas por las siguientes etiquetas: biología; matemáticas y algoritmo; algoritmo e inteligencia.

Una vez realizado lo anterior, era momento de conectarse con BibSonomy, para lo que fue necesario obtener unos credenciales, de tal forma que BibSonomy te otorgue una clave asociada a un usuario que te permita el acceso. Una vez obtenido, ya se pueden realizar las búsquedas deseadas. La API cuenta con una clase, `GetPostsQuery`, el cual permite acceder a una lista de posts, habiendo seleccionado previamente alguna característica de los mismos, como en este caso, la etiqueta o etiquetas asociadas a cada uno. De esta forma, se iteró sobre el primer array de etiquetas a buscar, se creó dicho objeto fijando como etiquetas las correspondientes en los dos arrays mencionados, y se realizó la búsqueda. Una vez se obtuvieron los resultados, que son una lista de posts, se iteró sobre ellos, de tal forma que se accedió al resúmen de cada uno, y se guardó en un fichero, así como sus etiquetas asociadas. En el caso de que la segunda etiqueta estuviera vacía, se guardó como *null*. Para separar cada resumen, se estableció un separador inequívoco que facilitara la identificación de cada uno de ellos por separado.

Cuando se realizaron las diferentes pruebas, se observó que algunas de las publicaciones que figuraban en la página web, no contaban con resumen, por lo que el campo relativo a él aparecía como `null`. Por ello, se decidió contemplar esa posibilidad antes de escribir en el fichero, no llegando a guardar aquellos artículos de los que no se contaba con el resumen.

Pseudocódigo

El pseudocódigo referente a lo anteriormente explicado está recogido en el algoritmo 1.

Entrada: lista de etiquetas a buscar, credenciales necesarios para acceder a BibSonomy
Salida: archivo de salida con los resúmenes de cada etiqueta buscada, fichero de clases con la etiqueta asociada.

```
1: Iniciar una sesión en BibSonomy con los credenciales obtenidos previamente.
2: for cada e_i del conjunto de etiquetas a buscar do
3:    Se instancia una consulta, fijando como etiquetas de la misma e1_i y e2_i.
4:    Se ejecuta la consulta, obteniendo una lista de posts.
5:    for cada post p de la lista de posts do
6:       Se accede al resumen del post p_i.
7:       if el resumen de p_i es ≠ NULL then
8:          Escribir el resumen en el archivo de salida.
9:          if e2_i = " " then
10:            Escribir la primera etiqueta del fichero de clases como e1_i y la segunda
               como null.
11:         else
12:            Escribir la primera etiqueta del fichero de clases como e1_i y como se-
               gunda e2_i.
13:         end if
14:      end if
15:   end for
16: end for
```

Algoritmo 1: Extracción de los datos

3.4.3. Elaboración del conjunto de entrenamiento

Como resultado de la ejecución del método anterior se generaron dos archivos: uno con los resúmenes de cada búsqueda, y otro con las etiquetas asociadas a cada resumen. Como se explicó, la API estaba escrita en Java, pero el software que se

decidió utilizar para generar los conjuntos de entrenamiento estaba escrito en Python. Por ello, se debían importar ambos ficheros en una estructura de datos que se pudiera utilizar en dicho lenguaje. Para facilitarlo, se decidió crear una clase que representara un artículo, el cual estaría formado por el resumen y sus etiquetas, unificándose, de esta manera, los dos archivos que se habían generado en el paso anterior.

Una vez se habían importado ambos archivos, era momento de preprocesar y limpiar los datos. Dado que los resumenes eran unidades de información que cada usuario había escrito, éstos variaban en forma y representación, por lo que era necesario unificarlos. Para ello, se realizó lo siguiente:

1. Se eliminó cualquier dígito que no fuera una letra o un espacio.

2. Se llevó a la raíz cada palabra (*stemming*). Este paso es vital para identificar correctamente la aparición de una misma palabra, pero que difiere en su forma. Por ejemplo, si no se realizara este paso, se considerarían diferentes las palabras: algoritmo y algoritmos, detectando o detectáramos, cuando en esencia representan un mismo concepto: algoritmo y detectar. Además, al llevar a la raíz la palabra, también se transforman las mayúsculas a minúsculas, para así tampoco diferenciar entre ambos.

3. Se eliminaron aquellos resumenes que no estuvieran en inglés.

4. Se eliminaron las palabras más frecuentes del idioma (*stop words*), en este caso del inglés, a fin de que no interfieran en el análisis de la frecuencia de las palabras posterior.

Una vez se hubieron aplicado las modificaciones anteriores, era momento de utilizar la librería *scikit-learn*, mediante la cual se generó la matriz TF-IDF. Esto se realizó mediante la clase `TfidfVectorizer`, donde se pueden ajustar diferentes parámetros, de los cuales solo se han utilizado cuatro:

1. `norm = 'L2'`: se refiere al método utilizado para la normalización de los vectores. En este caso, se ha decidido dejar la normalización que viene por defecto, es decir, la L2 (referida, a su vez, a la distancia Euclídea).

2. `stop_words = 'English'`: aunque se comentó en el proceso anterior que se eliminaban las palabras frecuentes, es realmente aquí cuando se lleva a cabo. Se decidió utilizar esta funcionalidad ofrecida en lugar de realizarla por separado.

3. `analyzer='word'`: se refiere a qué unidades debe analizar (si caracteres o palabras).

4. `min_df =[0.0,1.0]`: indica el umbral mínimo de frecuencia del documento. De esta manera, este valor se puede ajustar para obtener más o menos atributos, un paso clave para no tener demasiados, lo que podría provocar que el algoritmo del clasificador pudiera no resolverlo, o tardara demasiado. Dicho valor se fijó, inicialmente, en 0,05.

De esta manera, se creó un objeto de dicha clase con las características fijadas. Sobre él, se puede aplicar un método (`fit_transform`), para que se genere la matriz TF-IDF sobre el conjunto de datos que se desee, en este caso los resúmenes. Además, esta librería permite acceder a las palabras asociadas a cada peso de la matriz, algo crucial para poder visualizar los atributos.

Se ha decidido incluir un gráfico que recoja el proceso explicado en este punto, recogido en la Figura 3.4, para facilitar su comprensión.

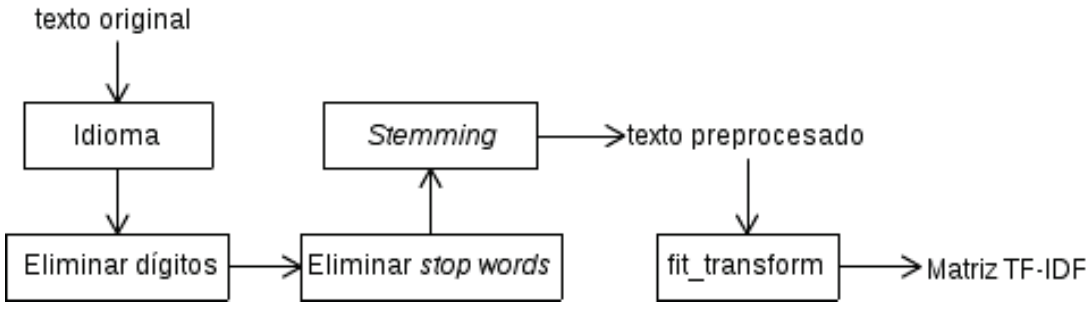

Figura 3.4: Proceso de preprocesado y limpieza

Después de que se hayan efectuado estos pasos, ya se dispone de los elementos necesarios para generar el archivo de entrenamiento: las etiquetas, que se corresponderían con las clases; los atributos, que serían las palabras obtenidas después del

preprocesamiento y limpieza; y, finalmente, las frecuencias asociadas a cada palabra, generadas en la matriz TF-IDF, que sería el valor de cada atributo. La forma de dicho archivo variará según sea el software a utilizar para la generación de los modelos.

Pseudocódigo

El pseudocódigo referente a lo anteriormente explicado está recogido en el algoritmo 2.

Entrada: archivo de salida con los resúmenes de cada búsqueda, fichero de clases con las etiquetas asociadas.
Salida: archivo de entrenamiento
1: Cargar el fichero de resúmenes en un array.
2: Cargar el fichero de clases en un array.
3: Crear un array de clases tipo artículo, con cada resumen y sus etiquetas asociadas a cada campo.
4: **for** cada resumen del array de resumenes **do**
5: Eliminar caracteres que no sean letras o espacios.
6: Eliminar resúmenes que no estén en inglés.
7: Eliminar las *stop words*.
8: Transformar cada palabra en su correspondiente palabra raíz.
9: Guardar como nuevo resumen del objeto artículo el modifcado.
10: **end for**
11: Se crea la matriz TF-IDF tomando como base los resúmenes modificados.
12: Se llama al algoritmo de generación de fichero de entrenamiento

Algoritmo 2: : Elaboración de los conjuntos de entrenamiento

Generación del fichero de entrenamiento

Se ha decidido realizar una sección aparte para evidenciar que el código hasta aquí descrito no se debería modificar si se eligiera otro software para generar los modelos. Hasta este punto, ya se contaría con todo lo necesario para adaptar la información según sean los requerimientos de dicho software.

En este caso, se ha decidido utilizar Meka, que requiere ficheros en formato ".arff",

pero también se realizaron diferentes pruebas con otras librerías, como lo es la propia que ofrece *scikit-learn* o MULAN, comentadas ya en la Sección 2.5.

El fichero que necesitaba Meka se puede dividir en tres secciónes: las clases, los atributos, y los datos. El primero de ellos, se corresponde a las etiquetas por las cuales se realizaron las búsquedas en el módulo anterior. El segundo, a las palabras asociadas a cada peso recogido en la matriz TF-IDF. Finalmente, el tercero está compuesto de una fila por artículo, donde figuran los valores de cada documento para las dos secciones anteriores: el valor de cada clase y los valores de la matriz TF-IDF de cada atributo para dicho artículo.

La escritura de las clases de cada documento tiene una dificultad añadida, pues tienen que aparecer en el mismo orden que se presentaron en la sección primera. Por ejemplo, si se dispusiera de cuatro etiquetas: A, B, C y D, y el contenido de un artículo en partícular estuviera relacionado con dos de ellas, la B y la D, en la fila de datos que se refiriera a él debería de figurar: {no, si, no, si}. Esto se solucionó buscando cada etiqueta de contenido del artículo en el array que recogía todas las etiquetas, de tal forma que devolviera la posición que tenía la etiqueta buscada en el array. Este número, indicaría cuándo se debía escribir 'si'.

El pseudocódigo está recogido en el algoritmo 3.

Entrada: array de etiquetas, matriz TF-IDF, etiquetas de cada artículo.
Salida: archivo tipo .arff

```
1: Escribir en el fichero de salida las etiquetas y sus valores posibles.
2: Escribir en el fichero de salida los atributos y su tipo.
3: for cada ejemplo do
4:   if la segunda clase del ejemplo es = NULL then
5:     Buscar en el array de etiquetas la posición de la clase primera.
6:     for cada etiqueta del array do
7:       if la posición coincide con la iteración then
8:         Escribir en el fichero de salida 'SI'
9:       else
10:        Escribir en el fichero de salida 'NO'
11:      end if
12:    end for
13:  else
14:    Buscar en el array de etiquetas la posición de la clase primera.
15:    Buscar en el array de etiquetas la posición de la clase segunda.
16:    for para cada etiqueta del array do
17:      if alguna de las posiciones coincide con la iteración then
18:        Escribir en el fichero de salida 'SI'
19:      else
20:        Escribir en el fichero de salida 'NO'
21:      end if
22:    end for
23:  end if
24:  Escribir en el fichero de salida el valor de los atributos de la matriz TF-IDF
25: end for
```

Algoritmo 3: : Generación del Fichero de Entrenamiento

3.4.4. Generación de los modelos y evaluación

Una vez se dispone del fichero de entrenamiento, se puede proceder a utilizar el software elegido para la generación de los modelos, en este caso Meka. Para ello, no

ha sido necesario programar nada, pues se ha utilizado la interfaz gráfica que provee, lo cual permite reducir el tiempo de aprendizaje de uso. Por lo tanto, sólo hubo que configurar el entorno de la prueba, los métodos que se querían aplicar, cargar el fichero de entrenamiento y ejecutar el sistema. Una vez se ha ejecutado, se pueden visualizar las métricas de evaluación en pantalla y guardar si así se requiere. Esta parte está explicada con más profundidad en el capítulo 4, pero se ha querido incluir una mención aquí para que se visualice el sistema en su totalidad.

Capítulo 4

Evaluación

El propósito de esta sección consiste en recoger y documentar las pruebas desarrolladas. En primer lugar, se presentarán las pruebas cuyo objetivo es comprobar el correcto funcionamiento del sistema implementado. Una vez realizada dicha comprobación, se puede proceder a generar los diferentes modelos, y examinar las diferentes métricas de evaluación que reflejan su comportamiento.

4.1. Funcionamiento del sistema

Las pruebas recogidas en esta sección se refieren a las partes implementadas del sistema, donde se tratará de determinar que están comportándose de la forma deseada. Se repasará, por tanto, las pruebas que fueron pasadas a la API, implementada en Java, y al fichero relacionado con el procesamiento del texto y la generación del fichero de entrenamiento, escrito en Python.

4.1.1. API

La primera prueba desarrollada fue referida a la API implementada. Ésta, debía comunicarse con *BibSonomy* y recoger los resúmenes de los textos referidos a las etiquetas especificadas. Se realizaron, por tanto, dos búsquedas de forma paralela, una de forma manual en la página web y otra a través de la API. Además, el código desarrollado se modificó para poder llevar a cabo dos pruebas:

1. Los resúmenes de los artículos a los que se accedía en la API tenían el mismo contenido que en la página. Para ello, se mostraba por pantalla en el código el resumenque se había conseguido mediante la API, y además se abrían algunos de los resúmenes mostrados en la página, comprobándose que el contenido era el mismo.

2. El número de resúmenes era el mismo en la API que en la búsqueda manual. Esto se hizo mediante un contador que recogía cuántos resúmenes se habían encontrado, número que también se puede visualizar en *BibSonomy* al realizar una búsqueda.

En un primer momento, se detectó que había una diferencia entre los resúmenes encontrados, obteniéndose más en la búsqueda manual. Profundizando un poco, se averiguó el por qué: no todas las publicaciones disponían de resumen, pues este campo puede estar en blanco si el autor no lo incluyó en su artículo. De esta manera, se observó que en la API figuraban numerosos resúmenes cuyo valor era `null`, lo que se solucionó de forma sencilla, comprobando previamente el valor del resumen antes de incluirlo al documento.

También se realizaron pruebas básicas de contenido, a fin de comprobar que la información que llegaba por la API era la misma que la recogida en la página web, así como búsquedas por varias etiquetas, para comprobar lo mismo.

Con esto, no solo se comprobó que la información para comenzar el entrenamiento era la correcta, sino que también evidenció que el espacio de búsqueda era menor de lo que se creía, pues no todos los resúmenes que figuraban servían.

4.1.2. Python

Una vez se corroboró el correcto funcionamiento de la API implementada, se quisieron verificar los siguientes aspectos del código descrito en Python, cuyo objetivo era elaborar los conjuntos de entrenamiento:

1. Importación de los datos correcta. La separación de cada resumen se llevó a cabo mediante un separador, que fue insertado en la API. En un primer momento, éste fue “abs-”, pero se observó que no salía el mismo número de resúmenes en

la API y en Python, por lo que se decidió cambiar a un separador más restrictivo, en este caso: " -MiSeparador- ", consiguiéndose la separación correcta.

2. Preprocesamiento y limpieza. Una vez aplicados los mecanismos de limpieza, se visualizaron los resultados, a fin de comprobar que no había símbolos o números, no había resúmenes en inglés, y todas las palabras habían sido llevadas a su raíz.

A la hora de generar la matriz TF-IDF, se verificó que el número de filas se correspondía con el número de ejemplos obtenidos en la API, a fin de que no se perdiera alguno. Asimismo, se modificó el valor del atributo *min_df*, visualizando como el número de atributos se reducía a medida que ese número descendía.

Finalmente, para comprobar que la escritura en el fichero era correcta, se verificó lo siguiente:

1. El valor de los atributos escritos, correspondiente a la sección de datos del fichero ".arff", era menor que el umbral seleccionado al generar la matriz TF-IDF. Esto no quiere decir que no pudiera haber atributos cuyo valor no fuera 0, sino que de aquellos en que figurara algún valor, fuera mayor o igual que dicho umbral.

2. El valor de las clases que figuraba para cada documento. Esto resultaba sencillo de comprobar porque en todo momento se guarda el orden de los ejemplos. De esta manera, los valores de las clases del fichero arff debían ser iguales que los especificados en la API.

El fichero resultante se cargó en Meka, a fin de corroborar si el formato de dicho fichero era el correcto.

4.2. Pruebas de comportamiento

El objetivo de esta sección consiste en mostrar el comportamiento de los clasificadores obtenidos. Para ello, se explicará, en primer lugar, cómo se han diseñado esas pruebas, para lo que es necesario tomar las siguientes decisiones:

- Elección del conjunto de etiquetas: a la hora de desarrollar las pruebas, se deben elegir qué etiquetas va a diferenciar el clasificador. Para ello, se explicará cuántas se han decidido escoger, así como cuáles.

- Elección de la técnica de clasificación: dado que Meka ofrece numerosos algoritmos para entrenar al clasificador, se debe decidir cuáles de ellos elegir.

- Elección de las métricas de evaluación: de igual forma al punto anterior, Meka también ofrece numerosas métricas, algunas más interesantes que otras para el problema a resolver.

- Elección del punto de referencia (*baseline*): a fin de establecer un valor con que comparar el valor de las métricas obtenidas para los modelos generados, se debe elegir un método básico de clasificación con que compararlas. De esta forma, cualquier mejora sobre él, supondrá que la labor de aprendizaje ha sido exitosa.

- Validación de las pruebas: una vez se han diseñado las pruebas, se debe especificar el modo de proceder, es decir, el número de iteraciones y la elección del conjunto de test para validar los resultados.

Una vez se supo cómo se iban a llevar a cabo las diferentes pruebas, se procedió a ejecutar la primera. Después, se fue variando el conjunto de etiquetas según se iba observando el comportamiento de los diferentes clasificadores. En cada prueba, se ha mostrado una tabla resumen donde se recoge qué etiquetas fueron escogidas para efectuar la búsqueda, así como cuantos resúmenes se obtuvieron por cada una de ellas. Asimismo, por cada prueba, se ha incluido una tabla que recoge el valor obtenido de las diferentes métricas para cada clasificador elegido.

4.2.1. Elección del conjunto de etiquetas

El primer paso a la hora de efectuar las pruebas se correspondía con la selección de clases que iba a diferenciar el clasificador. Esta decisión estaba restringida por los resúmenes que tenía *BibSonomy*, pues, aunque se quisieran realizar búsquedas combinadas de numerosas etiquetas, esto podía provocar que no hubiera ningún

resumen en la página con tal especificación. En un primer momento, se decidió fijar en 4 el número de etiquetas, teniéndose que realizar búsquedas con la combinación de todas ellas, es decir, $2^4 = 16$. Gracias a que esto se hacía de forma automática mediante la API, no suponía ningún problema.

Para seleccionar las etiquetas que iba a diferenciar el clasificador, se debía escoger un conjunto que, al combinar entre sí dichas etiquetas y realizar las búsquedas correspondientes, se lograra un número suficiente de resúmenes. Para lograrlo, dichas etiquetas debían recoger conceptos que estuvieran relacionados, de tal forma que un artículo pudiera tratar sobre ellas a la vez. Gracias a *BibSonomy* esta tarea resultó más sencilla, pues, al buscar por una etiqueta, muestra en una sección las etiquetas relacionadas (*related tags*), que son aquellas que suelen ser asignadas juntas en un artículo. Además, la página web muestra los *tags* más populares. Se procedió del siguiente modo:

1. Se seleccionó una etiqueta del conjunto de etiquetas más populares.

2. Al realizar la búsqueda seleccionada en el paso anterior, se obtuvieron las etiquetas relacionadas a ella. De ellas, se escogió una.

3. Se realizó la búsqueda con la etiqueta seleccionada en el paso 2. De ella, se calculó la intersección de sus etiquetas relacionadas, con las obtenidas en el paso 1 y 2. Si se obtenía algún resultado, sería la cuarta etiqueta a tener en cuenta. Si no, se debía repetir la búsqueda desde el paso 2.

Tras algunas pruebas, se obtuvieron las siguientes etiquetas, cuyo orden hace referencia al paso en que se eligieron: $\{analysis, processing, ai, recognition\}$. De esta forma, se puede asegurar que cada una de ellas tiene al resto de etiquetas como relacionadas.

El siguiente paso consistió en introducir las etiquetas en la parte del sistema encargada de recoger los datos, para comprobar si el número de resúmenes obtenido era lo suficientemente grande como para aprender de él. Asimismo, resultaría interesante que las clases estuvieran balanceadas. En este paso, se detectó que, a medida que se añadian etiquetas a la búsqueda, el número de ejemplos descendía bruscamente. Por ejemplo, si se trataba de buscar por las cuatro etiquetas a la vez, solo se obtenían 121 ejemplos. De esta forma, se decidió que, en un primer momento, el

número de etiquetas quedaría restringido a 2. Por lo tanto, se realizarían combinaciones de pares de un espacio de 4 elementos, sin repetición y sin que importe el orden: $C(4,2) = \frac{4!}{(4-2)!2!} = 6$.

El número obtenido de resúmenes por combinación de etiquetas se puede observar en la Tabla 4.1

Etiqueta 1	**Etiqueta 2**	**Resúmenes**
analysis	*ai*	359
analysis	*processing*	212
analysis	*recognition*	264
processing	*ai*	971
processing	*recognition*	211
ai	*recognition*	367
	Total	2384

Tabla 4.1: Número de resúmenes prueba 1

Como se puede observar, las clases están balanceadas excepto para las etiquetas $\{processing, ai\}$, donde se ha obtenido un número superior a los demás.

4.2.2. Elección de la la técnica de clasificación

Una vez se tenía dispuesto el conjunto de entrenamiento, el siguiente paso consistía en elegir qué algoritmos se iban a utilizar para la clasificación. Como ya se explicó, se decidió usar Meka, donde se pueden escoger una serie amplia de ellos. Se decidió utilizar los siguientes: BR (*Binary Relevance*), LP (*Label Power Set*), ECC (*Classifier Chains*), RAkEl (*Random k-LabEl Pruned Sets*), BPNN (*Back Propagation Neural Network*). La elección se llevó a cabo de tal forma que hubiera algunos algoritmos basados en *binary relevance* (BR, CC), otros en *label power set* (LP, RAkEL) y alguno de adaptación de algoritmos (BPNN). Además, se eligieron aquellos para los que se había estudiado su funcionamiento en este trabajo.

4.2.3. Elección de las métricas de evaluación

Para comparar los diferentes modelos generados, se debía elegir una serie de métricas de las ofrecidas por Meka que los evaluaran. De las presentadas en la Sección 2.4.2 (Métricas de evaluación), se han elegido las siguientes: *accuracy* (accu), *hamming-loss* (hamm-loss) y *F1, micro y macro*. Como ya fueron explicadas, sólo se quiere hacer una mención a la métrica F1, pues Meka otorga tres valores referentes a ella: macro por etiqueta, macro por instancia de entrenamiento (*by example*), y micro, mostrando de todas ellas la media. Sin embargo, se ha observado que la métrica F1 macro por instancia de entrenamiento y la F1 micro, suelen mostrar valores casi idénticos, por lo que se ha decidido prescindir de la F1 macro *by example*.

Además, Meka ofrece otro tipo de métricas que no han sido analizadas en el presente trabajo por no ser objeto de él, como por ejemplo algunas típicas de teoría de conjuntos o las basadas en ranking (*one-error o average-precision*). Para averiguar si había alguna otra métrica que pudiera ser interesante para evaluar los modelos generados, se han estudiado las ofrecidas, eligiéndose solo *0-1 loss*, la cual se define a continuación.

0-1 loss. Esta función asigna un coste (*loss*) de 1 en caso de que se falle en la predicción de la clase. Su expresión formal se ha recogido en la Ecuación 4.1, donde $C_X(E)$ representa la clase predicha por el clasificador X para el ejemplo E, y $C(E)$ la clase real de dicho ejemplo (Domingos y Pazzani, 1997). Un valor igual a 0 supondría que no se ha fallado en ninguna predicción, por lo que, a medida que su valor descienda, mejor será el desempeño del clasificador. del clasificador.

$$\textit{0-1 loss} = L_x(E) = \begin{cases} 0 & si\ C_X(E) = C(E) \\ 1 & en\ otro\ caso \end{cases} \tag{4.1}$$

Se quiere señalar, asimismo, que durante el estudio de las diferentes métricas ofrecidas por Meka, se encontró otra forma de representar la *accuracy*, y es mediante el índice de Jaccard o *Jaccard index*, que es un estadístico que se utiliza para medir la similitud y diversidad de un una muestra de conjuntos. Se ha definido en la Ecuación 4.2 (Hwang y Yang, 2014).

$$jaccard\_index(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.2}$$

A fin de facilitar el análisis de los resultados, en las tablas donde se recogen valores de cada prueba, se ha incluido una flecha hacia arriba (↑) o hacia abajo (↓) al lado del nombre de cada métrica. De esta forma, la primera significa que se busca maximizar el valor de la métrica en cuestión, y, la segunda, lo contrario. A modo resumen, las métricas a maximizar son: *accuracy* (accu), *F1-micro, F1-macro*, y las métricas a minimizar: *hamming-loss* (hamm-loss) y *0-1 loss*.

4.2.4. Elección del punto de referencia

A la hora de analizar los valores de las diferentes métricas obtenidas, resulta común establecer un punto de referencia o *baseline*. Para ello, se elige una técnica de clasificación básica en la que, por ejemplo, a la hora de clasificar un nuevo dato, se asigne la clase aleatoriamente o la clase mayoritaria. De esta manera, se tendrá una serie de métricas correspondientes a una clasificación sin aprendizaje. Por tanto, cualquier mejora sobre ellas, justificará la labor de aprendizaje.

En este caso, se consideró interesante que, como comparativa, se utilizara un clasificador que asignara la clase mayoritaria a cada ejemplo. Para ello, se analizaron las posibilidades que ofrecía Meka, y se decidió elegir un clasificador de relevancia binaria (BR) que utilizara para cada clasificación binaria un método denominado ZeroR. En una clasificación de una sola clase, este método asignaría la clase más frecuente a toda instancia a clasificar. En este caso, al ser multietiqueta, se aplicaría la misma idea, pero para cada etiqueta.

A modo de ejemplo, se ha generado la Tabla 4.2, que recoge las decisiones que se tomarían si se aplicara la técnica de zeroR para las etiquetas de la prueba 1. Dicha Tabla, ha sido construida a partir de los resultados de la Tabla 4.1. La decisión dependerá del número de resúmenes totales: en este caso, se obtuvieron 2384 resúmenes totales en la prueba 1: si la suma de cada etiqueta es mayor que la mitad (1192), se determinará que se debe asignar esa clase, y, si es menor, no.

Etiqueta	**Número de resúmenes**	**Decisión**
ai	359+971+367 = 1697	SI
analysis	359+212+264 = 835	NO
processing	212+971+211 = 1394	SI
recognition	211+367+264 = 842	NO

Tabla 4.2: Ejemplo de decisión del clasificador zeroR

De esta manera, para cualquier resumen que se quisiera clasificar, se le asignarían las etiquetas {*ai, processing*}, sin analizar ningún otro aspecto.

4.2.5. Validación de las pruebas

A la hora de desarrollar cada prueba, se debía fijar un número mínimo de ejecuciones de tal forma que se analizara el comportamiento de cada clasificador varias veces, así como el mecanismo de evaluación respecto el conjunto de test. En este último caso, y al estar utilizando el método *experimenter* de Meka, se disponían de dos opciones: validación cruzada, pudiéndose seleccionar el número de iteraciones, y *holdout* (denominado en Meka como *Percentage split*), pudiéndose expresar bajo qué porcentaje se quería aplicar.

Finalmente, se decidió que cada prueba estaría formada por cinco repeticiones, es decir, cada clasificador se ejecutaría cinco veces. Por otro lado, se decidió aplicar el método de validación cruzada con 10 iteraciones, a fin de que los resultados obtenidos fueran lo suficientemente veraces.

4.2.6. Primera prueba

Para la primera prueba, como ya se ha explicado, se utilizaron las etiquetas: $conjunto_{p1} = \{analysis, processing, ai, recognition\}$, combinándolas de dos en dos, obteniéndose los resultados motrados en la Tabla 4.3. Como se puede observar, los resultados distan de indicar una buena labor de clasificación, mejorándose solo en algunos casos las métricas ofrecidas por el *baseline*. Se cree que el motivo de esto puede estar derivado de la naturaleza de las etiquetas. Al estar tan relacionadas,

los atributos se parecen demasiado como para poder establecer una clasificación relevante.

	Accu ↑	**F1 macro ↑**	**F1 micro ↑**	**Hamm-loss ↓**	**0-1 loss ↓**
BR − zeroR	**0.5729**	0.3421	**0.6525**	**0.3475**	**0.5864**
BPNN	0.4617	0.3692	0.5528	0.4467	0.7266
BR	0.4520	0.3605	0.5540	0.4515	0.7703
CC	0.4782	0.3744	0.5812	0.4188	0.7279
LC	0.4805	0.3932	0.5965	0.4035	0.7516
RakEL	0.4743	**0.3948**	0.5871	0.4199	0.7527

Tabla 4.3: Resultados prueba 1

Modificación de la prueba 1

En un primer momento, se pensó que el problema podía ser que se recogían pocos atributos, es decir, que el umbral mínimo (`min_df`) seleccionado al construir la matriz TF-IDF era demasiado alto. De esta manera, se redujo su valor de 0,05 a 0,01, a fin de comprobar si se mejoraba el comportamiento del clasificador, provocando que el número de atributos pasara de ser 313 a 1058. El tiempo de construcción del modelo también se vio incrementado, tardando casi 4 veces más que en la primera prueba. Los resultados han sido recogidos en la Tabla 4.4. Si se comparan dichos resultados, se podrá observar que la mejora es ínfima, mientras que el tiempo de ejecución aumentó demasiado. Por ello, se decidió fijar el umbral en su valor inicial, 0,5, para las demás pruebas.

	Accu ↑	**F1 macro ↑**	**F1 micro ↑**	**Hamm-loss ↓**	**0-1 loss ↓**
BR − zeroR	**0.5729**	0.3421	**0.6525**	**0.3475**	**0.5864**
BPNN	0.4456	0.3578	0.5317	0.4678	0.7361
BR	0.4311	0.3573	0.5380	0.4658	0.7961
CC	0.4518	0.3668	0.5583	0.4417	0.7613
LC	0.4919	0.3950	0.6025	0.3975	0.7293
RakEL	0.4876	**0.3955**	0.5948	0.4113	0.7303

Tabla 4.4: Resultados modificación prueba 1

4.2.7. Segunda prueba

Dados los malos resultados de la primera prueba, se trató de encontrar etiquetas que estuvieran relacionadas, pero sin forzar las combinaciones. De esta manera, se describieron cinco etiquetas diferentes, con las etiquetas que figuran en el $conjunto_{p2}$, combinándose de la forma mostrada en la Tabla 4.5. El número de resúmenes obtenido para cada una de ellas también está recogido en dicha Tabla.

$conjunto_{p2} = \{analysis, processing, ai, knowledge, ontology\}$

Etiqueta 1	**Etiqueta 2**	**Resúmenes**
analysis	*processing*	212
analysis	*ai*	359
analysis	*knowledge*	84
processing	*ai*	971
ai	*ontology*	257
knowledge	*ontology*	200
	Total	2083

Tabla 4.5: Número de resúmenes prueba 2

Los resultados de esta prueba han sido recogidos en la Tabla 4.6. Los mejores resultados se pueden visualizar en los valores correspondientes al clasificador BR con zeroR, el utilizado para el *baseline*, lo que significa que la labor de aprendizaje no había sido la correcta.

	Accu ↑	**F1 macro ↑**	**F1 micro ↑**	**Hamm-loss ↓**	**0-1 loss ↓**
BR − zeroR	**0.6871**	0.4699	**0.7719**	**0.2737**	**0.5379**
BPNN	0.6128	**0.5076**	0.7276	0.3266	0.7402
BR	0.6046	0.5012	0.7285	0.3278	0.8205
CC	0.5994	0.4949	0.7219	0.3337	0.7514
LC	0.5810	0.4852	0.7055	0.3534	0.7695
RakEL	0.5914	0.4957	0.7172	0.3435	0.7866

Tabla 4.6: Resultados prueba 2.

4.2.8. Tercera prueba

Partiendo de los resultados de la prueba anterior, se decidió realizar una prueba con todas las combinaciones de las etiquetas (recogidas en el $conjunto_{p3}$), esto es, $C(5,2) = 10$ búsquedas. Dichas combinaciones, así como el número de resúmenes conseguido para cada una de ellas, han sido mostrados en la Tabla 4.7.

$$conjunto_{p3} = conjunto_{p2} = \{analysis, processing, ai, knowledge, ontology\}$$

Etiqueta 1	**Etiqueta 2**	**Resúmenes**
analysis	*ai*	359
analysis	*knowledge*	84
analysis	*processing*	212
analysis	*ontology*	37
ai	*knowledge*	529
ai	*processing*	971
ai	*ontology*	257
knowledge	*processing*	506
knowledge	*ontology*	1200
processing	*ontology*	165
	Total	3320

Tabla 4.7: Número de resúmenes prueba 3

Los resultados de esta prueba han sido recogidos en la Tabla 4.8. Como se puede observar, ocurrió algo parecido a la segunda prueba, donde el aprendizaje parece no aportar nada a la clasificación.

	Accu ↑	**F1 macro ↑**	**F1 micro ↑**	**Hamm-loss ↓**	**0-1 loss ↓**
BR – zeroR	**0.6128**	0.4579	**0.7281**	**0.3262**	**0.7127**
BPNN	0.5054	0.4795	0.6474	0.4229	0.9231
BR	0.4587	0.4626	0.6097	0.4749	0.9643
CC	0.5233	0.4906	0.6648	0.4022	0.8762
LC	0.5040	0.4894	0.6576	0.4108	0.9391
RakEL	0.4952	**0.4917**	0.6468	0.4364	0.9490

Tabla 4.8: Resultados prueba 3

4.2.9. Cuarta prueba

Mediante las pruebas anteriores se detectó que, si las etiquetas recogían conceptos relacionados entre sí, la labor de aprendizaje se dificultaba enormemente, pues el contenido, seguramente, sería similar. Por ello, se decidió utilizar dos conjuntos de etiquetas con conceptos diferentes: $conjunto1_{p4} = \{education, learning, design\}$ y $conjunto2_{p4} = \{ai, analysis, processing\}$, y realizar búsquedas con las combinaciones entre las etiquetas de cada conjunto. En total, se realizarían $2 \cdot [C(3,2)] = 6$ búsquedas. El número de resúmenes encontrado para cada combinación ha sido recogido en la Tabla 4.9.

Etiqueta 1	**Etiqueta 2**	**Resúmenes**
education	*learning*	359
education	*design*	206
learning	*design*	403
analysis	*ai*	359
analysis	*processing*	212
ai	*processing*	971
	Total	2510

Tabla 4.9: Número de resúmenes prueba 4

Los resultados de esta prueba han sido recogidos en la Tabla 4.10. En este caso, el desempeño de BPNN es mejor que el realizado por BR con zeroR, por lo que se la

labor de aprendizaje, sí parece aportar algo a la clasificación. De hecho, si se observa el valor de las métricas de los otros clasificadores, también están por encima del ofrecido por el *baseline*, exceptuando el de la métrica *0-1 loss*, que tiende a mostrar unos buenos valores debido a su configuración.

	Accu	**F1 macro**	**F1 micro**	**Hamm-loss**	**0-1 loss**
BR − zeroR	0.6389	0.5102	0.7397	0.3471	**0.6259**
BPNN	**0.7269**	0.5935	**0.8264**	**0.2313**	0.7085
BR	0.6982	0.5967	0.8090	0.2558	0.8074
CC	0.6993	0.5919	0.8068	0.2577	0.7093
LC	0.6905	0.5872	0.8007	0.2657	0.7267
RakEL	0.6762	**0.6001**	0.7937	0.2826	0.8234

Tabla 4.10: Resultados prueba 4

4.2.10. Clasificación de una nueva instancia

Se ha querido mencionar, aunque no ha sido implementado en este sistema, cómo se podría clasificar una nueva instancia. En una tarea habitual de clasificación, para clasificar una nueva instancia, es necesario obtener su valor para los atributos que fueron utilizados en el entrenamiento, pues es en base a dichos atributos como el sistema clasifica. Ahora bien, los atributos en este tipo de clasificadores son obtenidos a partir de los textos del entrenamiento, los cuales recogen las palabras más representativas de esa colección de textos. A la hora de clasificar un nuevo texto, no serviría solo con obtener las palabras más representativas de él, pues seguramente no serían exactamente las mismas que los atributos con los que se entrenó el clasificador.

En orden de proponer una forma de solventar este problema, se quiere comentar una forma en la que podría haberse hecho. Una vez se ha preprocesado el texto que se quiere clasificar, para lo que valdría la parte de este sistema diseñada para tal fin, se obtendrían sus propios atributos. Para ello, se calcularía su propia matriz TF-IDF, en la que en realidad no se tendría en cuenta más documentos que él. Una vez obtenidos sus atributos, se procederían a comparar con los obtenidos durante el entrenamiento, aplicándose el siguiente método: si el atributo i coincide con algún atributo de entrenamiento, se asignará su valor como valor del nuevo ejemplo para

dicho atributo de entrenamiento. Si no, su valor será 0. Una vez se tuviera un vector de atributos igual a los que se utilizaron en el entrenamiento, se podría proceder a clasificar el nuevo resumen con el modelo elegido.

El pseudocódigo de este proceso se ha recogido en el Algoritmo 4.

Entrada: nuevo ejemplo a clasificar, fichero de entrenamiento, modelo entrenado.
Salida: clase del nuevo ejemplo.

```
1: Se preprocesa el nuevo ejemplo.
2: Se obtiene la matriz TF-IDF del nuevo ejemplo.
3: for cada atributo de entrenamiento do
4:    for cada atributo del nuevo ejemplo do
5:       if atributo de entrenamiento = atributo del nuevo ejemplo then
6:          escribir el valor del atributo del nuevo ejemplo en su vector de atributos.
7:       else
8:          escribir 0 como valor del atributo del nuevo ejemplo en su vector de atri-
            butos.
9:       end if
10:   end for
11: end for
12: Introducir el nuevo vector con el valor de los atributos del nuevo ejemplo en el
    modelo entrenado.
```

Algoritmo 4: : Clasificación de una nueva instancia

4.3. Conclusiones pruebas de comportamiento

Mediante el desarrollo de las cuatro pruebas de comportamiento, se ha observado la dificultad existente de los clasificadores a la hora de intentar diferenciar entre etiquetas de ámbitos muy parecidos. En un primer momento, se pensó que podía ser debido a que se recogían pocos atributos para poder recoger las diferencias más sutiles, pero como se demostró en la modificación de la prueba 1, no se produjo ninguna mejoría al recoger más atributos. De hecho, sólo se detectó alguna mejoría cuando se separó el ámbito de conocimiento de las etiquetas. De hecho, aunque pudiera haber resúmenes que pertenecieran a ambos ámbitos, no se incluyeron combinaciones que

los mezclaran, a fin de facilitar la labor de aprendizaje.

En cuanto a los clasificadores, la única prueba que puede servir como referencia es la última, donde se detecta una ligera primacía en las métricas del método BPNN sobre las demás.

Capítulo 5

Planificación

La primera reunión con el cliente, en este caso el tutor del proyecto, se produjo el día 24 de Septiembre de 2015. En ella, se plantearon los diferentes temas que podían ser objeto del Trabajo de Fin de Grado, basándose las preferencias personales del autor, enmarcados dentro del tratamiento del lenguaje natural. En dicha reunión, se concluyó que sería interesante que el trabajo consistiera en el desarrollo de un clasificador multietiqueta a partir de textos, lo que lo relacionaba con dicho lenguaje natural. El problema principal consistía en que el autor no tenía conocimiento previo sobre este tipo de clasificación, por lo que los primeros cuatro meses se dedicaron a contextualizar la idea. No obstante, cabe señalar que el tiempo que se pudo dedicar a la búsqueda de información del tema propuesto fue reducido por la finalización de los estudios del autor. Se puede decir, por lo tanto, que el día 21 de Enero de 2016, cuando se produjo la segunda reunión, fue cuando se inicio realmente el proyecto como tal. En cuanto a la fecha límite, se correspondería con la fecha de entrega de proyecto: el 22 de Junio de 2016. La duración total del proyecto, como puede observarse, sería de 154 días. En base al tiempo disponible del autor, se estableció cuántas horas podía dedicarle según una serie de periodos concretos. Esta información ha sido recogida en la Tabla 5.1.

Periodo	**Días**	**Dedicación**	**Horas**
25 de enero - 27 de febrero	34 días	2h / día	68
28 de febrero - 1 de junio	95 días	3h / día	285
2 de junio - 22 de junio	21 días	5h / día	105
		Total	458 horas

Tabla 5.1: Dedicación por período

A continuación, se han recogido las tareas principales detectadas a la hora de desarrollar el presente trabajo.

1. Delimitación de la idea del proyecto y revisión del estado de la cuestión.

2. Fase de Análisis: se deben desarrollar los requisitos, los casos de uso, y todo diagrama que ayude a comprender qué debe poder hacer el sistema.

3. Fase de Diseño: tomando como base la fase anterior, se debe diseñar la arquitectura del sistema.

4. Preparación del entorno de desarrollo: se deben instalar y preparar aquellos elemento software que vayan a ser necesarios para la implementación.

5. Fase de Implementación: una vez completadas las fases anteriores, se puede comenzar a desarrollar el sistema. En esta parte es donde más problemas pueden surgir, pues no se cuenta con experiencia previa utilizando los sistemas elegidos.

6. Evaluación: una vez se desarrolla el sistema, se debe comenzar a probar su correcto funcionamiento y, después, evaluar los modelos generados.

7. Elaboración de la documentación: esta fase puede ser paralelizada respecto a las demás, pues se puede ir generando la memoria a medida que se realizan dichas fases. No obstante, no podrá acabar sin que haya finalizado con anterioridad la última fase.

Tomando como base dichas tareas y las horas por periodo anteriormente referidas, se ha estimado la dedicación por tarea, recogido en la Tabla 5.2. Esto resultará útil para calcular el presupuesto ofrecido en el capítulo 6.

Tarea	**Días**	**Horas/día**	**Horas totales**
Análisis	12 días	2h / día	24 horas
Diseño	7 días	2h / día	14 horas
Preparación entorno	15 días	2h / día	30 horas
Implementación	57 días	3h / día	171 horas
Evaluación	12 días	3h / día	36 horas
Documentación	153 días	variable	183 horas
		Total	458 horas

Tabla 5.2: Dedicación por tarea

Lo anteriormente expuesto se ha recogido en el Diagrama de Gantt, observable en la Figura 5.1, donde se puede observar la referencia temporal de cada una de ellas. Además, también puede servir para establecer una secuenciación, de tal forma que se identifique qué tareas se deben hacer antes qué otras, y cuáles se pueden hacer de manera paralela.

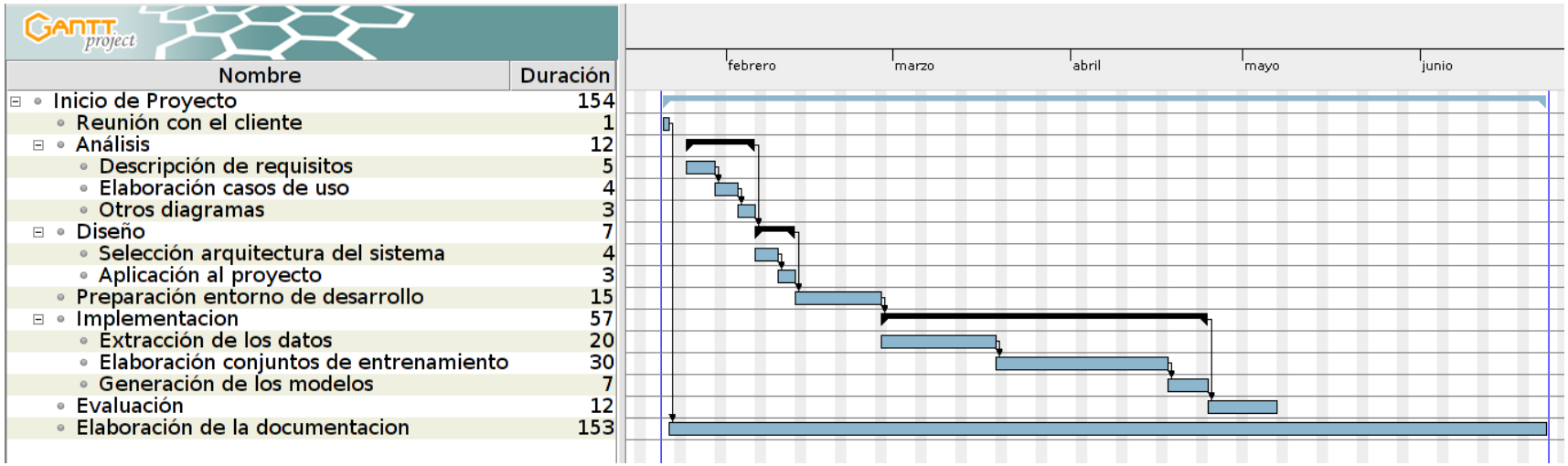

Figura 5.1: Diagrama de Gantt

Capítulo 6

Presupuesto

En esta sección se van a analizar los diferentes aspectos que pueden suponer un coste a la hora de desarrollar este proyecto. De esta manera, se podrá elaborar un presupuesto que sirva como base para estimar su casote. Estos costes se han decidido dividir en las siguientes categorías:

1. Salario del personal: se corresponde con el sueldo que habrían percibido las personas involucradas en el proyecto en función de las horas dedicadas.

2. Recursos hardware: representa los equipos informáticos necesarios para desarrollar el proyecto.

3. Recursos software: representa los programas necesarios para desarrollar el proyecto.

4. Otros costes, como material fungible, viajes y dietas, y costes indirectos: hacen referencia, por orden, a aquel material que se desgasta por el uso; a los costes derivados de los desplazamientos y la alimentación; y aquellos costes derivados de desarrollar el proyecto, como la luz.

La cuantificación de estos costes se ha intentado realizar lo más fiel a la realidad posible.

6.1. Salario del personal

En este proyecto, solo ha habido dos personas involucradas: el tutor, que ha servido de apoyo y guía, y el autor, que ha desarrollado el sistema representado diferentes roles según la situación lo requiriera. En la Tabla 6.1 se puede observar cómo se ha dividido el salario de este último a fin de representar las diferentes facetas de desarrollo.

Faceta	**Total horas**	**Coste/hora**	**Coste Total**
Analista	24 horas	33€/hora	792 €
Diseñador	14 horas	33 €/hora	462 €
Desarrollador	201 horas	25€/hora	5.025€
Pruebas	36 horas	20€/hora	720 €
Documentación	183 horas	15€/hora	2.745€
Total	458 horas		9.744 €

Tabla 6.1: Salario del personal

6.2. Recursos hardware

Para el desarrollo de este proyecto se ha necesitado lo recogido en la Tabla 6.2. Sin embargo, dado que el proyecto no abarca toda la vida útil de dichos recursos hardware, se ha calculado el coste imputable al mismo, recogido en la Tabla 6.3 En dicha Tabla, se han incluido varios campos para calcular el coste imputable al proyecto, los cuales son:

- Coste: precio de mercado.
- % de uso: porcentaje de uso del recurso en la realización de este proyecto.
- Dedicación: número de meses en los que se ha utilizado un recurso determinado. Para ello, se ha utilizado la duración del proyecto expresada en la planificación, es decir, 154 días (5,13 meses).

- Período de depreciación: para el cálculo de este período, se ha utilizado la garantía ofrecida por el fabricante. En caso de que no fuera ofrecida ninguna garantía, se utilizará el período que se cree que durará el bien en perfecto estado.

- Coste imputable: se calcula como la fracción del coste en base a los valores anteriores. Se ha incluido cómo calcularlo en la Ecuación 6.1.

$$Coste\text{-}imputable = \frac{coste \times \ \%\ de\ uso \times dedicación}{periodo\ depreciación} \tag{6.1}$$

Concepto	**Coste**
RH 1 - Toshiba P50-A-12Z Intel i7-4700MQ/8GB/1TB/GT745M/15.6”	917,00 €
RH 2 - Microsoft Wireless Mobile Mouse 1850, nano transceptor plug-and-go	13,99 €
RH 3 - Logitech Keyboard K120	13,95 €
Total	**944,94**

Tabla 6.2: Recursos hardware - coste total

Concepto	**Coste**	**% de uso**	**Dedicación**	**Periodo depreciación**	**Coste imputable**
RH 1	917,00 €	100	5,13 meses	24 meses	196 €
RH 2	13,99 €	100	5,13 meses	18 meses	3,99 €
RH 3	13,95 €	100	5,13 meses	24 meses	2,98 €
Total	**944,94 €**				**202,97 €**

Tabla 6.3: Recursos hardware - coste imputable

6.3. Recursos software

Para el desarrollo de este proyecto, han sido necesarios varios programas, los cuales se van a especificar aquí. Sin embargo, no ha sido necesario ningún desembolso, pues cuentan con sofrware libre o ha sido posible su acceso gracias a acuerdos con la Universidad Carlos III de Madrid.

Concepto	**Precio**
Ubuntu 14.04 LTS	0,00 €
Eclipse Version: Mars.1 Release (4.5.1)	0,00 €
Python: scikit-learn + nltk	0,00 €
Meka: A Multi-label Extension to WEKA	0,00 €
Gantt Project	0,00 €
Búsqueda de documentos	Licencia UC3M - 0,00 €
LaTeX	0,00 €
Gantt Project - 0€	0,00 €
Total	**0,00 €**

Tabla 6.4: Recursos software

6.4. Otros costes

Material fungible

Concepto	**Precio**
Cuaderno Croquera 16x21 cms, papel Bond 80 grs, 100 hojas	1,50 €
Bolígrafo Bic Mediano Cristal	0,80€
Total	**2,30 €**

Tabla 6.5: Material fungible

Viajes y dietas

En esta sección se han decidido imputar los costes derivados por desplazamiento para las reuniones con el tutor. El gasto en dietas no se ha decidido incluir por no ser un gasto directamente imputable al proyecto.

Concepto	**Precio por viaje**	**Nº de viajes**	**Total**
Viajes	6 €	4	24 €

Tabla 6.6: Viajes y dietas

Costes indirectos

Como costes indirectos, tales como la luz o internet, se ha decidido imputar un 15 % de los costes directos.

6.5. Resumen de costes

Los costes directos se pueden observar en la Tabla 6.7.

Coste directos	
Salario	9.744 €
Recursos hardware	202,97 €
Recursos software	0 €
Otros costes	26,30 €
Total	**9973,27 €**

Tabla 6.7: Resumen costes directos

Los costes indirectos, por su parte, se han calculado según lo expresado en la Ecuación 6.2.

$$costes\ indirectos = costes\ directos \times 15\,\% = 9973,27€ \times 15\,\% = 1495,99€ \quad (6.2)$$

Por lo tanto, los costes totales del proyecto se han recogido en la Tabla 6.8.

Coste totales	
Costes directos	9973,27 €
Costes indirectos	1495,99 €
Total	**11469,26 €**

Tabla 6.8: Costes totales

Capítulo 7

Conclusiones

Se considera que este trabajo sirve como acercamiento a cualquier problema de clasificación multietiqueta. Mediante su elaboración, se ha podido detectar el vasto conocimiento existente sobre él, del cual se ha podido recoger solo una parte. Son numerosos los trabajos realizados poniendo en práctica los métodos aquí mencionados, pero también hay muchos otros donde se realizan variaciones sobre ellos, de tal forma que se logre una mejora en su desempeño, o se adapten a un problema en particular. Cuando se estudiaron las métricas de evaluación, se encontró cuánta variedad se proponía para poder evaluar un sistema, atendiendo a cuál fuera su naturaleza o a qué métodos se utilizaran. Incluso, cuando se utilizó Meka, se descubrieron otras muchas técnicas que podrían servir. Con esta mención se quiere hacer referencia a la pequeña parte que este trabajo ha podido abarcar, pero que supone el primer paso para conocer la variante multietiqueta de un problema de clasificación.

Algo parecido ocurre con el *text mining*, pues en este trabajo sólo se ha aplicado una técnica que se centra en la frecuencia de las palabras en los textos. Sin embargo, durante la investigación, se detectó cómo se han desarrollado otras formas de analizar el lenguaje, teniendo en cuenta, por ejemplo, dónde están situadas las palabras en una frase o qué palabras acompañan a otras, algo mucho más complejo.

La dificultad asociada a ello se reveló cuando se desarrolló el sistema y se fueron encontrando los problemas que se mencionaban en la literatura, como la imposibilidad de tratar con demasiados atributos o la atención que requería la elección de una buena base de datos. A medida que se llevaron a cabo las pruebas, se mani-

festó cómo la labor de aprendizaje no era tan sencilla como podía parecer, sino que se debían localizar correctamente los ejemplos de entrenamiento. Como se comentó, si los ejemplos trataban sobre temas muy relacionados, el clasificador no tenía las herramientas suficientes para diferenciarlos.

No obstante, se ha logrado desarrollar el software mencionado en los objetivos, con algunas limitaciones que se comentarán a continuación, pasando por todas las fases necesarias, y se ha logrado presentar un conocimiento básico para enfrentar cualquier problema multietiqueta, sobre todo aquellos que se basen en el análisis de textos.

7.1. Líneas futuras de investigación y limitaciones

En esta sección se han querido tratar dos asuntos: las limitaciones del sistema desarrollado y las líneas futuras de investigación detectadas, es decir, qué aspectos, en base a lo desarrollado, podría ser interesante estudiar como continuación de este TFG. Para finalizar, se ha incluido un repaso sobre una serie de aspectos que se deberían haber tenido en cuenta a la hora de desarrollar el sistema, y que podrían servir de guía si alguien decidiera utilizar alguna de las plataformas aquí mencionadas, como *BibSonomy* o Meka.

7.1.1. Limitaciones del sistema

A continuación, se van a mencionar los límites del sistema desarrollado. Dichos límites, podrían ser solventados si se dispusiera de más tiempo y conocimientos.

- El sistema está implementado de tal forma que solo se pueden realizar búsquedas por dos etiquetas. Se hizo así pensando que sería aumentado a más, pero al descubrir que en *BibSonomy* se reducía el número de resúmenes a medida que aumentaba el número de etiquetas, se decidió que no sería prioridad el buscar por más de dos. Sin embargo, hubiera sido interesante poder realizar pruebas con búsquedas de varias etiquetas para ver qué tal respondía el sistema.

- Para generar los archivos de entrenamiento se deben utilizar textos en inglés.

- La evaluación de los modelos está restringida a las métricas que ofrece Meka en su interfaz gráfica.

- El sistema no está preparado para clasificar nuevos ejemplos, aspecto que ya se ha mencionado cómo podría resolverse.

7.1.2. Líneas futuras de investigación

El objetivo de esta sección consiste en recoger aquellos aspectos detectados durante la elaboración de este trabajo que podría ser interesante seguir investigando. En primer lugar, se quiere mencionar que todas las limitaciones expresadas en la Sección 7.1.1 marcarían el punto de partida para enriquecer el sistema. Sin embargo, tratando de mencionar temas que no han sido explorados en este trabajo por exceder su ámbito de actuación, se han encontrado los siguientes:

- Modelos y métricas relacionados con *Label Ranking*. En este trabajo, solo se buscaba desarrollar un sistema que fuera capaz de diferenciar entre una serie de etiquetas que podrían ser atribuidas a un resumen y las que no. Sin embargo, a medida que se estudió el estado de la cuestión, se fueron detectando menciones a este otro tipo de clasificación, donde el objetivo consiste en generar una jerarquía ordenada de las etiquetas.

- Generar un sistema de recomendación de etiquetas. En el sistema desarrollado, se han obtenido una serie de palabras que podrían servir para identificar a una etiqueta en concreto. Si se estableciera algún tipo de mecanismo que comparara los atributos de una etiqueta y de otra, podría establecerse un grado de similitud entre ellas. Si ese grado de similitud superara un determinado umbral, podría considerarse que están relacionadas, siendo interesante recomendar una cuando se realizaran búsquedas por la otra y viceversa. De hecho, *BibSonomy* hace algo parecido, pues al buscar por un *tag*, se puede visualizar una serie de *related tags*, cuya generación se produce basándose en etiquetas que suelen aparecer juntas. Podría ser interesante comparar los dos métodos mencionados, además de aportar algo al carácter social que promueven desde la página, pudiendo recomendar etiquetas a los usuarios.

- Profundizar más en los métodos mencionados en la Sección 2.3.2 Adaptación de Algoritmos, pues habría sido interesante investigar qué modificaciones se han propuesto sobre algoritmos originales para tratar con la clasificación multietiqueta.

7.1.3. Aspectos a tener en cuenta

A medida que se iba desarrollando el sistema, se detectaron algunas decisiones que no habrían sido tomadas así si se hubiera sabido las consecuencias que provocaron. Se ha considerado útil mencionarlas aquí, pues suponen un ejemplo de factores a tener en cuenta si se decidiera utilizar alguno de los sistemas aquí descritos.

BibSonomy. A la hora de elegir qué base de datos utilizar para obtener las instancias de entrenamiento, se realizó una búsqueda de posibles páginas que cumplieran con los requisitos establecidos. Sin embargo, no se encontraron muchas que los cumplieran. Finalmente, se decidió utilizar *BibSonomy* para así poder comenzar con el proyecto, pues en un principio parecía ser suficiente para ello. Sin embargo, hubo algunos aspectos que no se tuvieron en cuenta:

- La mayoría los artículos con los que cuenta son en alemán, de hecho, el grupo de desarrollo de la página está formado en su mayoría por personas alemanas. Por consiguiente, su base de datos en inglés es reducida. En un primer acercamiento, se consideró que el número de ejemplos sería suficiente, pero al observar los resultados de las pruebas, se cree que hubiera sido interesante tener más, para así enriquecer la labor de aprendizaje. Además, hay un número pequeño de etiquetas que cuentan con un número aceptables de resúmenes, de tal forma que la generación de diferentes conjuntos de entrenamiento estuvo muy limitada.

- Selección de las etiquetas: a la hora de añadir una referencia en *BibSonomy*, el usuario debe especificar qué etiquetas cree que explican el contenido. El problema se deriva de que los usuarios tienden agregar etiquetas que no recogen realmente de qué trata, sino referentes a, por ejemplo, la editorial (*Springer*), si es un libro (*book*) o si la realización ha sido propia (*myown*). Este último, por ejemplo, tiene 16043 resultados. De esta forma, al depender la veracidad de

las etiquetas de la decisión de cada usuario, se cree que la labor de aprendizaje se ha visto dificultada.

- Etiquetas compuestas: cuando un usuario especifica qué etiquetas cree que definen un artículo que está introduciendo, debe separarlas por espacios. Si el *tag* es compuesto o trata de dos conceptos que suelen aparecer unidos, el usuario debe decidir cómo unirlas. Por ejemplo, si quisiera expresar que un artículo trata sobre *Artificial intelligence*, podría ponerlo como: *artificial_intelligence* o *artificial-intelligence*. Eso podría no ser un problema si, al realizar una búsqueda, *Bibsonomy* no diferenciara en este tipo de cosas, pero los considera como etiquetas diferentes.

Diseño del sistema en varias plataformas. Cuando se comenzó a desarrollar el sistema, algunas herramientas obligaban el uso de determinada plataforma o lenguaje. Por ejemplo, la API de *BibSonomy* es una librería que está escrita en Java, y NLTK está escrita en Python. Sin embargo, Meka se decidió utilizar a través de su interfaz gráfica porque se consideraba que la curva de aprendizaje era demasiado pronunciada. No obstante, Meka puede tratar directamente con *scikit-learn*, como así se especifica en su página web. La ventaja principal de haber implementado Meka en el código de Python se habría encontrado a la hora de evaluar los diferentes modelos, pues se habría tenido mucha más flexibilidad. Por ejemplo, se podría haber accedido a los componentes de la Matriz de Confusión por separado. Se cree que esto podría haber enriquecido la evaluación.

Apéndice A

Resumen de contenidos en inglés

A.1. Abstract

The present project has been developed in order to finish the Dual Degree in Informatics Engineering and Business Administration. This document explains the process involved in developing a software that automatically generate the main words needed to label a scientific article from its abstract. To that end, it has been necessary to know the state of art, that is to say, to exactly define a multi-label problem and which techniques exists to resolve it. Furthermore, it has been necessary to know how a text can be transformed so we can extract some attributes that explain its content.

Once this has been done, the next phases can start: analysis, design and implementation. The mentioned phases has been appropriately documented and tested in the evaluation. For that, four modules have been generated: information extraction, training sets generation, model construction and evaluation.

This project ends with a chapter dedicated to the conclusions, where an overview of the relevant aspects noticed during the development was made. In addition to that, the limitations and the future research activities based on this project were mentioned.

Keywords: multi-label classification, machine learning, text mining

A.2. Introduction

Nowadays, although many people don't notice it, machine learning is present in many aspects of our lives. Examples include automatic spam detection in email accounts or product recommendation based on individual preferences in a web. Although this may seem easy, because we see it very often, it is not: there is a process, which includes complex algorithms and years of development in order to achieve it.

This process is even more complicated when we deal with natural language: for a human, it can be easy to read a document and to say a few words to define it. However, there is a complete field of study when it comes to achieving an automatic generation. In fact, it will be common to establish several words as keywords, like those used in this project. This trivial appreciation has meant the development of a new area of knowledge, in which this multi-label appreciation has been considered. Precisely, this project focus on that: automatic multi-label text processing.

A.2.1. Motivation

The main reason for choosing this topic is to investigate the multi-label text classification. In addition to that, it is interesting to move from investigation to development, in other words, to move from knowing how a problem can be solved, to actually resolve it.

On the other hand, this project is considered as a starting point for all multi-label problems, their specific area does not matter. Although the topic of this project is based on scientific texts, this knowledge can be applied in other multi-label problems. In the following examples, the multi-label knowledge had been used to:

- Image classification and recognition patterns. For example, an image showing a mountain and a beach at the same time (Boutell et al., 2004), (Li et al., 2011).
- Medical applications: multi-label learning for detecting more that one syndrome (Liu et al., 2012) or the study of the genome (Clare y King, 2001).
- Others applications: relation between music and emotions (Li y Ogihara, 2003) or dispute resolutions projects forecasting (Chou, 2012).

A.2.2. Objectives

The aim of this project is to design, implement and evaluate a software that automatically generate the main words needed to label a scientific article from its abstract. For that matter, machine learning techniques will be used, specifically, multi-label classification techniques.

Apart from this main objective, a few secondary objectives have been stated to be achieved in this project:

1. To research the peculiarities of multi-label classification: from classification methods to evaluation metrics.

2. To learn the different existing technologies for developing this type of classification.

3. To be familiar with the natural language in machine learning tasks.

4. To develop an entire project from start to finish, based on an own idea, understanding all the stages involved in order to achieve it.

A.2.3. Structure

The present document is divided in the following sections:

Chapter 1: Introduction. The aim of this chapter is to establish a preamble explaining the choice of the topic and the motivations for doing it.

Chapter 2: State of art. To develop the mentioned software, it is necessary to understand how we can do it. In this chapter there is an analysis of the techniques and evaluation metrics used for multi-label classification, a review of text mining and a reference to the used technologies.

Chapter 3: Description of the work. The phases for developing this project, based on a certain methodology, are explained in this chapter. These phases are: analysis, design and implementation.

Chapter 4: Evaluation. Once the software has been developed, it is time to test it. This can be done by tests to see the correct system performance and behavioral tests.

Chapter 5: Planning. In this section the temporal plan for developing the project is shown.

Chapter 6: Budget. In order to place this project in the actual socio-economic context, a detailed budget has been included.

Chapter 7: Conclusions. For closing this project, a final section with conclusions has been included. Also, the limitations and future research lines are mentioned.

Chapter 8: Bibliography. Several sources were consulted in order to complete this work. These sources have been referenced in the text and collected in this section.

A.3. System architecture

The system architecture most suited for the project was based on modules. In this case, four modules were developed, as can be seen in the Figura A.1. The mentioned modules are

1. Module 1: information extraction. This module communicates with the web BibSonomy through an API, this is implemented in Java. The output of this module consists in two files: one with the abstract and one with the label for each abstract.

2. Module 2: text processing and feature extraction, this can be done by NLTK and Scikit-learn, both written in Python. The output of this module is the TF-IDF Matrix.

   Module 2.1.: training file generation. In this module, the training file is composed taking into account whichever software for training is gonna be used. In this case, it is Meka.

3. Module 3: construction of the models and training.

4. Module 4: evaluation. In this module the evaluation metrics are analysed.

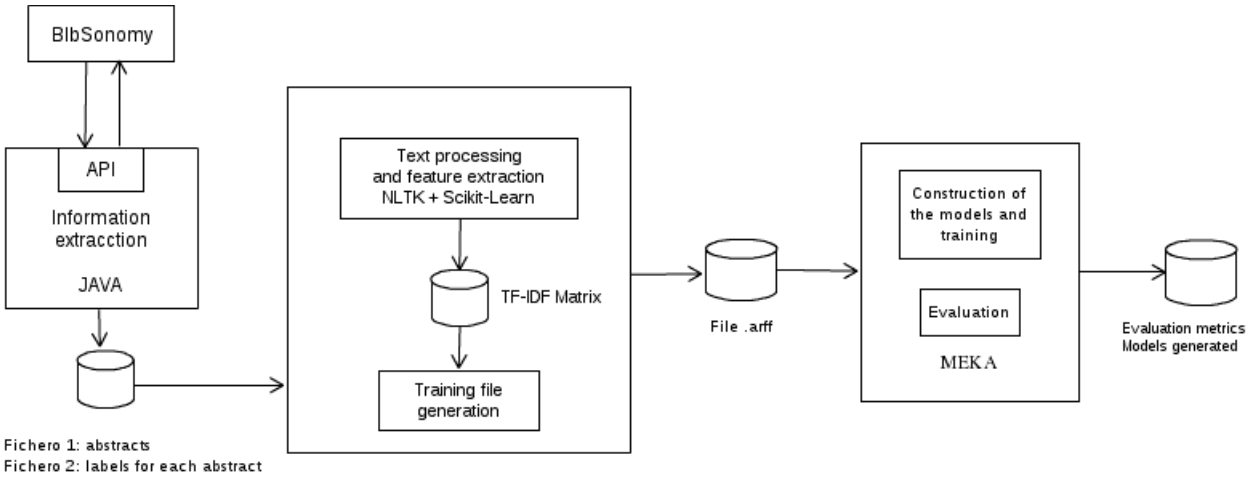

Figura A.1: Representation of the system architecture

A.4. Evaluation summary

Referring to the evaluation, four behavioral test have been developed, but only the last one provides interesting results. The first one has also been included to show the bad performance at the beginning of the tests.

The labels used on the first test have been taken up in the Tabla A.1. Also, the labels used on the fourth test have been taken up in the Tabla A.2.

Label 1	**Label 2**	**Abstracts**
analysis	*ai*	359
analysis	*processing*	212
analysis	*recognition*	264
processing	*ai*	971
processing	*recognition*	211
ai	*recognition*	367
	Total	2384

Tabla A.1: Number of abstracts for test 1

Label 1	**Label 2**	**Abstracts**
education	*learning*	359
education	*design*	206
learning	*design*	403
analysis	*ai*	359
analysis	*processing*	212
ai	*processing*	971
	Total	2510

Tabla A.2: Number of abstracts for test 4

The techniques used for classification were: BR (*Binary Relevance*), LP (*Label Power Set*), ECC (*Classifier Chains*), RAkEl (*Random k-LabEl Pruned Sets*) and BPNN (*Back Propagation Neural Network*). Also, a base line was included, and it constitutes a starting point for comparisons. The base line uses an algorithm that gives the majority class to each instance, generated by binary relevance with zeroR. The evaluation metrics used were: *accuracy* (accu), *hamming-loss* (hamm-loss), *F1, micro y macro* and 0-1 *loss*.

The results for the first test can be seen in Tabla A.3. In that test, the base line holds the best results, which mean that the training labor is not working well. The results for the fourth test can be seen in the Tabla A.4. As shown in the mentioned Tabla, the BPNN holds the bests results. The other test served to conclude the

following: if the labels were meant to relate two things (like artificial intelligence and analysis for example), the learning labor was more difficult.

	Accu ↑	**F1 macro ↑**	**F1 micro ↑**	**Hamm-loss ↓**	**0-1 loss ↓**
BR – zeroR	**0.5729**	0.3421	**0.6525**	**0.3475**	**0.5864**
BPNN	0.4617	0.3692	0.5528	0.4467	0.7266
BR	0.4520	0.3605	0.5540	0.4515	0.7703
CC	0.4782	0.3744	0.5812	0.4188	0.7279
LC	0.4805	0.3932	0.5965	0.4035	0.7516
RakEL	0.4743	**0.3948**	0.5871	0.4199	0.7527

Tabla A.3: Results for test 1

	Accu ↑	**F1 macro ↑**	**F1 micro ↑**	**Hamm-loss ↓**	**0-1 loss ↓**
BR – zeroR	0.6389	0.5102	0.7397	0.3471	**0.6259**
BPNN	**0.7269**	0.5935	**0.8264**	**0.2313**	0.7085
BR	0.6982	0.5967	0.8090	0.2558	0.8074
CC	0.6993	0.5919	0.8068	0.2577	0.7093
LC	0.6905	0.5872	0.8007	0.2657	0.7267
RakEL	0.6762	**0.6001**	0.7937	0.2826	0.8234

Tabla A.4: Results for test 4

A.5. Budget summary

In this section, the cost of the project is shown. The direct cost can be seen in Tabla A.5. The indirect costs can be calculated following ecuation: A.1. Finally, the total cost of the project is shown in Tabla A.6.

Direct costs	
Salaries	9.744 €
Hardware resources	202,97 €
Software resources	0 €
Other	26,30 €
Total	**9973,27 €**

Tabla A.5: Direct cost summary

$$\textit{indirect costs} = \textit{direct costs} \times 15\,\% = 9973,27€ \times 15\,\% = 1495,99€ \quad \text{(A.1)}$$

Total costs	
Direct costs	9973,27 €
Indirect costs	1495,99 €
Total	**11469,26 €**

Tabla A.6: Total costs

A.6. Planning summary

This project started on January 21, 2016, and were finished on June 22, 2016. Its total duration being 154 days. In this period of time, the following tasks were developed:

1. Definition of the project idea and review of the state of art.
2. Analysis phase. In this phase requirements, use cases, and other diagrams were developed.
3. Design phase: design of the architecture of the system.

4. Preparation of the development enviroment.
5. Implementation phase.
6. Evaluation.
7. Documentation: in this phase the present document have been developed, with other tasks being able to be completed at the same time.

The total hours for these tasks are shown in Tabla A.7

Task	**Days**	**Hours/day**	**Horas totales**
Analysis	12 days	2h / day	24 hours
Design	7 days	2h / day	14 hours
Development environment	15 days	2h / day	30 hours
Implementation	57 days	3h / day	171 hours
Evaluation	12 days	3h / day	36 hours
Documentation	153 days	variable	183 hours
		Total	458 hours

Tabla A.7: Hours per task

A.7. Conclusions

This work is considered to be an approach to any multi-label classification problem. Thank to the process of doing this project, a vast knowledge of the matter has been found, although only a small part of it has been recollected in this project. There are many articles that use the methods presented in this work, but there are also many others which modify something about them, in order to achieve a better performance or an adaptation to a concrete problem. When evaluation metrics were studied, it was found how many of them exists to evaluate a system, taking into account their nature or whichever methods were used. Even when Meka was used, other techniques that could have been used where found. This is intended to say that

this work did not cover all the multi-label field, but it is the first step to understand it.

There is a similar situation with text mining, because in this work only one technique were used, focused on the word frequency in texts. However, during the investigation, other ways to analyse the language were found. For example, they focus on where the words are in a sentence or what words are near what other words, something much more complex.

The difficulty associated to it was revealed when the system were developed and the problems mentioned in the literature were founded, like the impossibility to deal with too many attributes or the attention that the election of a good database is required. As tests were conducted, we could saw how the work of learning was not as simple as it seemed: training examples should be located property. As commented before, if training examples were very related, the classifier had not got the tools needed in order to differentiate them.

However, the software mentioned in the objectives, with some limitations that we comment below, was developed, and it presents the basic knowledge for dealing with a multi-label problem, in particular the problems based on text mining.

A.7.1. Future research lines and limitations of the system

In this section, two issues are presented: the constrains of the system and the future research lines detected.

Limitations of the system

And now, the limitations of the development system are introduced:

- The system only support searches consisting in two labels. At first, it was considered to increase that amount, but, in BibSonomy, when you search using more than two labels, only a few abstracts are shown.
- For generating training files, only English texts can be used.
- The model evaluation is restricted to the evaluation metrics that Meka offers in their graphical interface.

- The system is not prepared for classifying new examples.

Future research lines

The aim of this section is to show interesting issues that we can research in order to continue with this work. First of all, the mentioned limitations would be the priority.

- Models and evaluation metrics related to Label Ranking. In this project, we only want to develop a system capable of differentiating between two label sets: related to a certain abstract and not related to it. But there is another way to do it: the output of the classifier can be making a ranking of labels of all classes.
- Developing a label recommendation system. Based on a certain label, the system can be modified to recommend related labels based on its similarity.
- Investigating the methods mentioned on Section 2.3.2 Algorithm Adaptation.

Bibliografía

BibSonomy. *The blue social bookmark and publication sharing system.* URL www.bibSonomy.org. Fecha de consulta: 14-04-2016.

Matthew R Boutell, Jiebo Luo, Xipeng Shen, y Christopher M Brown. Learning multi-label scene classification. *Pattern recognition*, 37(9):1757–1771, 2004.

E. Alvares Cherman, M. Carolina Monard, y Jean Metz. Multi-label problem transformation methods: a case study. *CLEI ELECTRONIC JOURNAL Vol. 14, Number 1, Paper 4*, 2011. Pages 1-13.

Jui-Sheng Chou. Comparison of multilabel classification models to forecast project dispute resolutions. *Expert Systems with Applications*, 39(11):10202 – 10211, 2012. ISSN 0957-4174.

Amanda Clare y Ross D King. Knowledge discovery in multi-label phenotype data. En *European Conference on Principles of Data Mining and Knowledge Discovery*, págs. 42–53. Springer, 2001.

Rick Kazman; Len Bass; Paul Clements. *Software Architecture in Practice*. Addison-Wesley Professional, 2003. ISBN 0-321-15495-9, 978-0-321-15495-8.

Alistair Cockburn. Writing effective use cases. 1999.

Corinna Cortes y Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Laboratorio Nacional de Calidad del Software de INTECO. Ingeniería del software: metodologías y ciclos de vida. 2009.

Pedro Domingos y Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2):103–130, 1997. ISSN 1573-0565.

Eclipse. *The Eclipse Fundation open source community website*. URL https://eclipse.org/. Fecha de consulta: 14-04-2016.

W. Fan y A. Bifet. Mining big data: Current status, and forecast to the future. *SIGKDD Explorations*, 2012. Volume 14, Issue 2.

Ronen Feldman y James Sanger. *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge University Press, 2007.

Ioannis Katakis Grigorios Tsoumakas y Ioannis Vlahavas. *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. ISBN 0387244352, 9780387244358.

Chih-Wei Hsu y Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, 2002.

Chao-Ming Hwang y Miin-Shen Yang. New similarity measures between generalized trapezoidal fuzzy numbers using the jaccard index. *International Journal of Uncertainty, Fuzziness and Knowledge Based Systems*, 22(6):831 – 844, 2014. ISSN 02184885.

IEEE. IEEE Recommended Practice for Software Requirements Specifications. Inf. téc., 1998.

Tao Li y Mitsunori Ogihara. Detecting emotion in music. En *ISMIR*, tomo 3, págs. 239–240. 2003.

Teng Li, Shuicheng Yan, Tao Mei, Xian-Sheng Hua, y In-So Kweon. Image decomposition with multilabel context: Algorithms and applications. *IEEE Transactions on Image Processing*, 20(8):2301–2314, 2011.

Guo-Ping Liu, Jian-Jun Yan, Yi-Qin Wang, Jing-Jing Fu, Zhao-Xia Xu, Rui Guo, y Peng Qian. Application of multilabel learning using the relevant feature for each label in chronic gastritis syndrome diagnosis. *Evidence-based complementary and alternative medicine : eCAM*, 2012:135387–9, 2012.

Geoffrey Holmes Bernhard Pfahringer Peter Reutemann Ian H. Witte Mark Hall, Eibe Frank. The weka data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

J. McCarthy, M. L. Minsky, N. Rochester, y C.E. Shannon. A proposal for the Dartmouth summer research project on artificial intelligence. Disponible en: http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html [Fecha de consulta: 17 de Abril de 2016].

D. Borrajo Millán, J. González Boticario, y P. Isasi Viñuela. Aprendizaje Automático. 2006. ISBN 84-96094-73-1.

G. Nasierding y A. Z. Kouzani. Comparative evaluation of multi-label classification methods. En *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, págs. 679–683. 2012a.

Gulisong Nasierding y Abbas Z. Kouzani. Comparative evaluation of multi-label classification methods. *9th International Conference on Fuzzy Systems and Knowledge Discovery*, 2012b. Pages 679 - 683.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, y E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

J. Read, P.Reutemann, B. Pfahringer, y G. Holmes. Meka: A multi-label/multi-target extension to weka. *Journal of Machine Learning Research*, 17(21):1–5, 2016. URL http://jmlr.org/papers/v17/12-164.html.

Jesse Read, Bernhard Pfahringer, Geoff Holmes, y Eibe Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, 2011. ISSN 1573-0565.

Mário Rodrigues, António Teixeira, y SpringerLink (Online service). *Advanced Applications of Natural Language Processing for Performing Information Extraction*. Springer International Publishing, 2015.

Stuart J. Russell y Peter Norving. Artificial Intelligence: A Modern Approach. 1995.

Gerard Salton y Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986. ISBN 0070544840.

Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

Mohammad S Sorower. A literature survey on algorithms for multi-label learning. 2010.

Enrique J. Carmona Suárez. Tutorial sobre máquinas de vectores soportes (svm). 2014.

Richard S Sutton y Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

Grigorios Tsoumakas y Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM) 3*, 2007. Pages 1-13.

Grigorios Tsoumakas, Eleftherios Spyromitros-Xioufis, Jozef Vilcek, y Ioannis Vlahavas. Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011.

Grigorios Tsoumakas y Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. En *Machine learning: ECML 2007*, págs. 406–417. Springer, 2007.

A.M. Turing. Computing machinery and intelligence. *Mind, New Series, Vol. 59, No. 236*, págs. 433–460, 1950. Disponible en: http://www.csee.umbc.edu/courses/471/papers/turing.pdf [Fecha de consulta: 17 de Abril de 2016].

Vincent Van Asch. Macro-and micro-averaged evaluation measures. 2013.

Carlos Zapata y Gilma Liliana Garcés. Generación del diagrama de secuencias de uml 2.1. 1 desde esquemas preconceptuales. *Revista EIA*, (10), 2008.