TRABAJO FIN DE GRADO

INGENIERÍA EN INFORMÁTICA

# Sistema de Clasificación Automática de Iniciativas al Parlamento Andaluz

---


**Autor**

José Arcos Aneas

**Directores**

Luis Miguel de Campos Ibañéz

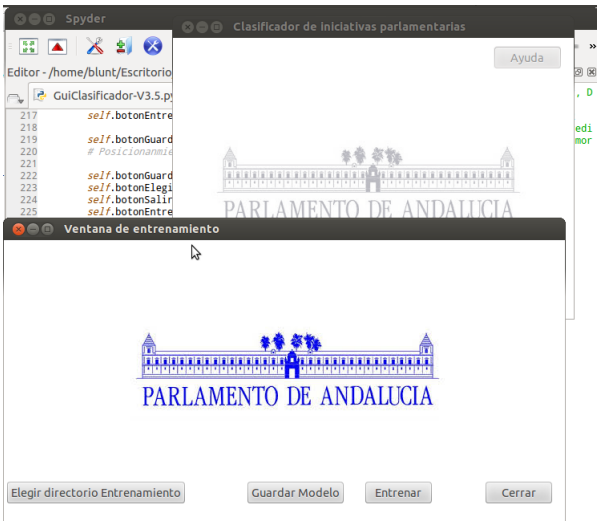
Juan Francisco Huete Guadix




Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación


---

Granada, 4 de Septiembre de 2015




# Sistema de Clasificación Automática de Iniciativas al Parlamento Andaluz

**Clasificador automático de texto.**

**Autor**
Jose Arcos Aneas

**Directores**
Luis Miguel de Campos Ibañéz
Juan Francisco Huete Guadix

DECSAI
Universidad de Granada

Departamento de Computación y Sistemas Inteligentes

# Sistema de Clasificación Automática de Iniciativas al Parlamento Andaluz: Clasificador automático de texto

José Arcos Aneas

**Palabras clave**: Clasificación, multilabel, texto, SVM, RandomForest, Naive Bayes, PageRank,Porter

## Resumen

El objetivo final de este trabajo es la de realizar un sistema automático de ayuda a la decisión en la clasificación de iniciativas parlamentarias al Parlamento Andaluz.

Para esto estudiaremos el proceso de la clasificación automática aplicada a textos, lo que se conoce como *clasificación documental*. Desarrollaremos una serie de clasificadores a partir de los algoritmos y técnicas más comunes para llevar a cabo este tipo de tareas, como son; el algoritmo de clasificación de Naive Bayes, el algoritmo RandomForest y por último las Máquinas de Vectores Soporte (SVM). Evaluaremos el rendimiento de estos clasificadores con el objetivo de determinar cuál de ellos es mas apropiado para esta tarea. Por último, diseñaré y desarrollaré una pequeña aplicación con la que poder sea posible clasificar documentos.

# Automatic Classification System of the Andalusian Parliament Initiatives: Auto Classifier text

José Arcos Aneas

**Keywords**:Classification, multilabel, text, SVM, RandomForest, Naive Bayes, PageRank, Porter

**Abstract**

The ultimate goal of this work is to perform a automatic system of decision support in sorting parliamentary initiatives for Andalusian Parliament.

To carry out this work, first, we will study the classification process focused on text files, and then develop a series of classifiers based on machine learning algorithms currently popular, such as; algorithm Naive Bayes, algorithm Random Forest and finally the Support Vector Machines (SVM). We evaluate the performance of these classifiers in order to determine which one is more suitable for use in developing an application for text classification. Finally, it will be designed and a small application that is possible to classify documents will be developed. The final application should allow classifying new documents from previously classified documents.

Yo, **Jose Arcos Aneas**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada: Clasificador automático de texto**, con DNI 74740565-H, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: José Arcos Aneas

Granada a 4 de Septiembre de 2015.

D. **Luis Miguel de Campos Ibañéz**, Profesor del Área de Ciencias de la Computación del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

D. **Juan Francisco Huete Guadix**, Profesor del Área de Ciencias de la Computación del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

**Informan:**

Que el presente trabajo, titulado *Sistema de Clasificación Automática de Iniciativas al Parlamento Andaluz: Clasificador automático de texto*, ha sido realizado bajo su supervisión por **José Arcos Aneas**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 4 de Septiembre de 2015 .

**Los directores:**

**Luis Miguel de Campos Ibañéz** **Juan Francisco Huete Guadix**

# Agradecimientos

Primero y como más importante, me gustaría agradecer sinceramente a mis tutores del Trabajo de Fin de Grado su esfuerzo y dedicación. Sus conocimientos, sus orientaciones, su manera de trabajar, su motivación y sobre todo su paciencia han sido fundamentales para poder llevar a cabo este trabajo.

También me gustaría agradecer a mi familia y amigos su apoyo constante e incondicional en toda mi vida y más aún en mis momentos más difíciles.

# Índice general

# Índice de figuras

# Índice de cuadros

# Introducción y objetivos del trabajo

Las organizaciones tienden a clasificar sus documentos de acuerdo a su contenido usando para esto un conjunto de descriptores que son extraídos de un vocabulario controlado o *tesauro*. Esto mejora muchos aspectos organizativos y facilita el acceso rápido a la información relevante a una determinada materia.

Muchos parlamentos europeos usan un tesauro, llamado **Eurovoc**, para clasificar las iniciativas parlamentarias. La *Food and Agrigulture Organization* (**FAO**) emplea **Agrovoc** para sus documentos, y la *National Library of Medicine* (**NLM**) usa **MeSH** para indexar los artículos científicos de revistas biomédicas.

La tarea de asignar materias, o descriptores, de un tesauro a un documento se lleva a cabo por documentalistas de forma manual, lo que es un proceso lento y costoso.

En este trabajo se pretende evaluar los diferentes algoritmos y técnicas enfocadas a la clasificación documental y en base a estos resultados se propondrá el diseño e implementación de una herramienta informática que pueda ayudar a los expertos humanos en esta tarea, aplicada al caso específico de iniciativas parlamentarias del Parlamento de Andalucía.

Durante los siguientes capítulos estudiaremos las técnicas y algoritmos más destacados en el campo de la clasificación de documentos. Veremos que pasos se han de seguir a la hora de trabajar con documentos de texto antes de que sea posible su clasificación. Implementaremos clasificadores con distintos de estos algoritmos y evaluaremos su rendimiento en base a una serie de métricas.

Por último, se presentará un diseño e implementación para una interfaz sencilla, con la que será posible llevar a cabo el proceso de la clasificación de este tipo de documentos.

# Capítulo 1

# Clasificación de iniciativas Parlamentarias

Durante este capítulo nos centraremos en definir el problema de clasificar manualmente documentos y como es llevado a cabo de forma habitual por parte de los expertos que se encargan de esta tarea. Veremos como técnicas de Aprendizaje Automático pueden sernos útiles a la hora de clasificar documentos comprobaremos si dichas técnicas pueden aplicarse estas técnicas a nuestro problema.

## 1.1. Definición del problema de la clasificación de iniciativas Parlamentarias

La gran cantidad de información generada por el Parlamento de Andalucía es archivada para facilitar su acceso de forma sencilla y eficiente.

Entre estos documentos generados se encuentran los *Diarios de Sesiones*. Estos documentos son transcripciones de las intervenciones de los parlamentarios en la cámara y contienen una serie de iniciativas parlamentarias. Estas iniciativas son preguntas, proposiciones no de ley o propuestas que son se llevadas a pleno o a comisiones por los parlamentarios. Cada sesión se compone de una serie de iniciativas de manera que en cada legislatura se generan gran cantidad de estos documentos. Estos documentos pueden ser consultados en la página Web del Parlamento Andaluz [1]. Cada una de estas iniciativas posee un título ,o pequeño resumen, donde se describe brevemente el motivo, o el contenido de la iniciativa. A este título le siguen una serie de párrafos que corresponden a las diferentes intervenciones que han realizado los parlamentarios en relación a esa iniciativa.

[1] http://www.parlamentodeandalucia.es/webdinamica/portal-web-parlamento/recursosdeinformacion/diariosdesesiones/plenos.do

Estas iniciativas se dirigen al servicio de documentación de la cámara autonómica, donde varios documentalistas procesan las iniciativas secuencialmente de forma manual e independiente. De manera que cada iniciativa es clasificada por un único documentalista. Cada documentalista tiene el trabajo de seleccionar los descriptores adecuados que mejor describan a una iniciativa.

La manera aleatoria del reparto implica que todos los documentalistas pueden clasificar cualquier materia, independientemente de su temática y etiquetar con gran precisión todas las iniciativas que les corresponda.

Esto en la mayoría de los casos les puede resultar sencillo pero en otros casos puede ser más complicado, ya sea porque el contenido de la iniciativa es más ambigüo o porque es necesario más de un descriptor para su correcta clasificación. En estos casos requieren que los expertos colaboren para conseguir una correcta clasificación.

Vemos que el proceso presenta una serie de problemas. En primer lugar, el proceso es completamente manual, además los documentalistas no disponen de especialización en ningún área, han de ser capaces de etiquetar todos los descriptores, y por último no existe ninguna colaboración entre los documentalistas.[12][8]

Este trabajo se basa en la hipótesis de que siendo asistidos por una herramienta software que les ayude en el proceso de toma de decisiones, se deben obtener mayor precisión en los resultados de la clasificación.

El objetivo de este trabajo consiste en estudiar la clasificación automática de documentos y proponer un software que asista a los expertos durante el proceso de clasificación.

En nuestro caso concreto disponemos de una serie de iniciativas correspondientes a la VIII Legislatura del Parlamento De Andalucía. Cada uno de estos documentos tiene una estructura común.

Cada documento de nuestra colección viene en formato *XML*. El uso de este tipo de formato permite dar a la información una estructura bien definida, con lo que es sencillo reutilizar la información. Que la información sea estructura quiere decir que se compone de partes, y estas partes a su vez se componen en otras partes. Entonces esta información viene representada en forma de árbol del que cuelgan una serie de nodos con distinta información. Habitualmente estos nodos se denominan **marcas** cuando se trabaja con *lenguajes* de marcas como es el caso del lenguaje *XML*.

En concreto, cada iniciativa está formada por una marca inicial llamado *Iniciativa-Completa*, esta marca inicial contendrá al resto de marcas de los que se compone el documento.

El formato de una iniciativa se muestra en la imagen siguiente:

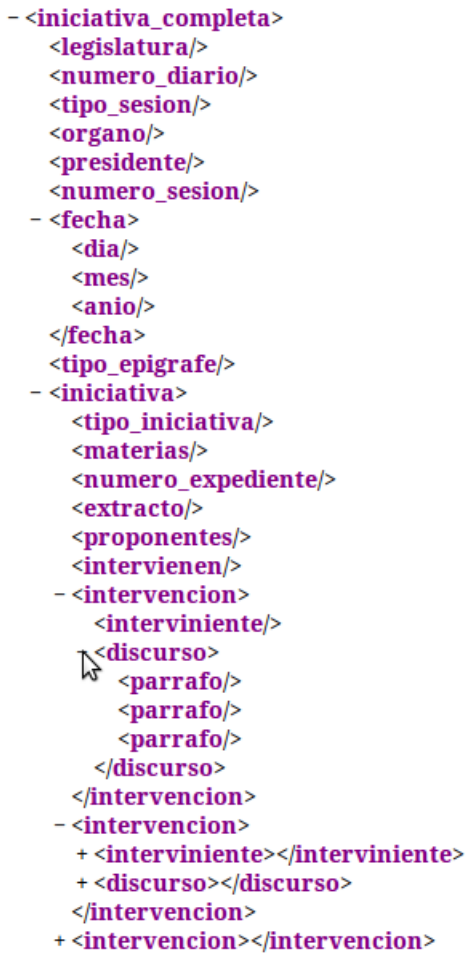



Figura 1.1: Estructura de una iniciativa.

Las primeras marcas de las estructura corresponden a información general del documento, como son:

- **Legislatura**: número que identifica la legislatura a la que pertenece dicha iniciativa. Está representado por un número en formato romano. Será el mismo para todos documentos que vamos a tratar.
- **Tipo de sesiones**: por lo general será de tipo ordinaria.
- **Organo**: comisión parlamentaria encargada de la discusión de la iniciativa.
- **Presidente**: presidente de la comisión que propone la iniciativa.

- **Número de sesión**: número que identifica la sesión en la que es emitida la iniciativa.
- **Fecha**: esta marca contendrá el día, mes y año en que se realizó esta iniciativa.
- **Tipo de epígrafe**: serán preguntas orales ,comparecencias, proposiciones de Ley, proposiciones no de Ley.

Esta información nos resulta inútil a la hora de asignar un descriptor. En la siguiente imagen podemos ver como el formato que posee esta parte del documento.

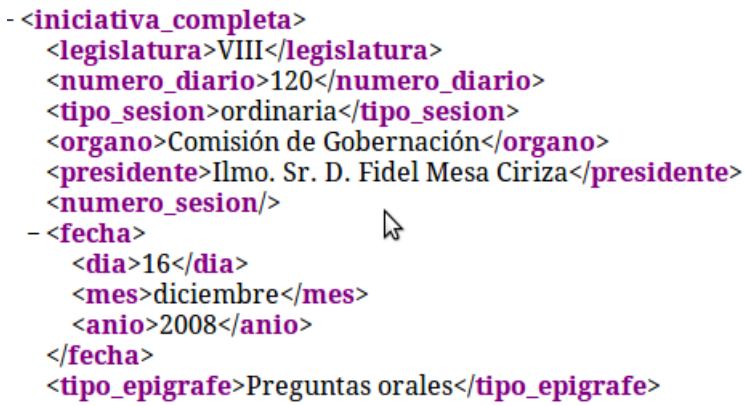



Figura 1.2: Información general de la iniciativa.

La última marca posee el nombre de **iniciativa** y en ella está almacenada la mayor parte de la información del documento, dentro de esta marca pueden diferenciarse otra serie de marcas que cuelgan de ella, como son:

- **Tipo iniciativa**: siempre es igual al tipo de epígrafe.
- **Materias**: serie de descriptores mas representativos para el contenido de la iniciativa. Estas materias, o descriptores, son las que han sido asignadas por los documentalistas a una iniciativa ya procesada.

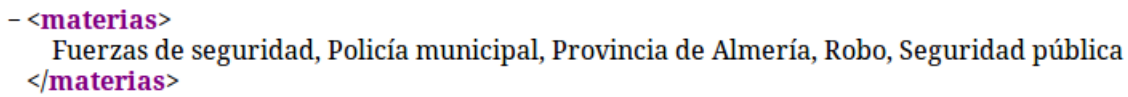



Figura 1.3: Materias de una iniciativa

- **Número de expediente**: número identificador, distinto para cada iniciativa.

- **Extracto**: tesis del contenido del documento. Consiste en una frase que describe el contenido de la iniciativa de forma breve pero en ocasiones muy concisa.

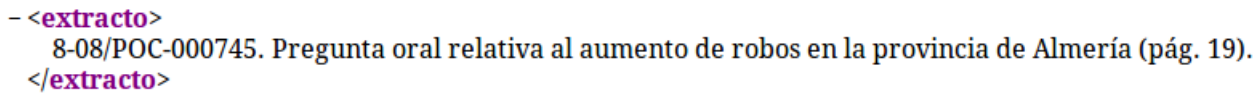



Figura 1.4: Extracto de una iniciativa.

- **Proponentes**: nombres que corresponden a las personas que proponen la iniciativa.

- **Intervienen**: nombres de las personas que intervienen en las preguntas orales o comparecencia de la iniciativa en cuestión.

- **Intervención**: esta marca se repite tantas veces como intervenciones haya. En ella pueden encontrarse diferentes marcas que estructuran su contenido en diferentes partes. Entre estas están la marca **intervinientes**; que contiene el nombre de las personas que participan en esta intervención y otra marca **discurso**; que contiene una serie de marcas denominadas *párrafo*, en las que se almacena la información de cada párrafo del contenido real de la iniciativa.

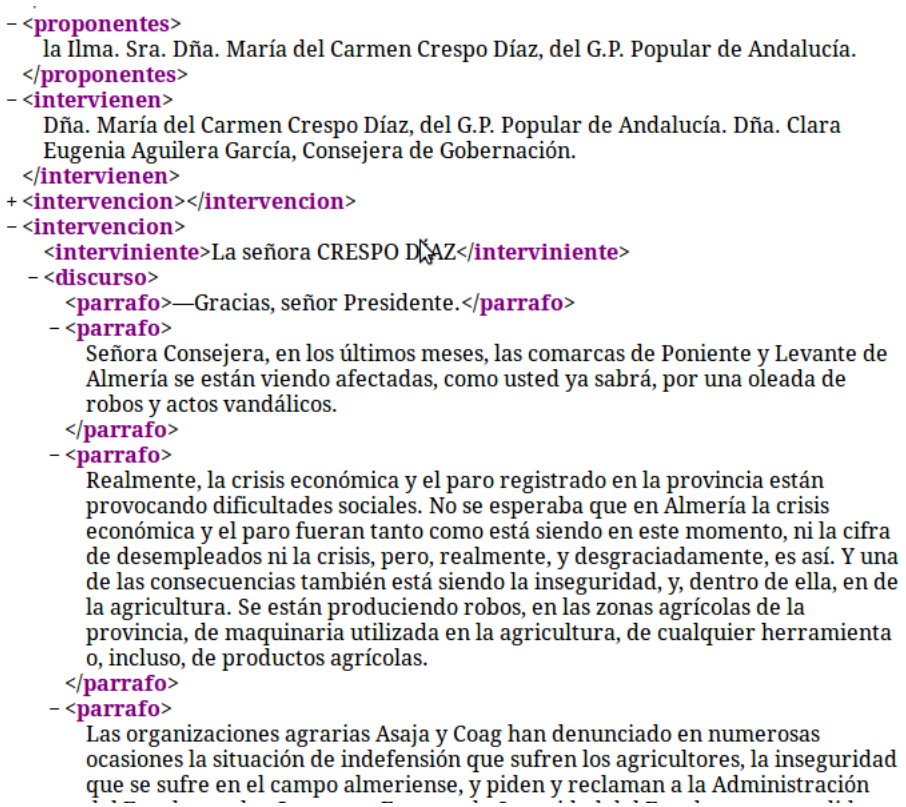



Figura 1.5: Intervención en una iniciativa.

De todo este árbol las marcas que nos interesan, para extraer información relevante del contenido de la iniciativa, son los que tienen nombre **párrafo** y **extracto**. En la primera de estas marcas encontramos la información que ha aportado cada parlamentario de los que intervienen en la iniciativa. En la marca *extracto* podemos encontrar información que sea útil para intuir de que puede tratar un documento pero a veces resulta insuficiente por su escaso contenido.

Nuestro objetivo es que a partir de la información de las marcas *párrafo* y/o *extracto* nuestra herramienta sea capaz de asignarlas a uno o varios descriptores.

## 1.2. Aprendizaje automático y clasificación documental

El *Aprendizaje automático* es una rama de la Inteligencia Artificial encargada del diseño y desarrollo de algoritmos que permiten a una máquina mejorar su comportamiento automáticamente a través de la experiencia. Un

algoritmo de aprendizaje automático es capaz de extraer características y patrones comunes de un conjunto de datos y aplicarlos a nuevos conjuntos de datos.

El aprendizaje automático se puede dividir en dos tipos: el aprendizaje supervisado y el aprendizaje no supervisado.

En el *aprendizaje automático supervisado* se deduce una función a partir de un conjunto de datos de entrenamiento que ya están etiquetados correctamente. En este tipo de algoritmos se proporciona un conjunto de datos de entrenamiento en forma de vector. El algoritmo infiere una función conocida de clasificación o función de regresión, dependiendo de si la salida es discreta o continua. La función inferida debe permitir predecir la clase para cualquier vector de entrada.

En primer lugar se lleva a cabo la fase de entrenamiento. Durante esta primera fase se extraen los atributos relevante para el clasificador. Estos atributos se habrán clasificado previamente por lo que se conoce su clase. De la extracción de atributos sale un conjunto de vectores que formarán la entrada al algoritmo, que a partir de los datos y sus clases generará un modelo que generalice las características extraídas. En una segunda fase, la de predicción, se realizará el mismo proceso de extraer atributos para generar un vector. El algoritmo, como ya habrá sido capaz de aprender un modelo, podrá aplicarlo a los datos y de esta manera predecir las clases de estas muestras.

En el *aprendizaje automático no supervisado* se intenta agrupar grandes cantidades de datos en función de las características que comparten. En estos métodos no se conoce nada a priori. Es una manera de encontrar jerarquía y orden en un conjunto de datos sin estructura. Uno dos principal métodos del aprendizaje no supervisado es el *clustering*. Lo más importante en este método es encontrar una función que sea capaz de cuantificar las similitudes entre datos como un número.

El aprendizaje automático se usa en diferentes campos como el diagnostico medico, el análisis de mercados, el marketing, clasificación de imágenes, clasificación de secuencias de ADN o la clasificación automática de documentos.[18]

### 1.2.1. Clasificación documental

Normalmente, cuando hablamos de clasificación automática de documentos nos referimos a la *clasificación documental*. Esta tarea no es más que una fase en el proceso de organización de documentos. En este proceso se identifican y establecen agrupaciones entre los documentos de acuerdo a su contenido. Esto implica un análisis del texto para encontrar términos cuyo significado describa el documento que los contiene. Existen multitud

de técnicas para extraer información relevante dentro del contenido de los documentos. En común, estas técnicas son capaces de definir esquemas basados en la representación de vectores de características de cada uno de los documentos según los términos o palabras que aparecen en este. Cada uno de estos vectores representa un conjunto de características que definen el documento y su importancia relativa.

No existe un método que en particular resuelva el problema de la *Clasificación Documental* de manera óptima. En ocasiones para resolver estos problemas se combinan varios de estos métodos para alcanzar una mayor precisión a la hora de clasificar los documentos.

Al hablar de clasificación documental podemos encontrar dos posibles escenarios o situaciones. Básicamente, la diferencia entre estos escenarios están en que; por un lado, tenemos una situación en la que se parte de una serie de descriptores o categorías conceptuales pre-diseñadas a priori, y en la que el trabajo a realizar por el clasificador, es asignar a cada documento el descriptor que le corresponda. Este tipo de escenario se conoce con el nombre de *Clasificación Supervisada*,ya que para resolver problemas en este escenario se emplean técnicas de aprendizaje automático supervisado, y requiere de la elaboración manual del esquema de los descriptores, además requiere de un *proceso de aprendizaje* o *entrenamiento* por parte del clasificador, que deberá ser supervisado manualmente. Habitualmente denominamos a la clasificación automática de textos, mediante las técnicas comunes del Aprendizaje automático, con el nombre de **categorización**.

En el segundo escenario no disponemos de categorías previas ni esquemas de los descriptores. En este caso la clasificación de los documento depende de ellos mismos. Este caso se conoce como *Clasificación no Supervisada*. Para resolver problemas de este tipo se emplean técnicas de aprendizaje no supervisado, en este caso se debe realizar una idenficación automática de dichas categorías, el problema se denomina **clustering de textos**.

Existen multitud de técnicas capaces de realizar la tarea de clasificar documentos. Básicamente la idea de estos métodos es la de conseguir, de alguna manera, construir un patrón representativo de cada uno de los descriptores y posteriormente aplicar una función que permita estimar la similitud entre el documento a clasificar y cada uno de los patrones que describen a los descriptores.
Un clasificador puede expresarse como una función, $f(x)$, que asigna una clase a una muestra:

$$f : \mathbf{X} \mapsto \mathbf{C} \tag{1.1}$$

donde $X$ es el conjunto de muestras (iniciativas o sus características) y $C$ el conjunto de etiquetas o clases de $X$.

Para la construcción de los patrones, de cada uno de los descriptores, utilizamos documentos clasificados manualmente. Estos documentos servirán como base de conocimiento para extraer información con la que serán generados cada uno de los patrones.

La categorización de textos suele desarrollarse en tres etapas: *procesamiento de datos, construcción del clasificador y categorización* (o asignación de descriptores a nuevos documentos).

En este capítulo daremos una pequeña definición de la primera, aunque junto al resto se detallarán de manera más profunda en el *capítulo 2* dado que los procesos dependerán directamente del algoritmo que se emplee.

A la hora de trabajar con texto, la fase del procesamiento de datos se maneja de una forma particular. Durante esta fase se transfiere el texto original de cada iniciativa a un formato compacto, que será utilizado en una serie de fases posteriores. Las fases que se siguen para transformar el texto original son las siguientes:

- *Reducción del contenido del texto*: suelen emplearse técnicas para reducir el tamaño de los textos. Estas técnicas extraen los contenidos mas relevantes en un texto, aportando como resultado un resumen del texto original.

- *Tokenización* del contenido resultante de la reducción de la dimensionalidad. Este proceso descompone el texto en elementos denominados *tokens*. Los tokens son unidades indivisibles del lenguaje.

  Por ejemplo en la siguiente frase:

  *Tengo un equipo de informáticos, formado por informáticos e informáticas.*

  tras un proceso de tokenización obtendríamos:

  *['Tengo', 'un', 'equipo', 'de', 'Informáticos', ',', 'formado', 'por', 'informáticos', 'e', 'informáticas', '.', 'Correo', ':', 'TFG', '@', 'correo.ugr.es'].*

  Como podemos ver el tokenizador no modifica el contenido de un párrafo, no elimina los simbiosis de puntuación.

- *Reducción de la dimensionalidad o selección de atributos*: el objetivo es reducir la dimensionalidad (número de atributos) del espacio de datos, eliminando aquellos que sean irrelevantes para la clasificación. Esto suele realizarse eliminando palabras que no aporten ningún significado al contenido, se denominan palabras vacías ( o Stop Words, que suelen

ser artículos, preposiciones, conjunciones,etc ) y otras dependencias del dominio.

Siguiendo con el ejemplo anterior si aplicamos esta tarea a la frase nos daría como resultado la siguiente frase:

*'Tengo', 'equipo', 'Informáticos', 'formado', 'informáticos', 'informáticas', 'Correo', 'TFG', '@', 'correo.ugr.es'*

Vemos que han desaparecido tokens que no aportaban ningún significado como por ejemplo *"de"* y *"por"*, también se han eliminado los puntos suspensivos. Además el correo electrónico ha sido separado a partir del término @, dando lugar a tres tokens.

- *Lematización*( o *stemming*), que reduce una palabra a su raíz (stem). Durante esta fase se extraen los sufijos y prefijos comunes de palabras literalmente diferentes pero con una raíz común que pueden ser consideradas como un sólo término.

  El resultado para nuestro ejemplo es:

  *'teng', 'equip', 'informat', 'form', 'informat', 'informat', 'corre', 'tfg', '@', 'correo.ugr.'*

  Como podemos ver ahora ya no se distingue entre mayúsculas y minúsculas. Ahora las palabras *informáticos e informáticas* se consideran como una sola. El correo electrónico ha perdido la "*-es*"final, esto se debe a que "*-es*"se usa en español para formar el plural y el stemmer lo considera así. Si en lugar de *es* hubiese sido *.com* no se habría eliminado. Existen varios algoritmos para realizar el stemming. En el *Capítulo 3* se detallará el algoritmo con el que trabajaremos para realizar esta tarea.

- *Asignación de distintos pesos a los atributos (o el mismo peso).* Suele usarse la técnica llamada **Tf-idf** (*term frequency-inverse document frequency*) para evaluar como de importante es una palabra dentro del conjunto de documentos. Este proceso se realiza después de realizar el stemming ya que nos asegura que la forma de las palabras no penalicé las frecuencias de estas.

Estos pasos, realizados de manera individual para cada documento de nuestra colección, convierten cada documento a una representación compacta y adecuada para usarla con algoritmos de aprendizaje.[16]

Durante la etapa de desarrollo de los clasificadores (*Capítulo 3*) veremos de forma mas profunda la forma en que actúan este tipo técnicas y los resultados que producen en el documento original.

Después de realizar la transformación de los documentos a conjuntos de atributos y valores, estos se utilizarán para construir un clasificador que será capaz de asignar categorías a nuevos documentos.

El proceso de generación de patrones es lo que se conoce como *fase de entrenamiento* del clasificador o *fase de aprendizaje*, y la colección de documentos pre-clasificados se conoce como *conjunto de entrenamiento*. Esta etapa se aborda de manera diferente dependiendo del algoritmo que estemos usando, por lo que definiremos mejor el proceso cuando entremos a definir los algoritmos que vamos a emplear (Capítulo 2).

Así mismo, se define con el nombre de *fase de predicción* o *fase de test*, a la fase en la que se eligen las categorías o predictores y la colección de documentos usada se denomina *conjunto de test.*[28]

## 1.3. Otras aplicaciones de la clasificación documental

Hoy en día se trata con una cantidad de información digital muy elevada, por lo que cada vez es más necesario almacenarla de forma que el tiempo para acceder a ella sea el mínimo posible. Esto nos lleva a la necesidad de desarrollar aplicaciones capaces de organizar esta información de manera eficiente. De esta forma se han desarrollado aplicaciones basadas en la clasificación documental para resolver problemas muy diversos.

La clasificación de documentos se ha usado con éxito en el filtrado de documentos, las asignación automática de descriptores a documentos, también se usa en *routing* o en la recuperación de información basadas en *browsing* entre otras.

Una aplicación con muchas características en común con el presente trabajo es el filtrado de documentos, que es usado por las aplicaciones de correo electrónico como mecanismo *anti-spam* . Este tipo de herramientas deciden si un correo electrónico es considerado *spam* o no. Para esto se analizan correos que previamente han sido catalogados como *spam* por el usuario. Hay que aclarar que los filtros *anti-spam* también se basan en listas de direcciones que son catalogadas directamente como *spam*, estas listas se conocen como *blacklist.*[19]

También tiene un uso muy común en periódicos y revistas que clasifican sus artículos en función a su contenido asignándoles al menos un descriptor por artículo. Otro uso muy de moda en la actualidad se lo dan las redes sociales como por ejemplo *Twitter* que ya permite a sus usuarios clasificar los *tweets* en función a su contenido.

# Capítulo 2

# Algoritmos y técnicas aplicadas a la clasificación automática de documentos

Durante capítulo describiremos los diferente algoritmos y técnicas que suelen emplearse en la categorización de documentos. De entre esos elegiremos algunos de ellos para hacer frente al problema de la categorización de iniciativas parlamentarias.

Estos no son todos los algoritmos que pueden dedicarse a la clasificación documental pero si son los mas populares en los últimos tiempos.

## 2.1. Algoritmo Naive Bayes

Las redes bayesianas han sido uno de los métodos más utilizados en aprendizaje automático durante estos últimos años. Es un método muy importante ya que no sólo ofrece un análisis cualitativo de los atributos y valores que pueden intervenir en el problema, si no porque dan cuenta de la importancia cuantitativa de esos atributos.

En el aspecto cualitativo, este tipo de clasificadores ayudan a representar como se relacionan esos atributos ya sea de forma causal, o indicando simplemente la correlación que existe entre los distintos atributos.

Cuantitativamente, este clasificador da una medida probabilística de la importancia de esas variables en el problema. Esta es quizá una de las diferencias fundamentales que ofrecen las redes bayesianas con respecto a otros métodos.

Entre las características que poseen este tipo de métodos en tareas de aprendizaje se pueden destacar las siguientes:

- Con cada ejemplo observado se modifica la probabilidad de que la hipótesis formulada sea correcta. Esto quiere decir que una hipótesis que no concuerda con un conjunto de ejemplos más o menos grande no es desechada si no que será disminuida la probabilidad estimada para esa hipótesis.
- Estos método son robustos frente al ruido presente en los ejemplos de entrenamiento y la posibilidad de que entre este conjunto de ejemplos existan algunos incompletos o erróneos.
- Dan resultados con probabilidades asociadas.
- Pueden clasificar combinando las predicciones de varias hipótesis.
- Se puede incluir conocimiento a priori: probabilidad de cada hipótesis; y la distribución de probabilidades de los ejemplos.

El aprendizaje bayesiano se puede ver como el proceso de encontrar la hipótesis más probable, dado un conjunto de ejemplos de entrenamiento $D$ y un conocimiento a priori sobre la probabilidad de cada.

Si denotamos con $P(D)$ a la probabilidad a priori de los datos, $P(D|h)$ la probabilidad de los datos dada una hipótesis, lo que queremos estimar $P(h|D)$, la probabilidad posterior de $h$ dados los datos. Eso lo podemos estimar con el **Teorema de Bayes**:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \tag{2.1}$$

Para estimar la hipótesis más probable o MAP(*maximum a posteriori hypothesis*): [23]

$$\begin{aligned} h_{MAP} &= argmax_{h\in H}(P(h|d)) \\ &= argmax_{h\in H}(\frac{P(D|h)P(h)}{P(D)} \\ &= argmax_{h\in H}(P(D|h)P(h)) \end{aligned} \tag{2.2}$$

Ya que $P(D)$ es una constante independiente de $h$.

Asumiendo que todas la hipótesis son igualmente probables, nos queda la hipótesis de máxima verosimilitud o ML (*maximum likelihood*):

$$h_{ML} = argmax_{h\in H}(P(D|h)) \tag{2.3}$$

Una forma de aplicar un algoritmo Bayesiano es calculando para todas la posibles hipótesis $P(h|D)$ y quedándose con la mayor probabilidad. Esto

hace que su coste computacional sea muy elevado y dependa linealmente del número de hipótesis.

También para aplicarlo, necesitamos especificar los valores para $P(h)$ y para $P(D|h)$. Si asumimos que no hay ruido y todas las hipótesis son igualmente probables ($P(H) = \frac{1}{|H|} \forall h \in H$), $P(D|h) = 1$ si, y solo si, $D$ es consistente con $h$. Esto es:

$$P(h|D) = \frac{1}{|VS_{H,D}|} \tag{2.4}$$

donde , $VS_{H,D}$ es el subconjunto de hipótesis de $H$ que es consistente con $D$. Por lo tanto, toda hipótesis consistente es hipótesis $MAP$. Esto quiere decir que todo sistema de aprendizaje que nos de hipótesis consistentes, asumiendo que no hay ruido y que todas las hipótesis son igualmente probables, nos da hipótesis MAP.[20]

### Clasificador basado en Navie Bayes

Este tipo de clasificador se utiliza cuando queremos clasificar una instancia descrita por un conjunto de atributos ($a_i$) en un conjunto finito de clases ($V$). Deseamos clasificar un nuevo ejemplo de acuerdo con el valor mas probable dados los valores de sus atributos:

$$v_{MAP} = argmax_{v_j \in V} P(v_j|a_1, .., a_n) \tag{2.5}$$

Lo que aplicando el teorema de Bayes, visto anteriormente nos queda:

$$\begin{aligned} v_{MAP} &= argmax_{v_j \in V} \frac{P(a_1, .., a_n|v_j)P(v_j)}{P(a_1, ..., a_n)} \\ &= argmax_{v_j \in V}(P(a_1, .., a_n|v_j)P(v_j)) \end{aligned} \tag{2.6}$$

Tanto el término $P(v_j)$ como el término $P(a_1, ..., a_n|v_j)$ se han de estimar basándose en los ejemplos de entrenamiento. $P(v_j)$ se calcula contando la frecuencia con la que ocurre cada valor $v_j$ El coste de calcular $P(a_1, ..., a_n)$ es muy elevado ya que es necesario tener muchos ejemplos Para estimar el término $P(a_1, a_2, a_3, ...|v_j)$, es decir, las veces en que para cada de entrenamiento. La suposición del clasificador naive es que los atributos son independientes entre si con respecto al concepto objetivo y , por la tanto:

$$P(a_1, .., a_n|v_j) = \prod_i P(a_i|v_j) \tag{2.7}$$

La aproximación del clasificador bayesiano Naive es:

$$v_{nb} = argmax P(v_j) \prod_i P(a_i|v_j) \tag{2.8}$$

Las probabilidades $P(a_i|v_j)$ son mucho mas fáciles de estimar que las $P(a_1, .., a_n)$

En el caso particular de la clasificación de textos las instancias son los documentos de textos y el objetivo es : dado un documento asignarle una posible hipótesis.[17][20]

## 2.2. Algoritmo Random Forest

Random Forest es un algoritmo para clasificación y regresión de amplio uso que tiene un rendimiento bastante bueno para datos de alta dimensionalidad. Puede considerarse como la integración de las siguientes técnicas: Decision Trees, Bagging, y Random Subspace.

Vamos a definir por separado cada una de estas técnicas.

Los árboles de decisión son estructuras lógicas con amplia utilización en la toma de decisiones, predicción y minería de datos. Un árbol de decisión puede ser tan simple como la imposición de una regla sobre el comportamiento mayoritario de un conjunto de datos. Este tipo de reglas son siempre de la forma *IF...THEN...*, y pueden aplicarse tanto a variables continuas como discretas. La pregunta que se plantea en los árboles de decisión es elegir una regla para obtener el árbol de decisión mas preciso. Un aspecto esencial en los algoritmos basados en árboles de decisión es la elección del mejor criterio para dividir los datos. Una de las mejores opciones es realizar esta selección basándose en el concepto de entropía.

$$Entropia = -\sum_{\forall x} p_x log_2(p_x) \tag{2.9}$$

donde $p_x$ es la probabilidad de la clase $x$. Un valor alto de entropía produce mas impredicibilidad, mientra que un valor bajo de la entropía se interpreta como una menor impredecibilidad.

La *ganancia de información* es la reducción de la entropía causada por el particionado de los datos por un determinado atributo.

La técnica **bagging** se enmarca dentro de las llamadas técnicas *ensamble methods*. Este tipo de técnicas son combinaciones de modelos en la que se ha de conocer como se van a crear los modelos y como se van a combinar los resultados de cada uno de los modelos para producir una predicción final. El objetivo de estos métodos es producir una mejor predicción que los métodos habituales (miembros individuales del ensemble). Esta técnica manipula el conjunto de entrenamiento de forma que se extraen muestras diferentes del

conjunto de entrenamiento ( muestras bootstrap), y se utilizan estas muestras como si fuera el conjunto de entrenamiento verdadero. En bagging, cada miembro del *ensemble* es entrenado con una muestra diferente del conjunto de entrenamiento. El tamaño es el mismo que el conjunto de entrenamiento original pero no su composición.El resultado final es el promedio de la predicción de cada uno de los miembros del ensamble.

La técnica *random subspace* es una técnica de manipulación de atributos. En este caso, el clasificador se construye haciendo uso sólo de un subconjunto de variables, de tal forma que cada clasificador se especializa en un subconjunto de atributos. Esta técnica debe usarse con cuidado ya que la eliminación de atributos puede afectar muy significativamente al rendimiento de los clasificadores.[15]

El algoritmo Random Forest consiste en aplicar bagging usando árboles de decisión como predictivos pero construidos eligiendo en cada nodo, para bifurcar, el mejor de entre una muestra aleatoria de los atributos. Estos árboles no se podan.[3]

### 2.2.1. Clasificador basado en Random Forest

Para clasificar un nuevo objetivo, cada árbol del ensamble lo toma como entrada y produce una salida, su clasificación. La decisión del *ensamble* se toma como la clase con mayoría de votos.

La técnica *Random Forest* engloba a las tres técnicas anteriores. Cualquier conjunto de clasificadores en el que el clasificador base es un árbol de decisión construido a partir de un vector de números aleatorios, donde cada uno de ellos son independientes e idénticamente distribuidos y el resultado del clasificador final se obtiene mendiante votación no ponderada se conoce como *Random Forest*. El objetivo de este método es inyectar al algoritmo la aleatoriedad justa para maximizar la independencia de los árboles manteniendo una precisión razonable. [4]

Entre las características de la técnica *Random Forest* cabe destacar las siguientes[5]:

- Funciona de manera eficiente en grandes bases de datos.
- Aporta estimaciones de que variables son más importantes en la clasificación.
- La construcción de árboles es rápida, al explorar sólo unos pocos atributos.
- Los nodos que contienen atributos muy descriptivos compensan a los otros.

- Se calculan proximidades entre pares de casos que se pueden utilizar en la agrupación, la locación de los valores atípicos, o dar interesante vistas sobre los datos.
- Ofrece un método experimental para detectar interacciones entre variables.

## 2.3. Máquinas de vectores soporte (SVM)

La máquinas de vectores soporte (**SVM**, del inglés *Support Vector Machines*) tienen su origen en los trabajos sobre la teoría del aprendizaje estadístico y fueron introducidas en los años 90 por *Vapnik*.

Originalmente fueron pensadas para resolver problemas de clasificación binaria, actualmente se utilizan para resolver todo tipo de problemas ( regresión, agrupamiento, multiclasificación).

Dentro de la tarea de clasificación, las *SVMs* pertenecen a la categoría de los clasificadores lineales, puesto que inducen separadores lineales o hiperplanos, ya sea en el espacio original de los ejemplos de entrada, si estos son separable o cuasiseparebles (por el ruido), o en un espacio de transformado (espacio de características), si los ejemplos no son separables linealmente en el espacio original.
Básicamente las máquinas de vectores soporte son un algoritmo de patrones binario, cuya finalidad es la de asignar cada patrón a una clase.

Para explicar el funcionamiento de las SVM empezamos por el caso mas simple, aquel en que los datos son linealmente separables. Los datos de entrenamiento serán una serie de vectores de la forma :

$$\vec{x_i}, y_i i = 1, ..., l \tag{2.10}$$

Donde cada $x_i$ es el un vector de observaciones en un espacio de dimensión $d$. Y el valor $y_i \in -1, 1$. El problema consistirá en asignar cada vector a su clase correspondiente,1 ó -1, para lo que construye un hiperplano de separación que divida el espacio $R^d$ en dos regiones. Las muestras que caigan a un lado del hiperplano pertenecerán a la clase .1 y las que caigan en la otra a la clase 1. Los puntos $\vec{x}$ que caen justo en este hiperplano satisfarán la ecuación:

$$\vec{w}.\vec{x} + b = 0 \tag{2.11}$$

Donde $\vec{w}$ es un vector normal al hiperplano de separación y $b$ una constante. Así, $\frac{|b|}{||\vec{w}||}$ sera la distancia perpendicular desde el hiperplano al origen, donde $||\vec{w}||$ es la norma euclidea de $\vec{w}$. Debemos calcular dos distancias mas ,$d_+$ y $d_-$ existentes entre el hiperplano de separación y las muestras mas cercanas da la clase 1 y -1 respectivamente.

Con estas últimas distancias se define el margen del hiperplano de separación, este margen será la distancia entre las muestras más cercanas de las clases:

$$m = d_+ + d_- \tag{2.12}$$

Para problemas de este tipo el objetivo será encontrar, de todos los posibles, el hiperplano de separación que hace máximo este margen $m$. A la hora de formular el problema formalmente supondremos que todas las muestras de entrenamiento cumplen una de las siguientes restricciones:

$$\vec{x_i}\vec{w} + d >= +1 si y_i = +1 \vec{x_i}\vec{w} + d <= -1 si y_i = -1 \tag{2.13}$$

Las restricciones anteriores se traducen en que los vectores son separables en dos clases, combinando ambas restricciones obtenermos:

$$y_i(\vec{x_i}\vec{w} + b) - 1 >= 0 \forall i \tag{2.14}$$

Ahora consideremos los puntos para los que se cumple: $\vec{x_i}\vec{w} + b = +1$. Estos puntos son los más cercanos al hiperplano y conocidos como vectores soporte, que estarán contenidos en otro nuevo hiperplano, al que llamaremos $H_1$ y cuya ecuación es la expresada anteriormente. De forma análoga se define el hiperplano $H_2$, cuya ecuacion será : $\vec{x_i}\vec{w} + b = -1$.

Los hiperplanos $H_1$ y $H_2$ son paralelos al hiperplano de separación, por lo tanto la componente normal se mantiene ($\vec{w}$). Y las correspondientes distancias al origen serán: $\frac{|1-b|}{||\vec{w}||}$ para $H_1$ y $\frac{|-1-b|}{||\vec{w}||}$ para $H_2$. Si el problema cumple las restricciones que hemos indicado anteriormente las distancias $d_+$ y $d_-$ serán $\frac{1}{||\vec{w}||}$ por lo que el margen se define como:

$$m = d_+ + d_- = \frac{1}{||\vec{w}||} + \frac{1}{||\vec{w}||} = \frac{2}{||\vec{w}||} \tag{2.15}$$

Volviendo al objetivo principal de las máquinas de soporte es entrar el hiperplano de separación que maximize el margen. Con la formulación mostrada hasta ahora el problema se reduce a minimizar $||\vec{w}||^2$ sujeto a la restricción:

$$y_i(\vec{x_i}\vec{w} + b) + 1 >= 0 \forall i \tag{2.16}$$

Para seguir explicando el desarrollo teórico del algoritmo es necesario cambiar a la formulación de Lagrange. Este tipo de formulación permite resolver un problema de optimización con restricciones mediante la introducción de nuevas variables, los multiplicadores de Lagrange, $\alpha_i$. Puede probarse que es posible obtener el hiperplano óptimo de separación, mediante la combinación lineal de los vectores de soporte. El peso de cada uno de estos vectores se obtiene mediante los multiplicadores de Lagrange.

Además al cambiar la formulación la restricción $y_i(\vec{x_i}\vec{w}+b)+1 >= 0 \forall i$ puede sustituirse por restricciones en los multiplicadores de Lagrange, lo que hace que la dificultad disminuya. Otro motivo no menos importante es que los datos de entrenamiento sólo aparecerán en forma de productos escalares entre vectores. Esta propiedad permite generalizar el procedimiento a casos no lineales.

Comenzamos introduciendo los multiplicadores de Lagrange (positivos) [6], $\alpha_i i = 1, ..., l$, uno por cada desigualdad de la restricción. Para cada restricción de la forma $c_i >= 0$, como en nuestro caso, debemos introducir un multiplicador de Lagrange, $\alpha_i$, que multiplique a la restricción, y posteriormente restar estas restricciones de la función objetivo, que en nuestro caso es la función a minimizar ,$\frac{1}{2}||\vec{w}||^2$ . Con todo esto obtenemos como resultado:

$$\begin{aligned} L_p &\equiv \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{l} \alpha_i (y_i(\vec{x_i}\vec{w}+b) - 1) \\ &= \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{l} \alpha_i y_i (\vec{x_i}\vec{w}+b) + \sum_{i=1}^{l} \alpha_i \end{aligned} \tag{2.17}$$

El siguiente paso es minimizar $L_p$ con respecto a las variables fundamentales $\vec{w}$,$b$, simultáneamente necesitamos que las derivadas de $Lp$ con respecto a las variables duales,$\alpha_i$,desaparezcan. A todo esto unido a la restricción de que $\alpha_i >= 0$, lo llamaremos conjunto de restricciones $C_i$.

Teniendo en cuenta que la función y el conjunto formado por los puntos que satisfacen las restricciones son convexos, la minimización de $L_p$ se puede tratar como un problema de programación cuadrática convexo. Para solucionar el problema se puede hacer uso de la dualidad y resolver el problema dual, cuya resolución será equivalente. El problema dual consiste en maximizar $L_p$ sujeto a unas restricciones. La primera es que la gradiente de $L_p$ con respecto a $\vec{w}$,$b$ y $\alpha$ debe anularse, por otro lado se debe seguir cumpliendo que $\alpha_i >= 0$. Llamamos a este conjunto $C_2$. El problema dual cumple la propiedad de que el máximo de $L_p$ sujeto al conjunto de restricciones $C_2$, se alcanza con los mismos valores $\vec{w}$,$b$ y $\alpha$ con los que se alcanza el mínimo de $L_p$ sujeto al conjunto $C_1$.

Con el primer requisito del conjunto $C_2$, que el gradiente de $L_p$ con respecto a $\vec{w}$ y $b$ desaparezca:

$$\frac{\partial}{\partial b} L(\vec{w}, b, \alpha) = 0; \frac{\partial}{\partial w} L(\vec{w}, b, \alpha) = 0 \tag{2.18}$$

Podemos obtener dos nuevas condiciones:

$$\begin{aligned} \vec{w} &= \sum_{i=1}^{l} \alpha_i y_i = 0 \\ \vec{w} &= \sum_{i=1}^{l} \alpha_i y_i \vec{x_i} \end{aligned} \tag{2.19}$$

que al sustituirlas en el problema original nos queda:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \vec{x_i} \vec{x_j} \tag{2.20}$$

Ahora trabajamos con un problema dual $(D)$. Para resolverlo recordamos que el vector solución, $\vec{w}$, puede escribirse en función de los vectores de entrenamiento, $x_i$, $\vec{w} = \sum_{i=1}^{l} \alpha_i y_i \vec{x_i}$. Cada vector de entrenamiento tendrá asociado un multiplicador de Lagrange,$\alpha_i$. El valor de este multiplicador será mayor que cero para los vectores que caigan en el hiperplano $H_1$ o $H_2$, para el resto de valores de entrenamiento el valor será por lo que no tendrán ninguna relevancia en el entrenamiento. [7] El hiperplano de separación dependerá sólo de los vectores soporte, las muestras más cercanas al límite entre ambas clases.

Una vez hemos calculado el hiperplano de separación debemos definir una forma de clasificar las muestras, la función $f(\vec{x_i} = \vec{w}\vec{x_i} + b$ media la distancia de cada vector al hiperplano de separación. Esta función será positiva para las muestras pertenecientes a la clase 1 y negativa para las de la clase -1, lo que permite clasificar cada muestra en su clase.

El alto nivel de ruido puede provocar cierto solapamiento entre muestras de ambas clases. Con la formulación vista hasta el momento, esas muestras no cumplirían la restricción $y_i(\vec{x_i}\vec{w} + b) - 1 >= 0 \forall i$ .

Lo que se suele hacer para afrontar problemas de este tipo es relajar las restricciones, para lo que se introduce un margen de error ,$\xi_{c,i}$. La restricción será ahora:

$$y_i(\vec{x_i}\vec{w} + b) >= 1 - \xi_{c,i} \tag{2.21}$$

Al añadir esta nueva variable pasaremos de uno a dos criterio a la hora de encontrar el hiperplano de separación:

- Maximizar el margen entre las clases, exactamente el criterio que ya teníamos antes.
- Minimizar la función de pérdidas que será proporcional a las muestras incorrectamente clasificadas.

La relevancia de un criterio frente al otro se controla a través de la variable $C$, que llamaremos coste.

La nueva función a maximizar será:

$$\begin{aligned} Maximizar : \tau(\overline{w}, \xi_c) = \frac{1}{2}||\overline{w}||^2 + C\frac{1}{l}\sum_{i=l}^{l}\xi_{c,j} \\ Sujeto : 0 <= \xi_{c,j} <= 1 - y_i f(\vec{x_i}) \end{aligned} \tag{2.22}$$

El primer término de la ecuación hace referencia a la minimización de $\vec{w}$ con vista a maximizar el margen entre clases, el segundo término tiene en cuenta las muestras incorrectamente clasificadas. $\xi_{c,j}$ será cero para las muestras correctamente clasificadas, $y_i f(\vec{x_i}) > 1$, es decir la etiqueta y la distancia coinciden en signo. Por otro lado, sera distinto de cero para las muestras incorrectamente clasificadas $y_i f(\vec{x_i}) < 1$. Cabe destacar que las muestras de una clase 1 que caigan entre el hiperplano de separación y el plano $H_1$ estarán clasificadas correctamente pero tendrán un $\xi_{c,j}$ asociado distinto de cero. De manera similar sucederá para las muestras de la clase -1. Estas muestras cumplirán:

$$0 < y_i f(\vec{x_i} < 1 \tag{2.23}$$

Se define a continuación la función de pérdidas del sistema que será en definitiva la suma de los márgenes, $\xi_{c,j}$, sumados a cada una de las muestras, podemos expresar la función de pérdidas de cada muestra de la siguiente manera:

$$f_{perdidas}(\vec{x_i}) = \text{máx}\{0, 1 - y_i f(\vec{x_i})\} \tag{2.24}$$

Hasta aquí se ha hablado de las maquinas de vectores soporte basadas clasificación partiendo de la premisa de que los datos de entrenamiento eran linealmente separables. Para comenzar el estudio con unos datos de entrenamiento no separables, hemos de tener en cuenta que los datos están en un espacio de dimensión $d$. Una de las formas mas habituales de hacerlos linealmente separables es llevar los datos a un espacio de características de dimensión mayor, $d' > d$. Para esto definimos una función que mapee cada vector de características del espacio de dimension $d$ al espacio de dimensión$d'$:

$$\varphi \cdot R^d \rightarrow R^{d'} \tag{2.25}$$

Dada la formulación llevada a cabo hasta ahora, los vectores sólo aparecen como producto internos en el espacio de características (esta es una de las causas por la que se emplea la formulación de Lagrange). Este hecho permite definir una función *kernel* que nos permita calcular el producto

interno de dos vectores sin necesidad de conocer explícitamente el vector mapeado en el espacio de características final. En definitiva las funciones kernel permiten que algoritmos lineales se apliquen a problemas no lineales.

A continuación se muestran las diferentes funciones kernel para $SVM$:

Kernel Gausiano:

$$k(x, x') = exp(-\frac{\|x - x'\|}{2 * \sigma^2}) \tag{2.26}$$

En este tipo de kernel es posible ajustar el valor de $\sigma$.

Kernel Polinomial:

$$k(x, x') = \langle x, x' \rangle^d \tag{2.27}$$

En este tipo de kernel se puede ajustar el grado del polinomio ( parámetro $d$).

Kernel RBF:

$$k(x, x') = f\left(d\left(x, x'\right)\right); d\colon metrica; f\colon R^+ \mapsto R \tag{2.28}$$

En este trabajo no entraremos en detalle en ninguno en particular, es posible encontrar multitud de libros dedicados en exclusiva a su estudio y diseño.[14]

Algunas de las características de las máquinas de vector soporte son:

- El entrenamiento es relativamente sencillo.
- No tienen un óptimo local, como ocurre con las redes neuronales.
- Se escalan relativamente bien en datos con un alto espacio de dimensiones.
- Es posible dar como entrada cadenas de caracteres o árboles, en lugar de vectores de características.

No obstante, se necesita que la función *kernel* sea "buena", en caso contrario, los resultados pueden ser bastante deficientes.

## 2.4. Algoritmo Rocchio

Este algoritmo es un algoritmo de expansión de consulta basado en realimentación de relevancia del usuario, propuesto por Rocchio. La expansión de la consulta en un sistema se hace usando diferentes técnicas entre ellas la realimentación de relevancia del usuario, la realimentación automática de relevancia , técnica morfológicas que procesen los términos de la consulta y

técnicas semánticas que muestran términos similares a los dirigidos por el usuario.

El algoritmo de Rocchio hace uso de la realimentación de relevancia del usuario. Esta requiere que el usuario marque los documentos como relevantes o no relevantes y luego, a cada nueva consulta del usuario se le agregan o quitan los términos que el sistema ha encontrado como relevantes, o no, en los documentos marcados. Rocchio propone la siguiente fórmula para generar la consulta expandida:

$$\overrightarrow{q_e} = \alpha \overrightarrow{q_i} + \frac{\beta}{|R|} \sum_{d \in R} \overrightarrow{d} - \frac{\gamma}{|R'|} \sum_{d \in R'} \overrightarrow{d} \tag{2.29}$$

donde $\overrightarrow{d_i}$ es la consulta inicialmente, $R$ es un conjunto de documentos relevantes, $R'$ es un conjunto de documentos no relevantes, $\alpha$,$\beta$ y $\gamma$ son parámetros de afinación del algoritmo, y $\overrightarrow{q_e}$ es la consulta expandida. Una valor de $\beta > \gamma$ otorga mas peso los documentos relevantes. Si $\gamma > \beta$ le da más peso a los documentos no relevantes. El valor $\alpha$ es la importancia que se otorga a la consulta anterior. Es recomendable un valor $\alpha$ ¿0.5,con $\beta+\gamma = 1$. típicamente se toman los valores $\alpha = 1$ ,$\beta = 0.75$ y $\gamma = 0.75$.

El algoritmo de Rocchio se resume con la fórmula anterior, y se pueden destacar varios elementos, el primero de ellos un conjunto de documentos evaluado como relevantes, otro conjunto de documentos evaluados como no relevantes y una consulta expresada como vector de términos con pesos. Los conjuntos de documentos evaluado como relevante y no relevantes también se deben expresar como vectores de términos con pesos. En la práctica, el algoritmo no registra todos los documentos que han sido relevantes y no relevantes para cada usuario del sistema, en lugar de ellos, el perfil almacena: un vector de términos representativo del documento relevante promedio, el total de documentos que han sido relevantes , un vector de términos representativo del documento no relevante promedio y el número total de documentos que han sido evaluado como no relevantes.

## 2.5. Algoritmo k-nn

Este probablemente sea uno de los clasificadores más conocidos, el algoritmo del vecino mas cercano. La idea sobre la que se fundamenta este paradigma es que un nuevo caso se va a clasificar en la clase más frecuente a la que pertenecen sus vecinos mas cercanos. Este método es un método de clasificación supervisada que sirve para estimar la función de densidad $F(x/C_j)$ de las predictoras $x$ por cada clase $C_j$. Es un método de clasificación no paramétrico, que estima el valor de la función de densidad de probabilidad directamente la probabilidad a posteriori de que un elemento

$x$ pertenezca a la clase $C_j$ a partir de la información proporcionada por el conjunto de prototipos.En el proceso de aprendizaje no se hace ninguna suposición acerca de la distribución de las variables predictoras.

Los ejemplos de entrenamiento son vectores en un espacio característico multidimensional, cada ejemplo descrito en términos de $p$ atributos considerando $q$ clases para la clasificación. Los valores de los atributos de i-esimo ejemplo se representan por el vector p-dimensional :

$$x_i = (x_{1i}, ..., x_{pi}) \in X \tag{2.30}$$

El espacio es particionado en regiones por localizaciones y etiquetas de los ejemplos. Un punto en el espacio es asignado a la clase C si esta es la clase más frecuente entre los, $k$, ejemplos de entrenamiento mas cercano. Normalmente para el calculo de la distancia se usa la distancia euclídia:

$$d(x_i, x_j) = \sqrt[2]{\sum_{r=1}^{p}(x_{ri} - x_{rj})^2} \tag{2.31}$$

Aunque para algunos casos se suele usar la distancia de *hamming*.

La fase de entrenamiento del algoritmo consiste en almacenar los vectores característicos y las etiquetas de las clases de los ejemplos de entrenamiento. En la fase de clasificación, la evaluación del ejemplo es representada por un vector en el espacio característico. Se calcula la distancia entre los vectores almacenados y el nuevo vector, y se seleccionan los k-ejemplos más cercanos. El nuevo ejemplo es clasificado con la clase que más se repite en los vectores seleccionados. Este métodos supone que los vecinos más cercanos nos dan la mejor clasificación; el problema de esta suposición es que es posible que se tengan muchos atributos y algunos irrelevantes que dominen sobre la clasificación: dos atributos relevantes perdería peso entre otros veinte irrelevantes. Para corregir el posible sesgo se puede asignar un peso a las distancia de cada atributo, dándole así mayor importancia a los atributos más relevantes. Otra posibilidad consiste en tratar de determinar o ajustar los pesos con ejemplos conocidos de entrenamiento. Finalmente antes de asignar pesos es recomendable identificar eliminar los atributos que se consideren irrelevantes. Podemos ver el método k-nn como dos algoritmos independientes:

- Algoritmo de entrenamiento: Para cada ejemplo $\langle x, f(x)\rangle$, donde $x \in X$, agregamos el ejemplo a la estructura representando los ejemplos de aprendizaje.

- Algoritmo de clasificación: Dado un ejemplar $x_q$ que debe ser clasificado, sean $x_1, ..., x_k$ los $k$ vecinos más cercano a $x_q$ en los ejemplos de aprendizaje, devolvemos:

$$\hat{f}(x) \leftarrow argmax_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i)) \tag{2.32}$$

donde el valor $\delta(a, b) = 1$ si $a = b$; y 0 en cualquier otro caso. El valor $\hat{f}(x)$ el valor devuelto por el algoritmo como un estimado $f(x_p)$ es solo el valor más común de $f$ entre los k-vecinos más cercanos a $x_p$, si elegimos el valor $k = 1$; entonces el vecino más cercano a $x_i$ determina su valor.

[27]

## 2.6. Técnicas multietiqueta

La tarea de clasificación se definió originalemente considerando que la correspondencia entre patrones y clases es unívoca, sin embargo, hay muchos problemas en los que la restricción de una etiqueta por patrón no se cumple. En estos problemas un patrón puede estar asociado simultáneamente con más de una clase, y se conoce a este paradigma como *Clasificación multi-Etiqueta*. La clasificación multi-etiqueta se caracteriza porque los conjuntos de clases no son excluyentes entre sí, y por tanto puede haber patrones a los que se les asocie más de una etiqueta a la vez. En estas técnicas las etiquetas asociadas a un patrón son etiquetas binarias, que indican si dicho patrón esta asociado o no a la etiqueta. Existen varias formas de enfrentar este tipo de problemas pero las técnicas más sencillas son la técnica de *uno-contra-resto* y la técnica de *uno-contra-uno*. Ambas técnicas van a transformar el problema original en problemas que puedan ser resueltos con clasificadores clásicos.

- **One versus rest (OVR)** : Un clasificador de este tipo se entrena para separar una clase del resto. El método estándar para descomponer un problema general de clasificación a partir de ejemplos en dicotomías consiste en generar un número de clasificadores igual al número de clases con las que trabajemos. Posteriormente se predice la clase de una muestra para todos los clasificadores. La clase asignada será aquella o aquellas con las que se consiguió mayor margen.

- **One versus one (OVO)** : Cada función clasificadora parcial, enfrenta una clase con otra, sin considerar las restantes clases. Con lo que se necesitan $k * (k-1)/2$ clasificadores,uno para cada una de los posibles pares de clases. En este caso se usa una estrategia de votación para clasificar: cada clasificador binario se considera como un voto y se tomará la clase o clases con mayor número de votos.

Estos dos técnicas nos permitirán resolver problemas multilabel de manera que podremos clasificar una instancia en más de una clase si fuese necesario.[9]

Para la realización de las pruebas y tras consultar documentación sobre estas dos técnicas trabajaremos con la estrategia OVR. Según la documentación consultada [13], la técnica OVO presenta algunas deficiencias a la hora de clasificar categorías que podríamos definir como difíciles. Este problema deriva de que cada clasificador dicotómico no toma en consideración todo el espacio de trabajo. Otro problema de la técnica OVO es que el número de clasificadores puede crecer de manera superlineal con respecto al número de posible clases.

# Capítulo 3

# Implementación del clasificador

En este capítulo se explicarán los diferentes pasos seguidos para la creación de un clasificador automático de documentos *XML*. Los algoritmos que hemos elegido para realizar las pruebas son: el algoritmo Random Forest, el algoritmo Naive Bayes y las máquinas de vectores soporte. Se han elegido estos porque son los más habituales para resolver este tipo de tareas.

Empezaremos a explicar los tres pasos básicos para crear clasificadores de documentos y con que herramientas se han resuelto junto a una justificación del por qué se han usado estas herramientas y no otras.

Los pasos de representación de la información y preparación de los datos se han llevado a cabo de igual forma independientemente del algoritmo que se haya usado, excepto, como se menciona más adelante, en la asignación de pesos para el clasificador Naive ya que su implementación así lo requería.

Dado que este trabajo no se centra en la implementación de un algoritmo particular, haré uso de librerías externas que proporcionen estos algoritmos.

Finalmente mostraré con un ejemplo el efecto producido al transformar un texto.

## 3.1. Fase de preparación de los datos

Como he comentado, para llevar a cabo la tarea de implementación, haré uso de distintas herramientas con las que resolveremos los diversos problemas que se nos presenten.

Para empezar, se nos plantea el problema de que lenguaje de programa-

ción usar de entre los muchos que se recomiendan para este tipo de tareas.

Últimamente el lenguaje **R** ha sido muy usado entre los desarrolladores de herramientas relacionadas con el *Aprendizaje Automático*, de hecho, en los últimos dos años es el que se ha adoptado para impartir esta materia en la *Universidad de Granada*.

Otros lenguajes muy usados para este tipo de aplicaciones son de más bajo nivel e incorporan librerías externas que pueden darnos una solución sencilla a la hora de realizar tareas de aprendizaje automático, de entre estos lenguajes de más bajo nivel podemos destacar el uso de **Java** y **C/C++**. Para facilitar la programación y reducir el tamaño del código, he elegido el lenguaje **Python** que aunque se trata de un lenguaje de tipo *Script* es posible usarlo como lenguaje orientado a objetos. Además existen implementaciones de **Python** de las principales librerías científicas que trabajan el problema de la clasificación así como para el tratamiento de textos.

La elección de *Python* como lenguaje de programación se debe a que, a pesar de no ser el lenguaje más usado durante la carrera, presenta algunas características que lo hacen un lenguaje útil y sencillo. Es un lenguaje fácil de aprender, con una sintaxis limpia y sencilla. Además el interprete interactivo de *Python* tiene un *Debugger* bastante explicativo. Otras características de *Python* es la facilidad para comprimir bucles en una sola sentencia, la facilidad de asignación de variables, debido a su tipado dinámico, y además permite la programación orientada a objetos.

### Lectura y reducción del texto

Para la lectura de las iniciativas, con las que vamos a trabajar, haré uso de una librería para la lectura de documentos *XML*, llamada **DOM**. Junto a *SAX*, *DOM* es la herramienta más usada para analizar documentos especificados en lenguajes *XML* y definir la estructura de un documento, aunque existen muchas otras. Una de las diferencias entre estas librerías es que DOM verifica que el documento esté bien formado de acuerdo a las reglas *XML*. Para *DOM* un documento *XML* es un árbol de nodos, donde cada nodo representa a un elemento *XML*. Los elementos hijos y el texto contenido dentro de un elemento son subnodos. **DOM** es un estándar *W3C* (*World Wide Web Consortium* ) .

La principal ventaja de *DOM* es que permite acceder a datos en función de la jerarquía de elementos, crear documentos desde cero o modificar el contenido. El problema es que lee el documento y lo almacena en memoria, lo que para documentos largos puede suponer un problema por lo que no se recomienda para dispositivos empotrados con poca memoria y baja capacidad de cómputo, lo cuál en un principio no es un problema para nuestro

trabajo.

Una vez leídas las iniciativas y almacenado el contenido de estas se nos plantea el problema de como trabajar con este texto ya que debemos adaptarlo para nuestros intereses.

Como ya comentamos anteriormente, es recomendable reducir el tamaño de los documentos con los que vamos a tratar. Para este cometido vamos ha hacer uso de la librería *Summa*. Esta librería permite reducir el tamaño de los textos realizando resúmenes de estos.

Esta librería se basa en el algoritmo *TextRank*, que hace uso del famoso algoritmo de ordenamiento *PageRank*, una de las claves del éxito de *Google*. **PageRank** es utilizado para medir la importancia de cualquier página Web en Internet en función de los enlaces que dicha página recibe. La idea básica de TextRank es aplicar un algoritmo de ordenación basado en grafos a problemas relacionados con el procesamiento del lenguaje natural. La formalización del algoritmo *PageRank* es simple; dado un grafo $G = (V, E)$, donde $V$ es el conjunto de vértices y $E$ un conjunto de arcos (o aristas) dirigidos entre dos vértices, se definen en primer lugar $E(Vi)$ y $S(Vi)$ que calculan , respectivamente, el número de arcos que entran o salen del vértice $Vi$. A partir de estas dos operaciones se define la puntuación (o *PageRank*) de un determinado vértice con la siguiente fórmula:

$$P(V_i) = (1 - d) + d_j \cdot E(V_j) \cdot \frac{1}{|S(V_j)| \cdot P(V_j)} \tag{3.1}$$

Donde $d$ es un factor de amortiguación que tiene como objetivo incluir en el modelo la probabilidad de que haya un salto aleatorio de un vértice del grafo a cualquier otro. Partiendo de valores arbitrarios para las puntuaciones de los nodos de un grafo, se alcanza un punto de convergencia aplicando iterativamente la fórmula hasta que la mayor diferencia de puntuaciones obtenidas para cada nodo, entre dos iteraciones, es menor que un determinado umbral. Una vez finalizado el algoritmo , la puntuación alcanzada por cada nodo representa la importancia del mismo , y puede ser usada como criterio de toma de decisiones.

Este algoritmo se adapta a tareas de *PLN* (*procesamiento del lenguaje natural*), y es lo que se conoce como algoritmo **TextRank**. El algoritmo TextRank esencialmente ejecuta PageRank en un grafo. La creación del grafo se realiza usando un conjunto de unidades de texto como vértices. Las aristas son basadas en alguna medida de semántica o similiridad léxica entre las unidades de texto de los vértices. A diferencia de PageRank, las aristas suelen ser no dirigidas y se les puede asignar pesos para reflejar un grado de similiridad. Una vez se construye el grafo, se usa para formar una matriz estocástica, y el ranking existente sobre los vértices se obtiene al encontrar

el vector propio correspondiente al valor propio de 1.

Los vértices suelen ser palabras individuales y las aristas se crean basándose en la coocurrencia de palabras. Dos vértices están conectados por una aristas si las palabras aparecen dentro de una ventana de tamaño N en el texto original, el tamaño N suele determinarse tomarse como un valor que agrupe de 2 a 10 palabras.

Para formar las frases finales, debido a que el algoritmo simplemente da un valor a cada vértice individualmente, es necesario determinar un umbral o producir una cantidad limitada de frases. La técnica con la que se resuelve este problema consiste en definir un contador T que corresponde a una fracción definida por el usuario del total del número de vértices en el grafo.

Este algoritmo funciona correctamente gracias a que una palabra que aparece múltiples veces a través de un texto puede tener múltiples vecinos coocurrente, por lo que el grafo coocurrente contendrá regiones conectadas densamente para términos que parecen a menudo y en diferentes contextos. Un recorrido aleatorio por este grafo tendrá una distribución estacionaria que asigna largas probabilidades a términos que se encuentran en el centro de los conjuntos , lo cual es similar a la forma como las páginas Web se encuentran conectadas densamente, razón por la cual el algoritmo PageRank se puede aplicar a un grafo.[25]

La elección de *Summa* viene dada por una serie de factores que la hacen especialmente útil para resolver nuestro problema; en primer lugar nos permitirá generar distintos tamaños del contenido de cada iniciativa en función del número de términos o de la proporción que deseamos. También es interesante que se implementa para trabajar con distintos idiomas con lo que no habrá ningún problema a la hora de usarlo para el lenguaje castellano.

Esta reducción del texto nos servirá también para comprobar que cantidad de este permite obtener mejores resultados a la hora de ser usado en la clasificación. Para esto trabajaremos con distintos tamaños de texto, que variarán tanto en el contenido como en la parte del documento a la que pertenecen. Trabajaremos con diferentes tamaños del contenido de los *párrafos*, así como con el contenido del *extracto*.

### Tokenización, Stop Word y Stemming

Una vez con el contenido deseado de cada iniciativa se nos plantea un nuevo problema que es: como trabajar con el texto.

Para trabajar con el texto haré uso de una librería disponible para **Python**, y de tipo *OpenSource*, llamada **NLTK**.

*NLTK* es una herramienta muy potente para el *Procesamiento del Len-*

*guaje natural*. Fue recomienda por mis tutores y tras investigar un poco sobre ella, cabe destacar, que se esta usando ahora mismo como estándar en el *procesamiento de datos*, también es frecuente su uso en lo que se conoce como *Minería de opinión*. Su principal ventaja es que incorpora todas las herramientas que necesitaremos para realizar un análisis del texto. De entre los módulos de *NLTK* que vamos a usar están: *StopWords*, *Snowball* y *WordTokenize*.

El módulo *WordTokenize* provee de herramientas para realizar una tokenización del contenido de un documento y además permite hacerlo en diferentes idiomas.

El módulo *Stop Words* se usa para la eliminación de palabras vacías, que son aquellas palabras que no aportan ningún tipo de significado al documento. Aunque el módulo *Stop Words* viene implementado en la librería *NLTK*, y que es posible seleccionar de entre una serie de idiomas, no está del todo completo y es conveniente realizar esta tarea incluyendo algún tipo de diccionario donde se incluyan palabras que son susceptibles de ser eliminadas dado su baja importancia. Estas palabras vacías en principio no son un problema porque el algoritmo para el cálculo de las frecuencias de cada término dentro del documento (*Tf-idf* en nuestro caso) las desestimará por ser palabras de uso muy común como por ejemplo preposiciones o los artículos.

Por último, el módulo *SnowBall* implementa un *Stemmer*. Esta herramienta se usará para encontrar coincidencias en los lexemas de palabras que al final vienen a tener el mismo significado. La elección de este *Stemmer*, no se basa únicamente en que esta incluido en la librería **NLTK**, si no a la gran popularidad de esta herramienta en el análisis de datos, además *SnowBall* presenta una multitud de idiomas que permiten tratar textos escritos en diferentes lenguas.

*SnowBall* es implementa el conocido algoritmo de Porter. El algoritmo de Porter permite extraer los sufijos y prefijos comunes de palabras literalmente diferentes pero con una raíz común y que puedan ser considerados como un mismo término. En la referencia [1] podemos encontrar información detallada sobre diversas formas de realizar la tarea del Stemming. Los algoritmos de Stemming o Lematización para los que hablamos castellano , son conocidos desde los años 70, y podemos destacar como los más relevantes el algoritmo de Lovins(1968), el algoritmo de Porter (1980) y el algoritmo de Paice (1990).

Todos ellos eliminan los sufijos de las palabras de forma iterativa, y requieren de una serie de pasos para llegar a la raíz de un término, pero no requieren a priori conocer todas las posibles terminaciones.

El algoritmo que implementa la libreria NLTK es el algoritmo de Porter. Básicamente el algoritmo de Porter lee un archivo, toma una serie de caracteres, y de esa serie una palabra; luego válida que todos los caracteres

de la palabra sean letras y finalemente aplica la lematización. Este lematizador hace pasar una palabra por varios conjuntos de reglas, cada uno de estos conjunto formado por un numero determinado de reglas y cada regla constituida por:

- Un identificador para la regla.
- Un sufijo a identificar.
- Un texto por el que se reemplaza el sufijo.
- El tamaño del texto de reemplazo
- El tamaño mínimo que debe tener la raíz resultante después de aplicar esta regla.
- Una función de validación que verifique si se debe aplicar la función una vez encontrado el sufijo.

Cuando se han aplicado estas reglas se devuelve la palabra resultante. Para la selección de los grupos de reglas se ha de tener en cuenta que dos sufijos que ocurren juntos no pueden pertenecer al mismo conjunto, que las reglas que quiten sufijos más al final de la palabra deben ser procesador antes que otros., y por ultimo hay que tener en cuenta que un sufijo que aparece siempre que ocurra otro, es condicional a la aparición del anterior.

A parte de esto hay que tener en cuenta que existen reglas propias para cada idioma. Por ejemplo en castellano, el sufijo *nos* no es sufijo en palabras como *campesinos*, *caminos*,..., pero si lo es en *hacernos*, *presentarnos*,...

La herramienta NLTK nos proporciona un stemmer basado en el algoritmo de Porter en el que se han definido reglas para diferentes idiomas incluidos el castellano.

### Asignación de pesos

Antes de ponderar con pesos los elementos restantes tras las fases anteriores, debemos representarlos de forma numérica. Durante este proceso se generá una matriz de conteo para cada documento de la colección. Para construir esta matriz se hace uso de la función *CountVectorizer* de *Scikit Learn*. Este proceso asigna un valor númerico a cada palabra distinta y cálcula el número de veces que este se repite. Tras el paso anterior ya tenemos representada la información del contenido de cada iniciativa en forma de matriz, nuestro siguiente paso será determinar como de relevante es cada término dentro de nuestro documento y dentro de la colección en general. Para esto volvemos ha hacer uso de *Scikit Learn*, en particular la función *TfidfTransformer*, con la que podemos transformar la matriz de conteo en un *Tf-idf*(

del inglés *Term frequency- Inverse document frequecy*), frecuencia inversa del documento, que es una medida que nos dice como de importante es un término tanto para el documento en el que pertenece como para el conjunto del documentos. Este método es de uso muy frecuente en la recuperación de información y en la clasificación documental. El valor *Tf-idf* asociado a un término aumenta propocionalmente al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia para la colección de documentos, esto es muy útil ya que nos permite conocer el hecho de que alguna palabras generalmente son más comunes que otras.

Este último paso no es mas que la ponderación de los términos que tiene como finalidad conocer la importancia de los términos para representar un documento y permitir su posterior recuperación. Implica que debemos determinar la capacidad de los términos para representar el contenido de los documentos de la colección, que permitan identificar cuáles son relevantes o no, dentro de cada iniciativa. Al valor e índice que es capaz de determinar este término se le denomina *peso del término* o *ponderación del término* y su cálculo implica determinar la *Frecuencia de aparición del término* (**TF**) y la *Frecuencia inversa del documento para un término* (**IDF**).

El término **TF** o frecuencia de aparición del término (del inglés *Term Frequency*), es la suma de todas las ocurrencias ó el número de veces que aparece un término en un documento. A este tipo de frecuencia de aparición también se le denomina *Frecuencia de aparición relativa* porque atañe a un documento en concreto y no a toda la colección. Una frecuencia de aparición baja en un término indica una representatividad elevada, por el contrario si esta frecuencia es alta para un término su representatividad en baja.

El cálculo del valor TF para un término $n$ y un documento $D1$ es el número de veces que aparece el término en el documento.

El factor **IDF**, *Frecuencia Inversa del Documento para un Término* del inglés *Inverse Document Frequency*, de un término es inversamente proporcional al número de documentos en los que aparece dicho término. Esto significa que cuanto menor sea la cantidad de documentos, así como la frecuencia absoluta de aparición del término, mayor será su factor **IDF** y a la inversa, cuanto mayor sea la frecuencia absoluta relativa a una alta presencia en todos los documentos de la colección, menor será su factor discriminatorio. Un término genérico aparecerá en la mayoría de los documentos y su poder discriminatorio sera muy bajo. En cambio, un término que aparezca en pocos documentos su poder discriminatorio será alto.

El factor **IDF** es único para cada término de la colección. Para su cálculo dado un término **n** se realiza aplicando el logaritmo en base 10 de **N** (donde **N** será el número total de documentos de la colección) dividido entre la *Frecuencia de documentos para un término n dado.* Al valor resultante se le suma 1 para corregir los valores para los términos con **IDF** muy bajos. La

función que muestra el cálculo del valor IDF esta la siguiente:

$$IDF_n = \log_{10} \frac{N}{DF_n} + 1 \tag{3.2}$$

Donde DF representa al número de documentos en los que aparece el término ($n$) a lo largo de toda la colección de documentos.

El peso de un término (*Td-idf*) en un documento es el producto de su frecuencia de aparición en dicho documento (TF) y su frecuencia inversa de documentos (IDF).

Y se representa de la siguiente manera:

$$TF - IDF_{(n,d)} = TF_{(n,d)} \cdot IDF_n \tag{3.3}$$

Esto significa que el peso o ponderación se calcula para cada término en cada documento. Se puede comprobar que cada término tiene frecuencias distintas en cada documento. Los pesos obtenidos son denotativos de la importancia del término en cada documento.

Una vez determinada la relevancia de cada término disponemos de todo para empezar a entrenar el clasificador.

### Ejemplo del procesamiento de un texto

En esta fase nos dedicamos a transformar los datos de entrada, que en nuestro caso son textos, a una representación apropiada para los algoritmos de clasificación que trataremos.

Para realizar distintas pruebas con cada algoritmo de clasificación usaremos distinto contenido del presente en cada iniciativa. Este contenido estará formado por el extracto y/o por el contenido completo de los párrafos o el contenido de los párrafos resumido.

Por tanto, como primer paso, para solucionar el problema de la clasificación, extraeremos la información que nos es útil para el aprendizaje. Diferenciaremos entre el contenido de los párrafos (texto contenido dentro de la marca *parrafo* y el del extracto (texto contenido dentro de la marca *extracto*) para realizar las distintas pruebas.

Para leer el contenido de las iniciativas haremos uso de la herramienta *DOM*, que nos permite leer el contenido del fichero diferenciando entre las distintas marcas que lo forman. Una vez leída la información de las marcas *extracto*, *parrafo* y *materias*, pasaremos a trabajar esta información. El contenido de las *materias* requiere de un tratamiento distinto ya que suprimiremos diferentes etapas que aplicamos al texto de las marcas *extracto* y *parrafo*.

Para trabajar con el texto debemos definir el contenido de información que vamos a usar para cada una de las pruebas. Independientemente del algoritmo que usemos para clasificar se realizarán distintas pruebas con distinto volumen de contenido para cada algoritmo. Por tanto, nuestro primer paso será determinar que contenido usaremos.

Para realizar las pruebas vamos hacer uso de distinto contenido de las iniciativas. En una primera prueba trabajaremos con el texto contenido en los párrafos de forma integra, es decir, sin realizar modificaciones en su tamaño, en una segunda prueba reduciremos el tamaño del texto hasta la mitad, en la tercera prueba se usara el 25 % del texto, en una cuarta prueba usaremos únicamente el contenido de la marca extracto y por último haremos una prueba con el contenido del extracto añadiéndole un resumen del contenido de los párrafos de un 25 %.

Para realizar estos resúmenes haremos uso de la herramienta *SUMMA*, la cuál ha sido explicada en el apartado anterior. Con las funciones de esta librería crearemos distintos resúmenes de cada una de las iniciativas que variarán en la cantidad de información almacenada en cada una de ellas. Acabado esto tendremos para una iniciativa diferentes archivos con distinto contenido de ella, este contenido será tratado de manera individual y serán usados para las pruebas.

Este contenido por iniciativa, sea cual sea su tamaño, deberemos tokenizarlo. Para esto haremos uso de la herramienta *NLTK*. La tokenización nos proporcionará una lista con todos los términos del contenido de la iniciativa que vamos a tratar. En esta lista de palabras que hemos obtenido tras la tokenización debemos realizar algunos ajuste. En primer lugar, eliminaremos las palabras o términos que no aporten ningún valor al contenido de la iniciativa, este tipo de palabras se denominan palabras vacías y carecen de significado, suelen ser artículos, pronombres, preposiciones,...

Este último proceso reducirá el tamaño el contenido con el que estamos trabajando sin modificar su significado.

A los términos que restan después de aplicar la tokenización y la eliminación de palabras vacías vamos a aplicarles un paso más. Este último paso,el *stemming*, este proceso consiste en la identificación de una raíz (*stem*), para lo que utilizamos *SnowBall*, que es el *stemmer* que mejores resultados obtiene para el lenguaje español según el artículo de *Técnicas de Procesamiento del Lenguaje Natural en la Recuperación de Información* , realizado por el Centro de Investigación sobre Tecnología de Lingüística [24], y que como todas las herramientas usadas en de licencia libre y está disponible en [2].

Todo el trabajo hecho hasta este punto sirve para realizar una limpieza del texto, de la que deben desaparecer; signos de puntuación y palabras sin significado. Además el uso de un *stemmer* nos ayuda a identificar palabras que poseen un significado similar.

Antes de continuar voy a mostrar un ejemplo el proceso de preparación de los datos, paso previo a la asignación de pesos a los términos.

Tendremos por tanto un texto original, extraído de los párrafos de una de nuestras iniciativas ,extraído de una de las iniciativas con las que contamos. En nuestro caso de ejemplo el texto es el siguiente:

*El texto dice: «Las Protegidas son documentos que sirven para entender la evolución de la arquitectura andaluza en un momento de transición entre el periodo anterior a la guerra y la posterior arquitectura de los años cincuenta, base de la actual arquitectura contemporánea española. Tienen, por tanto, un valor histórico». Continúa más adelante diciendo: «Su valor artístico se basa en la novedad y en la expresión de sus funciones, con una buscada e intencionada voluntad, de forma que les otorga un valor de contemporaneidad a considerar en su conservación. Todos estos valores los hacen susceptibles de ser considerados como patrimonio por una comunidad».*

*Y eso es lo que hemos hecho, señorías: valor artístico, valor histórico, valor documental, y —permítanme que, si es posible, le añada otro— valor simbólico: el valor simbólico que tienen estos edificios como muestra de los primeros pasos que nuestra comunidad dio en el viaje en el que nos hemos dejado llevar hasta una región, una comunidad, plenamente integrada social, económica, artística y culturalmente en el escenario europeo y en el escenario internacional. Todo eso significa paran nosotros Las Protegidas, y por todo ello la hemos vuelto a proteger.*

*«Si se nos permite una valoración enteramente subjetiva, aunque creemos que, evidentemente, necesaria, nadie se explica hoy en Jaén cómo han podido inscribir como parte del patrimonio histórico andaluz al conjunto de las Viviendas Protegidas, zona absolutamente degradada en el entorno de la ciudad y que, precisamente por su especial ubicación, lo que necesitaría sería una reedificación total, con la reestructuración integral del barrio. De ahí que en su momento causó cierta sorpresa la decisión de iniciar, incoar en 2004, el expediente de catalogación, cuando, un año antes, parte de una de las manzanas catalogadas, propiedad íntegra de la Diputación Provincial de Jaén, gobernada por el mismo partido político que la Administración autonómica, fue demolida, sin que absolutamente nadie, ni asociaciones de amigos de los íberos, ni administraciones, ni Junta de Andalucía» —usted, señora Consejera, usted— «opusieran reparo algo, y cuando el edificio colindante a las que ahora se catalogan, la Escuela de Peritos de Jaén, con un indudable aire de arquitectura modernista, había sido permutado por la Administración autonómica a un centro comercial para su inmediata demolición, sin que tampoco nadie opusiera reproche técnico o reproche jurídico alguno. Entenderá la sala» —dicen los propietarios de estas viviendas— «que tenemos el honor de dirigirnos simple y llanamente con la idea de que esto*

*nos ha causado indignación.»*

Como ya se ha comentado, *Summa* da la posibilidad de resumir un texto indicando el número de palabras que contiene o definiendo el porcentaje de texto que se desea obtener de resumen. He preferido hacer uso de la segunda opción, definiendo un porcentaje para obtener los diferentes tamaños de resumen con los que se van a realizar las pruebas.

Del texto original vamos a mostrar un resumen del 50 %, que quedaría como se muestra a continuación:

*El texto dice: «Las Protegidas son documentos que sirven para entender la evolución de la arquitectura andaluza en un momento de transición entre el periodo anterior a la guerra y la posterior arquitectura de los años cincuenta, base de la actual arquitectura contemporánea española.*

*«Si se nos permite una valoración enteramente subjetiva, aunque creemos que, evidentemente, necesaria, nadie se explica hoy en Jaén cómo han podido inscribir como parte del patrimonio histórico andaluz al conjunto de las Viviendas Protegidas, zona absolutamente degradada en el entorno de la ciudad y que, precisamente por su especial ubicación, lo que necesitaría sería una reedificación total, con la reestructuración integral del barrio.*

*De ahí que en su momento causó cierta sorpresa la decisión de iniciar, incoar en 2004, el expediente de catalogación, cuando, un año antes, parte de una de las manzanas catalogadas, propiedad íntegra de la Diputación Provincial de Jaén, gobernada por el mismo partido político que la Administración autonómica, fue demolida, sin que absolutamente nadie, ni asociaciones de amigos de los íberos, ni administraciones, ni Junta de Andalucía» —usted, señora Consejera, usted— «opusieran reparo algo, y cuando el edificio colindante a las que ahora se catalogan, la Escuela de Peritos de Jaén, con un indudable aire de arquitectura modernista, había sido permutado por la Administración autonómica a un centro comercial para su inmediata demolición, sin que tampoco nadie opusiera reproche técnico o reproche jurídico alguno.*

Como observamos esta reducción del tamaño del texto no modifica las palabras, ni elimina los símbolos de puntuación, únicamente reduce el contenido del texto original. Posteriormente aplicamos al resultado obtenido la tokenización, este proceso lo único que realizaría sería separar los diferentes tokens contenidos en el resumen, como ejemplo mostramos el primer párrafo de los obtenidos en el resumen anterior que quedaría de la siguiente manera:

*'El', 'texto', 'dice', ':', 'Las', 'Protegidas', 'son', 'documentos', 'que', 'sirven', 'para', 'entender', 'la', 'evolución', 'de', 'la', 'arquitectura', 'andaluza', 'en', 'un', 'momento', 'de', 'transición', 'entre', 'el', 'periodo', 'anterior', 'a',*

*'la', 'guerra', 'y', 'la', 'posterior', 'arquitectura', 'de', 'los', 'años', 'cincuenta', ',', 'base', 'de', 'la', 'actual', 'arquitectura', 'contemporánea', 'española', '.'*

El resultado anterior representa un array de String, donde cada elemento corresponde a una palabra.

A continuación, debemos eliminar las palabras vacías para posteriormente aplicar el proceso de *Stemming*.

Eliminando palabras vacías el resultado sería el siguiente:

*'texto', 'dice', 'Protegidas', 'documentos', 'sirven', 'entender', 'evolución', 'arquitectura', 'andaluza', 'momento', 'transición', 'periodo', 'anterior', 'guerra', 'posterior', 'arquitectura', 'andaluza', 'cincuenta', 'base', 'actual', 'arquitectura', 'contemporánea', 'española'*

Y aplicando el *Stemmer*, el resultado sería:

*text, dic, proteg, document, sirv, entend, evolu, arquitectur, moment, transicion, period, anterior, guerr, posterior, arquitectur, andaluz, cincuent, bas, actual, conteporane, español*

Una vez realizados los pasos anteriores el contenido de nuestras iniciativas se verá reducido y simplificado, lo que debe reducir considerablemente el sobreajuste que podría darse debido al alto contenido de términos de cada una de las iniciativas. Insisto, estos procesos se realizan independientemente del tamaño extraído de las iniciativas.

Ahora es cuando comenzamos a representar los datos de forma que puedan ser usados para la clasificación. Esta fase es la primera de las tres en que suele dividirse el proceso del aprendizaje automático, seguida de la *fase de entrenamiento* y de la *fase de clasificación*. En la fase de preparación de los datos haremos uso de la librería *Scikit-Learn*, la cual provee de distintas funciones que serán de gran ayuda.

En primer lugar, debemos hacer un recuento de cada una de las apariciones de cada término dentro de nuestro documento y representarlos de igual forma para cada iniciativa, este proceso genera una matriz conocida como *matriz de conteo*. Para construir la *matriz de conteo* he hecho uso de una función de *Scikit Learn* llamada *CountVectorizer* que realiza precisamente esta función. Aunque no se ha mencionado, *CountVectorizer* asigna un valor numérico a cada palabra distinta, además del recuento de estas.

Posteriormente se procederá calcular los pesos a partir de la técnica TF-IDF. Con esta ponderación ya conocemos la importancia de los términos

para representar un documento.

En el caso particular de implementación del clasificador para el caso del algoritmo de Naive Bayes es necesario realizar un paso posterior al cálculo de los pesos *TF-ITF*. Este paso consiste en realizar una *Dense Transaformation*, esto es transformar la matriz en un *Dense Matrix*.

En el análisis numérico, una matriz se considera *Dense Matrix* si la mayoría de elementos de esta son distintos de cero. En caso de que la mayoría sean cero se denomina a la matriz *Sparse Matrix*.
En cuestión, el problema no es que los valores sean cero, el problema está en que en las *Sparse matrix* sólo se almacenan los valores que no son cero, dando a entender que los valores restantes son cero. Por esto mismo, muchos algoritmos no funcionan con matrices *Sparse*.

*Python* y *Scikit-Learn* trabajan comúnmente con matrices *Sparse*, con el objetivo de reducir el espacio de almacenamiento de la matriz. Este tipo de matrices trabaja con los algoritmos de *Scikit-learn*; *Random Forest* y *SVM* ( incluyendo la versión de *Libsvm*), en cambio para la implementación del clasificador Naive es necesario dar de entrada una representación *Dense* de la matriz de pesos *Tf-itf*.[10][26]

## 3.2. Fase de entrenamiento

Durante esta fase el objetivo es hacer que el clasificador aprenda de la información procesada previamente a ser capaz de distinguir entre diferentes clases. En esta fase de entrenamiento se recibe la información ya depurada y organizada, que la obtenemos después de realizar la transformación descrita en la sección anterior,y se construye un modelo.

Con el conjunto de entrenamiento en un formato correcto para ser procesado por el clasificador, este deberá inferir una función con la que poder predecir el valor de una instancia.

El entrenamiento depende del algoritmo que usemos para la clasificación como ya vimos en el *Capítulo 2*.

Para la tarea de la clasificación existen multitud de librerías que proporcionan algoritmos de aprendizaje automático, útiles para tareas de clasificación. Entre estas herramientas hay que destacar *Scikit- Learn*. *Scikit-Learn* es una de esas librerías *"MILAGRO"* en la que todo lo que se pueda necesitar está implementado. Se barajó la posibilidad de usar *Libsvm* para crear nuestro clasificador pero esto hubiese sido mucho más tedioso. El problema de *Libsvm* es que no incorpora clasificación *multilabel*, en cambio *Scikit-Learn* si lo hace, además, *Scikit-Learn* tiene mas módulos que nos facilitan la tarea de otorgar pesos a los términos, algo que *Libsvm* no pose.

Otro motivo fundamental para el uso de *Scikit* es que incorpora multitud

de algoritmos de clasificación, por el contrario *Libsvm* sólo incorpora el *SVM*.

En particular, para la clasificación multilabel haremos uso de *OneVsRest*. *OneVsRest* es un técnica *multiclase/mutlilabel*, utilizado normalmente para *SVMs*, también conocido por OVA (de las siglas en ingles *One versus All)*. Básicamente se compone de la construcción de una SVM por clase, que es entrenado para distinguir las muestras de una clase de las muestras de todas las clases restantes. Por lo general, la clasificación de un patrón desconocido se realiza de acuerdo a la puntuación máxima entre todos los SVMs.

Los pasos seguidos por este tipo de estrategia son los siguientes:

- Resolver $k$ problemas binarios diferentes, y clasificar la clase $K$ con el resto de las clases
- Asignar un test simple a la clase que de el mayor $fk(x)$ valor, donde $fk(x)$ es la solución *k-ésima*, y determina el margen con el que ha sido clasificado el dato. Debemos quedarnos con este.

Esta fase se realizará para cada algoritmo, de igual forma para todos ellos.

El resto de herramientas que voy a usar para la implementación como la validación cruzada y la evaluación, para problemas multilabel, de los resultados no están disponibles en la actualidad en ninguna librería que haya podido encontrar, por lo que habrá que desarrollarlas.

## 3.3. Fase de clasificación

En esta fase se pondrá a prueba el clasificador. En este paso se pasarán al clasificador nuevas iniciativas,después de procesarlas de igual forma que al conjunto de entrenamiento, y el determinará las etiquetas en las que clasificar dichas iniciativas.

Estas entradas en el clasificador deberán ser tratadas como hemos tratado los documentos que nos han servido para entrenar el clasificador, dependiendo del contenido que vamos a utilizar de cada documento.

# Capítulo 4

# Evaluación de los resultados

Para poder evaluar los clasificadores implementados para cada algoritmo debemos dividir nuestra colección de documentos de manera que podamos usar unos para entrenar y otros para clasificar. De manera que la función de aproximación sólo ajusta con el conjunto de datos de entrenamiento y a partir de aquí calcula los valores de salida para el conjunto de datos de prueba. Esto implica que en gran media la evaluación dependa de cómo es la división entre datos de entrenamiento y de prueba, y por tanto los resultados obtenidos pueden ser diferentes dependiendo de como se haga la división. Debido a este problema aparece el concepto de *validación cruzada.*

Existen varios tipos de validación cruzada. Para este trabajo se ha utilizado la validación cruzada de $K$ iteraciones o *kfolds*. En este tipo de validación los datos de muestra se dividen en $K$ conjuntos. Uno de ellos se usa como datos de entrenamiento y el resto (K-1) como datos de entrenamiento. Este proceso se repite durante $k$ iteraciones, con cada uno de los posibles subconjuntos de datos de prueba. finalmente se realiza la media aritmética de los resultados de cada iteración para obtener un único resultado. Este método es muy preciso puesto que evaluamos a partir de $K$ combinaciones de datos de entrenamiento y de prueba, aunque tiene el problema de ser lento desde el punto de vista computacional. En la práctica el número de iteraciones depende de la medida del conjunto de datos.[21][11]

Se usa normalmente en entornos donde el objetivo principal es la predicción y donde se quiere estimar como de preciso es un modelo. Para este trabajo concretamente el número de particiones ha sido 7. Haciendo uso de una de ellas para predecir y el resto (6) para entrenar. Se ha repetido el proceso 7 veces obteniendo distintos resultados. Como valor final se ha calculado la media aritmética de los 7 resultados.

Antes de empezar a evaluar los resultados de los distintos clasificadores creados con los diferentes algoritmos de aprendizaje automático que hemos visto el *capítulo 3*, debemos introducir el significado y la forma en que se

toman estas medidas.

## 4.1. Métricas usadas

Para evaluar los datos de los diferentes clasificadores se tomarán como medidas de evaluación la *precisión*, la *exhaustividad* y el medida $F1$ que es una combinación de ellas. La precisión se toma como:

$$\frac{TP}{(TP + FP)} \tag{4.1}$$

, donde $TP$ es el número de verdaderos positivos, $FP$ el número de falsos positivos. La precisión es intuitivamente la capacidad del clasificador de no etiquetar una muestra como positiva que es negativa. El *recall* o *exhaustividad* es la relación de:

$$\frac{TP}{TP + FN} \tag{4.2}$$

,donde $TP$ es el número de verdaderos positivos y $FN$ los falsos negativos. De manera intuitiva puede definirse como la capacidad del clasificador para encontrar todas las muestras positivas. Por último el valor $F1$ que no es más que una relación entre *precision* y *exhaustividad* que se representa como:

$$2 * [\frac{Precision * Exhaustividad}{Precision + Exhaustividad}] \tag{4.3}$$

, los valores de $F1$ cercanos a la unidad indican que se está dando la misma ponderación a la $Precision$ que a la $Exhaustividad$.

En problemas como el nuestro, con varias clases por muestra, no existe una manera de única de evaluar un clasificador, dando lugar a diferentes criterios. Nosotros usaremos un cálculo *micro* y otro *macro*. El método *micro* resume los valores del sistema para diferentes conjuntos y los aplica para obtener las estadísticas. O sea, se calculan las $TP$, $FP$ y $FN$ para subconjuntos diferentes, este resultado se suma para calcular el *recall* y la *precision*:

$$Micro - precision = \frac{TP1 + TP2}{TP1 + TP2 + FP1 + FP2} \tag{4.4}$$

$$Micro - recall = \frac{TP1 + TP2}{TP1 + TP2 + FN1 + FN2} \tag{4.5}$$

En el caso de la macro basta con hacer la media de la *precision* y el *recall* del sistema en diferentes conjuntos.

$$Macro - Precision = \frac{\sum_{i=1}^{n} P_i}{n} \tag{4.6}$$

$$Macro - Recall = \frac{\sum_{i=1}^{n} R_i}{n} \tag{4.7}$$

Los valores $Macro - F1$ y $Micro - F1$ se calcularán con la función descrita anteriormente.

Esta medida es la mas simple e intuitiva, computamos el rendimiento de cada muestra y lo promediamos. La medida *macro* puede utilizarse cuando se quiere saber cómo de bueno es el sistema a través de los conjuntos de datos. No se puede llegar a una decisión sobre el clasificador únicamente con esta medida. Por otro lado la *micro* puede ser una medida útil cuando el conjunto de datos varia de tamaño.

Con las medidas *micro* y *macro* podemos comprobar de que forma se obtienen mejores resultados para así ir mejorando el desempeño general del clasificador.[22][16]

## 4.2. Resultados con Random Forest

A la hora de evaluar los resultados he empezado por los clasificadores generados a partir del algoritmo RandomForest.

En las pruebas vamos a diferenciar por el contenido extraído de cada iniciativa para usar como información de entrenamiento en los clasificadores. Así pues, he realizado pruebas con el extracto, con toda la iniciativa, con el 50 % del contenido de los párrafos, con el 25 % y con el extracto más el 25 %.

RandomForest nos ofrece pocos parámetros para gestionar el entrenamiento. Entre estos es interesante la importancia del número de estimadores que usarán para predecir una clase. Se han realizado distintas pruebas en base al número de estimadores, en particular se han obtenido resultado de pruebas con 10, 20, 50, 100 y 150 estimados para predecir una clase. Con el objetivo de no atosigar de tablas la memoria, mencionaré que los resultados obtenidos por los clasificadores que usan un número muy bajo de estimadores es muy pobre en cuanto al *recall*, sin embargo, la precisión de estos clasificadores es muy alta, dando como resultado unos valores de $F1$ muy bajos. Estos clasificadores devuelven pocos resultados pero los que devuelven siempre son ciertos, en la mayoría de las iniciativas son incapaces de otorgar una respuesta por lo que el valor de *recall* es bajo. Los mejores resultados que se han obtenidos para nuestra colección de documentos ha sido con las pruebas realizadas con 100 estimadores y será con el número que continuaremos realizando las siguientes pruebas.

### 4.2.1. Resultados con el extracto

Para esta prueba implementamos un clasificador en el que usaremos como información para entrenar únicamente el contenido almacenado en la marca **extracto**.

Los resultados obtenidos han sido los siguientes:

| | Resultados extracto | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.805 | 0.451 | 0.78079 |
| Micro | 0.758 | 0.385 | 0.51063 |

Cuadro 4.1: Resultados extracto RandomForest.

Vemos que aunque el *recall* no es muy alto, la *precision* si lo es. Esto nos indica que los resultados devueltos por este clasificador, son en la mayoría de los casos ciertos, no obstante el bajo *recall* nos indica que muchas iniciativas no es posible etiquetarlas con la información del extracto, esto hace que el numero de $FN$ se aumente y por consiguiente el *recall* se vea afectado.

### 4.2.2. Resultados con todo el discurso

Para la iniciativa entera hemos obtenido los siguientes resultados:

| | Resultados con todo el discurso | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.5795 | 0.3509 | 0.437116 |
| Micro | 0.4510 | 0.3068 | 0.3651 |

Cuadro 4.2: Resultados de Random Forest con todo el discurso.

Como podemos ver en la tabla obtenemos unos resultados bastantes por debajo de los obtenidos únicamente por el extracto. Esto nos indica que el uso de un conjunto de entrenamiento en el que disponemos de mucha información puede llevarnos a cometer errores a la hora de clasificar como podemos ver en el descenso de la *precision*, y aún mas, con la perdida perdemos *recall*, lo que nos indica que el clasificador no clasifica bien, cometes muchos errores.

### 4.2.3. Resultados con resúmenes de los párrafos

La siguientes tablas corresponde a los resultados obtenidos a partir de un resumen del 50 % y 25 % respectivamente;

| | Resultados con el resumen del 50 % de los párrafos. | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.5293 | 0.4025 | 0.457272 |
| Micro | 0.3974 | 0.3917 | 0.394529 |

Cuadro 4.3: Resultados con el resumen del 50 % de los párrafos con Random Forest.

| | Resultados con el resumen del 25 % de los párrafos. | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.7536 | 0.562 | 0.6419239 |
| Micro | 0.7067 | 0.5387 | 0.611368 |

Cuadro 4.4: Resultados con el resumen del 25 % de los párrafos con Random Forest.

Se puede comprobar por los resultados que a medida que disminuimos el tamaño del texto usado aumentamos precisión y lo que es mas importante el recall aumenta considerablemente.

### 4.2.4. Resultados con el extracto y un resumen del 25 %

En este caso vamos ha usar un resumen del 25 % del contenido de las iniciativas añadiéndole el contenido del *extracto*. En este caso es probable que los resultados se vean empeorados por falta de conocimiento a la hora de entrenar.

| | Resultados con el extracto y un resumen al 25 % del contenido | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.7735 | 0.6932 | 0.73115183 |
| Micro | 0.6927 | 0.6420 | 0,72056990 |

Cuadro 4.5: Resultados con el extracto y un resumen al 25 % del contenido

Por los resultados obtenidos podemos observar que al añadir el extracto la calidad del clasificador aumenta.

### 4.2.5. Evaluación RandomForest

En este apartado haremos una evaluación de los resultados obtenidos por el clasificador implementado con el algoritmo RandomForest y para las diferentes pruebas realizadas con el distinto contenido de ellas.

En particular esta técnica presenta unos resultados muy buenos para los diferentes conjuntos de entrada.

Vemos como el uso de parte del contenido almacenado en las marcas **párrafo** junto con el que se extrae de las marcas **extracto** proporciona información mas relevante sobre el documento que esas partes utilizadas de manera independiente. No obstante, el uso tanto del 25 % del contenido de las marcas **párrafos** como el de las marcas **extracto** es suficiente para clasificar correctamente gran parte de los documentos.

## 4.3. Evaluación resultados Naive Bayes

Ahora evaluaremos los clasificadores creados a partir del algoritmo Naive Bayes. Los resultados obtenidos a partir del distinto contenido usado por cada una de las pruebas serán expuestos a continuación.

### 4.3.1. Resultados con el extracto

Con el caso del clasificador hecho con RandomForest empezamos con los resultados obtenidos empleando solo el contenido del *extracto*.

| | Resultados extracto con Navie Bayes | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.6895 | 0.5392 | 0.6051573 |
| Micro | 0.6351 | 0.4020 | 0.4923540 |

Cuadro 4.6: Resultados extracto con Navie Bayes.

Vemos que los resultados son bastante aceptables, la mayoría de etiquetas que devuelve son correctas, además el *recall* es no es del todo bajo, lo que nos dice que aunque se equivoque el clasificador es capaz de generar patrones, aunque estos no son del todo correctos.

### 4.3.2. Resultados con todo el discurso

Con la iniciativa entera se obtienen los siguientes resultados:

| | Resultados con todo el discurso Naive Bayes | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.4294 | 0.3891 | 0.4082578 |
| Micro | 0.5065 | 0.2865 | 0.36598 |

Cuadro 4.7: Resultados de todo el discurso con Naive Bayes

En este caso tanto precisión como *recall* son muy bajos, esto quiere decir que a las pocas iniciativas que es capaz de etiquetar no las etiqueta bien.

Este *recall* tan bajo se debe a que el alto contenido usado para entrenar sobreajusta los datos del clasificador.

### 4.3.3. Resultados con resúmenes de los párrafos

Ahora procedemos a generar un clasificador usando solo un resumen del discurso.

| | Resultados resumen del 50 % con Navie Bayes | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.6092 | 0.5465 | 0.576149 |
| Micro | 0.6891 | 0.4971 | 0.577561 |

Cuadro 4.8: Resultados resumen del 50 % con Navie Bayes

| | Resultados con un resumen del 25 % con Naive Bayes | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.7492 | 0.4655 | 0.57422013 |
| Micro | 0.6571 | 0.3031 | 0.414844 |

Cuadro 4.9: Resultados con un resumen del 25 % con Naive Bayes.

Vemos que en este caso se acierta en la mayoría de casos, no obstante vemos que en muchos otros casos el clasificador prefiere no etiquetar, lo que aumenta el numero de falsos negativos (FN), y por consiguiente, se disminuye el recall.

### 4.3.4. Resultados con el extracto y un resumen del 25 %

Por último añadimos al 25 % del texto el contenido del extracto.

| | Resultados con el extracto y un resumen del 25 % | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.6335 | 0.5092 | 0.56458 |
| Micro | 0.6901 | 0.5519 | 0.613311 |

Cuadro 4.10: Resultados con el extracto y un resumen del 25 %.

Vemos que el recall y la precisión son muy parecidos. Los datos no son lo suficientemente buenos, pero en ambos casos el valor F1 supera el 50 %.

### 4.3.5. Evaluación de los resultados de Naive Bayes

En los resultados obtenidos por este clasificador podemos observar que el contenido de los párrafos añadido al del extracto no varía significativamente al obtenido con el extracto. Si este clasificador fuese elegido para realizar la aplicación sería recomendable hacerlo únicamente con el extracto, el resultado no compensa el tiempo que se emplea en resumir las iniciativas.

## 4.4. Resultados con SVM

De igual forma, haremos diferentes pruebas con los clasificadores generados a partir del algoritmo de aprendizaje automático SVM.

Aunque se han hecho pruebas con diferentes kernels para el SVM los resultados correspondiente a los kernel *polinomial* y *rbf* no han sido incluidos en las siguiente tablas porque tanto *precisión* y en *recall* son muy bajos. Esto se debe a que en la clasificación de textos las categorías para asignarlos suelen ser linealmente independientes, aunque se esperaba que los resultados no fuesen buenos se decidió tomar algunos valores los cuales comentare en las conclusiones que se aportan al final de esta sección para los resultados obtenidos con SVM.

### 4.4.1. Resultados con el extracto

Como para el resto de clasificadores comenzamos solo con el extracto.

| | Resultados extracto SVM | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.7805 | 0.6702 | 0.721156 |
| Micro | 0.7539 | 0.6031 | 0.670120 |

Cuadro 4.11: Resultados extracto SVM.

Los resultados usando solo el extracto son bastante buenos, aunque el recall es bajo, esto se debe a que en algunos casos antes de equivocarse se decide no etiquetar, con lo que el número de falsos negativos aumenta rápidamente.

### 4.4.2. Resultados con todo el discurso

En este caso vamos a usar el contenido de la iniciativa entera. Evidentemente, la inversión de tiempo para este clasificador es muy elevada, ademas no es recomendable usar un contenido tan amplio de información para cla-

sificar. Como en el caso del RandomForest al usarse tanta información se tiende a sobreajustar los resultados y dar respuestas poco fiables.

| | Resultados con todo el discurso | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.6245 | 0.4723 | 0.537839 |
| Micro | 0.43293954 | 0.3954 | 0.4133011 |

Cuadro 4.12: Resultados discurso completo con SVM.

### 4.4.3. Resultados con resúmenes de los párrafos

Los resultados de este clasificador para los resúmenes de los párrafos son:

| | Resultados resumen 50 % SVM | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.5284 | 0.4065 | 0.459502 |
| Micro | 0.6928 | 0.3910 | 0.49987 |

Cuadro 4.13: Resultados resumen 50 % SVM.

| | Resultados resumen 25 % | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0,8603 | 0.4956 | 0.628902 |
| Micro | 0,6832 | 0.3905 | 0.496953 |

Cuadro 4.14: Resultados resumen 25 % SVM.

Seguimos acertando en muchos la mayoría de los casos. Aún así dejamos muchas iniciativas sin etiquetar.

### 4.4.4. Resultados con el extracto y un resumen del 25 %

En este apartado veremos los resultados obtenidos por un resumen del 25 % del contenido de las marcas *párrafos*, y por el contenido de la marca *extracto*. Esta prueba pretende que a la hora de aprender el clasificador haga uso de la información del extracto, que por lo general es representativa del texto y de un pequeño resumen del contenido almacenado en los párrafos de la iniciativa.

Como demuestran los resultados estos resultados son mejores que los del resto de pruebas.

| | Resultados extracto y un resumen del 25 % | | |
|---|---|---|---|
| | Precisión | Recall | F1 |
| Macro | 0.8193 | 0.7529 | 0.78469 |
| Micro | 0.7096 | 0.6942 | 0.7018155 |

Cuadro 4.15: Resultados extracto y un resumen del 25 %.

### 4.4.5. Evaluación resultados SVM

Respecto a los diferentes kernel, a los cuales no he hecho referencia, diré que se han probado el kernel polinomial, rbf, y lineal. No obstante, dado que por lo general las categorías de texto suelen ser linealmente independientes, el uso de un kernel de otro tipo que no sea lineal aportará uno resultados muy por debajo de lo que se espera para un clasificador de este tipo.

En el caso del kernel polinomial los resultados para algunos conjuntos de las validación cruzada no superaba el 10 % para la precisión y el 3 % para el recall. De igual formal el kernel rbf aporta resultados parecidos. Por esto último, todas las pruebas se han realizado con kernel lineal.

Vemos que por lo general hemos sido capaces de obtener mejores resultados a la hora de clasificar con SVM que con RandomForest y con Naive Bayes.

Era de esperar que SVM aportara buenos resultados.

En el caso del uso único del extracto los resultados son bastante buenos, aunque se ven mejorados al introducir en el contenido de entrenamiento de cada iniciativa un resumen del 25 % del contenido de los párrafos. Cuando aumentamos el contenido haciendo uso de un 50 % del resumen notamos que la precision disminuye de manera insignificante, no obstante en el recall notamos que la disminución de puntuación es mucho menor. Esto se debe a que el clasificador utiliza demasiada información para entrenar, lo que sobreajusta al clasificador y otorga menos fiabilidad a las elecciones.

De igual forma al usar el contenido completo de la iniciativa los resultados continúan empeorando ademas de aumentar en gran medida el tiempo de entrenamiento del modelo y predicción de cada iniciativa.

## 4.5. Conclusión de los resultados de los algoritmos

Dado que el algoritmo de clasificación que devuelve mejores resultados ha sido el SVM, haremos uso de él a la hora de implementar una interfaz de usuario (GUI) para el uso por parte de terceros de esta aplicación.

El hecho de que SVM aporte mejores datos que RamdonForest y Naive Bayes no garantiza que sea siempre mejor. Estos resultados dependen estrictamente del conjunto de datos con el que trabajamos. En principio se esperaba que SVM respondiese mejor a la hora de clasificar documentos ya que generalmente las categorías en las que clasificamos los textos suelen ser linealmente dependiente, por esto SVM junto a un kernel lineal era de esperar que nos devolviese mejores resultados.

En estos resultados también influye en gran medida que la colección de documentos este formada por algo más de 5000 iniciativas, esto además de que tenemos unos 1400 descriptores, influye considerablemente en los resultados. En problemas de clasificación automática se recomienda que el conjunto de documentos con los que trabajemos sea lo suficientemente amplio, quizás con un número mas elevado de iniciativas (alrededor de 10000) los resultados de los clasificadores podrían haber sido mejores.

Respecto al contenido que vamos a usar de las iniciativas, dado que los resultados del uso del extracto y un resumen del 25 % aportan resultados mejores haremos uso de este contenido para la aplicación. Ademas cabe destacar que generar un modelo de este tipo de clasificador no conlleva mucho tiempo, aunque esto dependerá del número de documentos que empleemos para generarlo.

También se ha comprobado que el extracto es bastante representativo del contenido de la iniciativa. En cambio, cuanto más contenido introducimos para cada documento la información se ve corrompida, ya que se introducen términos que quizás no tienen mucho que ver con la materia en cuestión en la que se desea asociar dicha iniciativa.

# Capítulo 5

# Diseño y desarrollo de la aplicación

Dado que durante todo el trabajo, en lo que se refiere a implementación, hemos usado el lenguaje Python, haremos uso de algún tipo de librería disponible en este lenguaje para la implementación de una interfaz gráfica que muestre la utilidad práctica de lo desarrollado durante este trabajo. Actualmente existen varias librerías que permiten implementar una interfaz gráfica de manera sencilla. Entre estas librerías destacan dos de ellas; **TKinter** y **WxPython**. Ambas son multiplataforma y con características de potencia y madurez similares. *WxPython* se adapta mejor a la apariencia nativa del sistema de ventanas donde se ejecute, mientras *TKinter* mantiene un aspecto diferente.

*Tkinter* viene incorporado en la distribución de **CPython** por lo que no requiere de ningún tipo de instalación, aunque está disponible en *apt* (*yum*,*rpm*,...) junto con *WxPython*. La información extraída de foros y comunidades desarrolladores parece ser que el aprendizaje de Tkinter es más sencillo y más extendido al uso de *Python*, pero posee un problema considerable, y es que cada vez se implementan menos cosas nuevas y esta perdiendo la categoría de estándar a favor de *wxPtyhon*, con más *widgets*, de más alto nivel y con más herramientas relacionadas.

Dado que no he usado ninguno de ellos durante la carrera, intentaré hacer uso de las dos con el objetivo de obtener una aplicación simple pero con toda la funcionalidad necesaria para llevar a cabo la clasificación documental.

La implementación entregada como aplicación incorpora el método de clasificación con el que mejores resultados hemos obtenido. En nuestro caso el algoritmo usado será un *SVM* con *kernel lineal*, y el contenido de las iniciativas que usaremos para aprender un modelo será el formado por el *extracto* junto con un resumen de un 25 % del contenido de las marcas

*parrafos*.

Para la aplicación lo que se ha hecho es adaptar los códigos desarrollados en la fase de pruebas y evaluación de los clasificadores, de manera que resuelvan los distintos problemas que un documentalista podría tener a la hora de etiquetar una iniciativa.

Entre otras funcionalidades la aplicación debe permitir elegir un directorio donde se encuentren los documentos que se emplearán para la creación del modelo o entrenamiento del clasificador, guardar el modelo para una clasificación posterior, permitir cargar un modelo generado previamente, seleccionar una carpeta donde se encuentren los documentos a clasificar y como objetivo fundamental debe proveer al documentalista de una ayuda en la hora de toma de decisiones sobre la elección de las categorías que identifican un documento en concreto. Esto último se hará a través de una ventana en la que se dará opción de elegir entre las categorías que han sido previamente seleccionadas por el modelo por su alta probabilidad de que el documento se integre en ellas, como añadido, se debe proveer de una forma en la que esas categorías seleccionadas por el documentalista puedan ser salvadas dentro del documento como sucede de manera habitual en la documentación manual.

## 5.1. Diseño de la aplicación

Para desarrollar correctamente la aplicación será necesario definir un conjunto de casos de usos y su forma de interactuar con los actores (usuarios) que harán uso del sistema. Como paso anterior a la creación de los casos de uso definiremos un conjunto de requisitos que deberá satisfacer el sistema (*requisitos del sistema*).

De esta forma definimos los límites del sistema y las relaciones entre el sistema y su entorno.

Un *caso de uso* es un conjunto de escenarios, o secuencias de pasos, que tienen una meta en común. Un caso de uso es una descripción de un proceso, relativamente largo, que incluye varias etapas o transacciones. Un *diagrama de casos de uso* es una manera específica de utilizar el sistema, es una historia que describe un uso particular del sistema. Representa una imagen de una funcionalidad del sistema, desencadenada en respuesta a un actor externo.

Este sistema solo puede ser usado por una persona a la vez, en nuestro caso el actor será un documentalista.

También añadiremos un diagrama de clases y otro de interacción para reflejar de manera mas precisa el comportamiento de este sistema.

### 5.1.1. Requisitos del sistema

En primer lugar vamos a definir una serie de requisitos del sistema:

- RS1: Elegir directorio de iniciativas para entrenamiento.
  - Prioridad: Esencial
  - Descripción: Elegir el directorio donde se almacenan las iniciativas que serán usadar para entrenar un clasificador.
- RS2: Elegir directorio de iniciativas para la clasificación.
  - Prioridad: Esencial
  - Descripción: Elegir el directorio donde se almacenan las iniciativas (o iniciativa) que se desea clasificar.
- RS3: Generar un modelo
  - Prioridad: Esencial.
  - Descripción: Creación de un modelo de un clasificador entrenado capaz de ser usado para clasificar iniciativas posteriormente.
- RS4: Almacenar modelo generado.
  - Prioridad: Esencial
  - Descripción: A partir de un modelo previamente generado se debe dar la posibilidad de guardar el modelo en una ruta elegida por el usuario y con el nombre que se desee, para ser reutilizado.
- RS5: Clasificar iniciativas.
  - Prioridad: Esencial
  - Descripción: Capacidad de generar etiquetas para un conjunto de iniciativas a partir de un conjunto de iniciativas y de un modelo entrenado previamente.
- RS6: Mostrar elecciones con mayor posibilidad de etiquetar una iniciativa.
  - Prioridad: Esencial.
  - Descripción: Mostrar un conjunto de etiquetas con alta probabilidad de ser susceptibles de ser etiquetas de una iniciativa.
- RS7:Mostrar elecciones con una probabilidad menor, ordenadas en orden descendiente según su probabilidad.
  - Prioridad: Deseable.

  - Descripción: Es recomendable mostrar las elecciones que no han sido elegidas por el clasificador para que el usuario pueda comprobar si alguna de estas se ajusta a la iniciativa.

- RS8: Mostrar un resumen de la iniciativa.

  - Prioridad: Deseable
  - Descripción: Mostrar un resumen de la iniciativa para que sea facil para el usuario identificar el contenido de esta iniciativa en cuestión.

- RS9: Visualizar la iniciativa completa.

  - Prioridad: Deseable
  - Descripción: Visualizar la iniciativa completa tal y como se proporcionan (formato XML)

- RS10: Permitir seleccionar las iniciativas que se crean oportunas.

  - Prioridad: Esencial.
  - Descripción: El sistema debe permitir al usuario elegir un conjunto de las etiquetas para cada iniciativa.

- RS11: Guardar las etiquetas.

  - Prioridad: Deseable.
  - Descripción: Aunque no es una tarea esencial, es deseable que el sistema sea capaz de almacenar las materias seleccionadas por el usuario dentro de la iniciativa correspondiente.

- RS12: Elegir un modelo.

  - Prioridad: Esencial
  - Descripción: Es fundamental que el usuario tenga la posibilidad de elegir un modelo para poder realizar la clasificación de un conjunto de iniciativas.

- RS13: Cerrar ventana de entrenamiento.

  - Prioridad: Esencial
  - Descripción: el sistema debe permitir cerrar la ventana en la que se genera el modelo para permitir al usuario realizar otra de las opciones.

- RS14: Cerrar ventana de clasificación.

  - Prioridad: Esencial

  - Descripción: El sistema debe permitir cerrar la venta de clasificación para permitir al usuario realizar otras opciones.

- RS15: Pasar a otra iniciativa.

  - Prioridad: Esencial.
  - Descripción: El sistema debe permitir pasar a otra iniciativa sin necesidad de que se guarden los datos seleccionados.

- RS16: Cerrar ventana de elecciones.

  - Prioridad: recomendable.
  - Descripción: El sistema debe permitir cerrar la ventana de elecciones.

### 5.1.2. Descripción de los actores.

Dado que es una aplicación muy sencilla y que su uso esta dirigido al apoyo al personal que realiza esta tarea de manera manual se describe a un solo actor:

- Nombre: Documentalista
- Descripción: Representa a un usuario no identificado frente al sistema. Generalmente será un documentalista.

### 5.1.3. Descripción de los casos de uso.

- **Caso de uso 1**

  - Nombre: Generar Modelo (entrenar clasificador)
  - Autor: José Arcos Aneas
  - Fecha: 7/6/2015
  - Descripción: Genera un modelo de clasificador a partir de un conjunto de entrenamiento.
  - Actores: Documentalista
  - Precondiciones: Deberá haber sido especificado un conjunto de iniciativas con las que poder entrenar el modelo.
  - Flujo normal: 1. Elegir un directorio donde se haya almacenado un conjunto de iniciativas. 2. Pulsar el botón con el que el sistema generará el modelo. 3 Almacenar el modelo con el nombre deseado.

- Poscondiciones: Existirá una variable en la que se haya almacenado el modelo.

- **Caso de uso 2**

  - Nombre: Clasificación de iniciativas
  - Autor: José Arcos Aneas
  - Fecha: 7/6/2015
  - Descripción: A partir de un modelo previamente generado se realizará la clasificación de un conjunto de iniciativas.
  - Actores: Documentalista
  - Precondiciones: Tener un modelo entrenado y almacenado.
  - Flujo normal: 1. Seleccionar un modelo. 2. Seleccionar un conjunto de iniciativas. 3. Clasificar el conjunto de iniciativas.
  - Poscondiciones: Una variable del sistema almacenará el conjunto de etiquetas preseleccionadas para cada iniciativa, asi como un resumen de cada iniciativa.

El diagrama de caso se muestra a continuación:

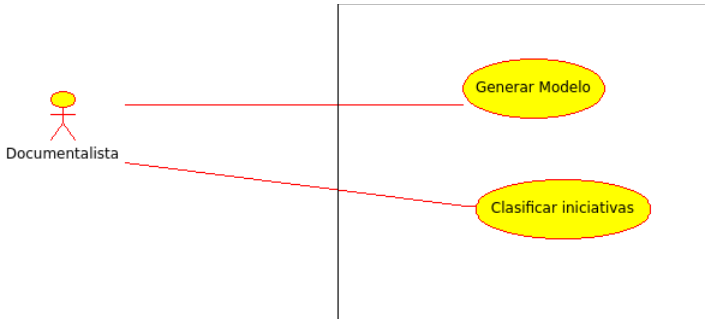



Figura 5.1: Diagrama de casos de uso

Ahora mostraremos el diagrama de clases desarrollado para esta aplicación:

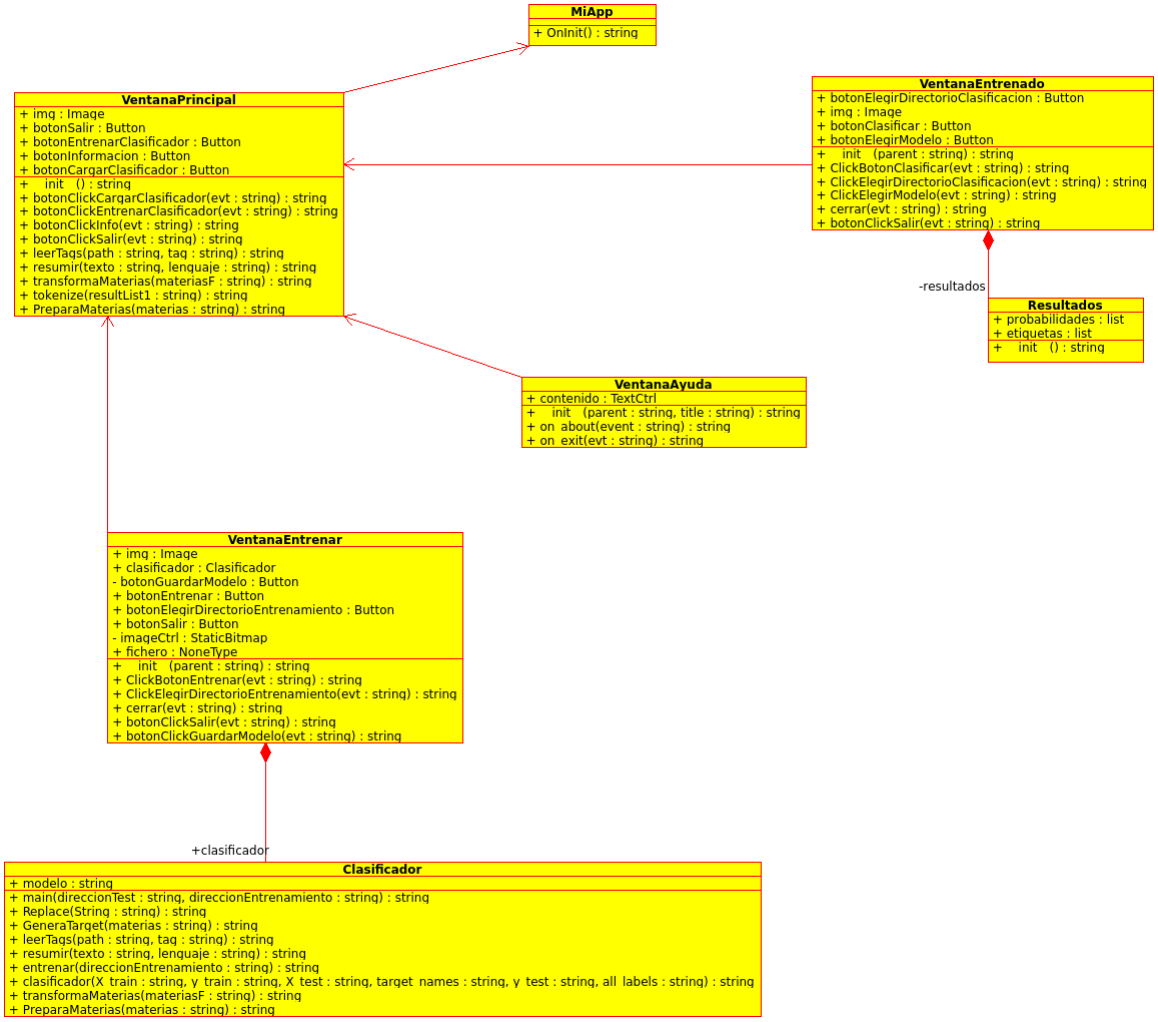



Figura 5.2: Diagrama de clases

## 5.2. Instalación de la aplicación.

Dado que hacemos uso de herramientas proveídas por terceros debemos asegurarnos de que estas dependencia están previamente instaladas en la máquina en que se ejecutará la aplicación.

Para realizar esta tarea se han desarrollado varios archivos. Uno de ellos es el que hace uso de la herramienta *setuptools* y que permite instalar dependencias externas a partir de un archivo donde debemos especificar el módulo a instalar junto a la versión de está de la que se hará uso. Para esto se ha desarrollado el archivo **setup.py**. El proble-

ma es que no se realiza la instalación de forma directa y que además no es posible instalar todas las dependencias necesarias. Por este motivo, se han realizado un *script* ( con nombre *install.sh* ) con el que con un simple comando podremos instalar la totalidad de las dependencias externas de las que hacemos uso.

El proceso a seguir, es sencillo. En primer lugar será necesario dar permisos de ejecución al archivo *install.sh*. Una vez hecho esto, debemos ejecutar el archivo asegurándonos de tener acceso a Internet y privilegios de administrador para que podamos realizar la descarga e instalación de las dependencias.

## 5.3. Solución presentada y pasos para su uso

Esta esta sección se presenta el resultado final de la aplicación, también se explican los pasos a seguir para hacer uso de ellas.

La aplicación consta de cuatro ventanas principales, que permiten llevar a cabo la clasificación de un conjunto de iniciativas. La aplicación se inicia con una ventana principal que da acceso a las dos funcionalidades del sistema; generar un modelo y realizar la clasificación de un conjunto de iniciativas.

Si existe un modelo ya generado es posible realizar la clasificación a partir de este. En caso de que no exista un modelo generado debemos generarlo antes de poder clasificar las iniciativas.

### 5.3.1. VENTANA PRINCIPAL

Esta ventana es la ventana principal del programa, desde la que podremos elegir cuatro opciones:

- 1. Entrenar clasificador Esta opción permite la creación de un clasificador, guardar el modelo y clasificar un conjunto de iniciativas.
- 2. Clasificar iniciativas Esta opción permite cargar un modelo de un clasificador previamente entrenado. También permite la clasificación de iniciativas.
- 3. Ayuda Este botón muestra la pantalla actual.
- 4. Cerrar. Cierra la aplicación.


Figura 5.3: Imagen de la página principal de la aplicación

### 5.3.2. ENTRENAR CLASIFICADOR

Esta ventana se inicia al clicar en el botón .Entrenar Clasificador". La ventana ofrece una serie de opciones que dan la posibilidad de generar un clasificador, guardar el modelo, elegir un directorio de entrenamiento y clasificar las iniciativas que contenga.

Los botones y su funcionalidad se muestran a continuación:

- Directorio Entrenamiento. Con este botón elegimos el directorio que contendrá las iniciativas que servirán para entrenar un clasificador. Aunque no es necesario, se aconseja sea lo que se realice en primer lugar. Tenga en cuenta que el tiempo que se puede dedicar a esta tarea dependerá del número de iniciativas que se empleen en el entrenamiento.
- Guardar modelo. Una vez entrenado el clasificador se aconseja guardar el modelo para poder cargarlo y ser usado para posteriores clasificaciones. Esto reducirá en gran medida el tiempo en realizar la clasificación, ya que suprimimos parte del tiempo que se dedica a entrenar el clasificador.
- Directorio entrenamiento. Este botón permite seleccionar un directorio de entrenamiento para realizar la clasificación. El directorio deberá contener el conjunto de iniciativas que se desean clasificar.
- Cerrar Este botón cierra la ventana actual y nos devuelve a la ventana inicial.


Figura 5.4: Imagen de la ventana de entrenamiento de la aplicación

### 5.3.3. CARGAR MODELO

Para acceder a esta ventana será necesario clicar en el botón Çargar modelo"desde la ventana inicial. Una vez desplegada la ventana serán posibles una serie de opciones para realizar la clasificación de un conjunto de iniciativas. A diferencia de la ventana anterior será posible cargar un modelo y reducir el tiempo dedicado a la clasificación. Los distintos botones disponibles en esta ventana junto a su funcionalidad se muestran a continuación. A continuación se describe la funcionalidad de cada uno de los diferentes botones disponibles en esta ventana.

- 1. Elegir modelo. Este botón permite cargar un archivo que contenga un modelo de un clasificador previamente entrenado. Esta opción reduce el coste de tiempo en llevar a cabo la clasificación.
- 2. Directorio clasificación Este botón permite cargar un directorio que contendrá las iniciativas que se desean clasificar.
- 3. Clasificar En el caso en que se haya elegido un modelo y un directorio de clasificación, esta opción permite clasificar el conjunto de iniciativas. Una vez se haya realizado la clasificación, emergerá de manera sucesiva una ventana por cada una de las iniciativas que se desean clasificar, donde será posible la elección de las etiquetas con las que clasificar una iniciativa.
- 4. Cerrar. Botón que permite cerrar la ventana actual.


Figura 5.5: Imagen de la ventana de clasificación de la aplicación

### 5.3.4. VENTANA ELECCIONES

Esta ventana permite la elección de una serie de etiquetas, previamente seleccionadas por el clasificador, distinguiendo entre las que poseen mayores posibilidades de ser apropiadas y las que poseen una posibilidad menor. Esta ventana mostrará el contenido de parte de la iniciativa, además de una serie de opciones para hacer más fácil la clasificación.

- 1. Guardar Esta opción salva las materias seleccionadas dentro del campo materias de la iniciativa a la que corresponda. En caso de no contener el nodo materias lo creará. En caso de existir el campo materias incluirá el contenido seleccionado junto al ya existente.
- 2. Visualizar iniciativa. Este botón permite visualizar una iniciativa. El entorno para la visualización será **Firefox**.
- 3. Siguiente. Este botón estará visible siempre que no estemos en la última iniciativa de las que nos disponemos a clasificar. Permite pasar a la página de selección de la próxima iniciativa.
- 4. Cerrar. Será visible en la última iniciativa y permite cerrar la página actual. Nos dirige a la página Çargar modelo".

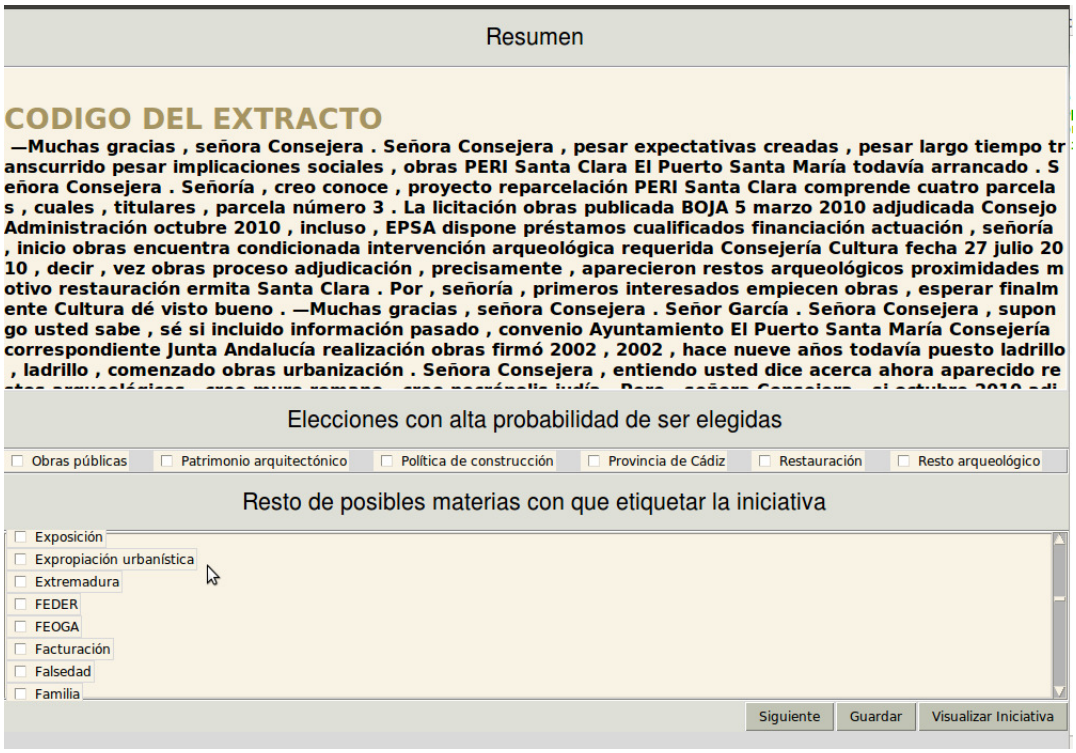



Figura 5.6: Imagen de la ventana de elecciones de la aplicación

# Capítulo 6

# Conclusiones y trabajo futuro

El hecho de incluir trabajo futuro muestra que soy consciente de que tanto los clasificadores implementados para realizar las pruebas como la aplicación presentada son muy mejorables.

Respecto a la implementación de los clasificadores, como se comento al principio de la memoria, en muchos casos se suelen combinar varios clasificadores para mejorar el resultado de los clasificadores de manera individual. Esto no ha sido considerado ya que el tiempo de creación de un modelo es lo muy elevado e incluir la creación de un nuevo modelo haría que se tardase una eternidad en poder llevar a cabo la tarea de clasificación. Pese a esto, dado que es posible relajar la probabilidad de que una materia sea o no seleccionada para una iniciativa hemos obtenido resultados bastante aceptables.

En principio se asume que todas las iniciativas son correctamente clasificadas algo que no es del todo cierto. En muchos casos las iniciativas no están correctamente etiquetas ya que este trabajo se realiza de manera individual por un solo documentalista, hecho que aumenta la posibilidad de que algunas materias no estén correctamente asignadas, lo que inevitablemente perjudica la calidad de nuestro clasificador. Otras iniciativas no están ni siquiera etiquetas por lo que ha sido necesario extraerlas del conjunto de entrenamiento para evitar errores a la hora de entrenar el modelo. Aunque sea tenido en cuenta a la hora de realizar las pruebas no se ha incluido en la aplicación ya que no me ha parecido oportuno eliminar las iniciativas del usuario, por lo que si se requiere el uso de esta aplicación seria necesario tener en cuenta esta cuestión. También me parecía curioso ver si iniciativas que no habían sido etiquetadas previamente podrían ser etiquetadas por el clasificador. En algunos casos el clasificador no propone ninguna,

pero en otros casos, iniciativas que no habían sido etiquetadas por los documentalistas han sido etiquetadas por el clasificador con bastante coherencia.

Respecto a la aplicación gráfica; ha sido implementada sin utilizar ningún tipo de herramienta para facilitar el proceso, o sea, ha sido *picando* directamente todo el código. No he trabajado mucho con interfaces gráficas por lo que no conocía muy bien la forma de implentarlas, esto ha influido considerablemente en el tiempo que empleado en esta tarea y también en el resultado obtenido.

A pesar de todo esto, la aplicación presenta una utilidad real para aconsejar a los documentalistas en sus tareas de decisión. La aplicación además de presentar una serie de *materias* con mucha probabilidad de ser a las que pertenece la iniciativa, muestra el resto de *materias* en orden decreciente de probabilidad, con lo que el propio documentalista puede buscar entre ellas las que considere mas oportuno para asignar a dicha iniciativa. Esto proporciona al documentalista una ayuda real a la hora de tomar una decisión.

## 6.1. Problemas encontrados

A la hora de procesas los archivos proporcionados, nos encontramos con algunos no pueden ser leídos correctamente. Esto en algunos casos se debe a que no están bien formados, en otros, se debe a que no poseen ningún contenido en las marcas que se indicaban para su lectura. Este problema se salvo sin muchos problemas suprimiendo estas iniciativas a la hora del entrenamiento del clasificador.

Otro problema surge por el hecho de que es muy costoso realizar algunas pruebas, el tiempo empleado en este aspecto ha sido muy elevado. Recomiendo salvar los modelos siempre que sea posible, para evitar perdidas de tiempo innecesarias. Pasa lo mismo a la hora de resumir las iniciativas, es aconsejable realizar los resúmenes una sola vez para realizar las posteriores pruebas a partir de los archivos ya resumidos.

El mayor problema surgió a la hora de realizar la implementación de la interfaz gráfica. Como el trabajo empezó con Python, no me pareció oportuno cambiar de lenguaje a la hora de realizar esta tarea. El hecho de no disponer de ninguna herramienta que facilitase la programación de la aplicación supuso un problema. Hubiese sido mas sencillo realizar este trabajo en C/C++ o en Java.

# Apéndice A

# Guía de instalación de la aplicación

Este apéndice se encarga de explicar de el proceso de instalación de manera mas profunda a la que se ha hecho anteriormente. La entrega se compone de diferentes carpetas entre ellas: la carpeta que contiene la documentación, la carpeta que contiene el código de pruebas y una última que contiene la aplicación desarrollada. En este apéndice nos centramos en la carpeta que contiene la aplicación.

En esta carpeta inicialmente nos encontramos con dos carpetas; **iniciativas** y **modelo**. En la primera de ellas, la carpeta iniciativas, nos encontramos con dos carpetas; *iniciativasTest* y *iniciativasTraining*. Estas carpetas se almacenarán las iniciativas que van a ser usadas para el entrenamiento y para realizar el test. En la carpeta *modelos* se recomienda almacenar los modelos, aunque no es totalmente necesario, para facilitar su uso. En la carpeta *aplicacion* encontramos también una serie de archivos: *Aplicacion.py*, *script.sh*, *setup.py*, *leeme.txt* y por último una imagen ( *images.jpg*). El archivo *Aplicacion.py* es el archivo que contiene el código fuente de la aplicación, este archivo hace uso de la imagen. El archivo *leeme.txt* sólo contiene un poco de información sobre la instalación y el funcionamiento de la aplicación. Por último, los ficheros *script.sh* y *setup.py* son los que se usan para la instalación. La instalación, para facilitar la instalación por parte de un usuario no experto, se realiza directamente desde el archivo *script.sh*. Para instalar las dependencias únicamente será necesario dar permisos de ejecución al archivo *script.sh*:

*sudo chmod +x script.sh*

El paso siguiente será ejecutar este archivo. Se recomienda hacerlo

desde terminal ya que será necesario hacerlo con privilegios de administrador para que se permita la instalación de las dependencias. Esto puede hacerse de la siguiente manera:

*sudo .\script.sh*

Es posible que si no estamos registrados como *root* sea necesario introducir la contraseña antes de que se inicie la instalación. Una vez hecho se iniciará el proceso de instalación de todas las dependencias. También se ha incluido un comando para dar privilegios al archivo *Aplicacion.py*, con lo que será posible iniciar el uso clicando encima del archivo. Alternativamente también podemos iniciar la aplicación haciendo uso del terminal:

*.\Aplicacion.py*

Con este proceso la aplicación podría ser usada con toda funcionalidad.

# Apéndice B

# Guía para el usuario

Este apéndice se centra inicialmente en el uso de la aplicación. Para que sea más sencillo su uso se propone el siguiente procedimiento.

Inicialmente sería recomendable incluir las iniciativas dentro de la carpeta *Aplicación*, dentro de esta diferenciamos las iniciativas que se emplearán para generar el modelo y las iniciativas que se usarán para ser etiquetas.

Una vez hecho esto podemos iniciar la aplicación para comenzar el uso de esta.

Iniciamos la aplicación y se nos propone generar un modelo o cargarlo para hacer uso de este.

Supondremos que inicialmente no disponemos de un modelo generado.Nuestro primer paso será clicar en la opción para generar un modelo. Si no sabemos como actuar se recomienda clicar el botón *Ayuda*. Una vez estamos en la ventana de entrenamiento o generación del modelo debemos elegir un directorio de entrenamiento, posteriormente clicaremos el botón que se indica para generar un modelo. El proceso de generación del modelo depende directamente del número de iniciativas que van a ser empleadas para el entrenamiento. Este proceso generará un archivo que puede llegar a ocupar varios gigabytes. Una vez terminado el proceso clicaremos el botón para guardarlo, le daremos un nombre y lo almacenaremos en la carpeta *modelo*. Hecho esto ya habremos realizado la mitad del proceso. Cerraremos la ventana actual, volviendo a la ventana de inicio.

El proceso que se explica a continuación corresponde a la situación en la que ya disponemos de un modelo generado.

Clicaremos el botón *Cargar Clasificador*, esto nos dirigirá a la ventana de clasificación. Una vez en la ventana podremos elegir el modelo deseado para la clasificación y el directorio donde se encuentran las

iniciativas que se desean clasificar. Elegidos el modelo y las iniciativas, clicaremos el botón *Clasificar*. Dependiendo el modelo usado y del tamaño del directorio que contiene las iniciativas a clasificar el proceso de clasificación será mas o menos costoso.

Cuando el proceso haya acabado se desplegará una ventana para cada iniciativa donde podremos seleccionar las etiquetas que consideremos mas apropiadas para cada iniciativa. En estas ventanas podremos elegir las etiquetas que deseemos, así como guardarlas dentro del archivo o si por el contrario no queremos podemos pasar a la siguiente iniciativa. Cuando lleguemos a la ultima el proceso habrá terminado y por consiguiente se el proceso de clasificación de las iniciativas seleccionadas.

Para terminar cerraremos la ventana actual de clasificación y posteriormente la ventana inicial de la aplicación.

Para un posterior uso, ahora que disponemos de un modelo, podremos generar otro modelo si así lo deseamos o hacer uso del que ya disponemos para clasificar. Si se han clasificado correctamente las iniciativas anteriores se recomienda trasladarlas a la carpeta de *iniciativasTrain* y volver a generar el modelo para retroalimentar el clasificador y mejorar su funcionamiento.

# Apéndice C

# Herramientas utilizadas para el desarrollo del proyecto

Algunas de estas herramientas ya han sido comentadas en el capítulo dedicado al desarrollo, no obstante ahora mencionaremos también la versión de estas que se ha usado, dado que una versión anterior, o posterior, podría no trabajar correctamente.

## C.1. Software usado

En primer lugar he de mencionar que la totalidad de las herramientas usadas en este trabajo son de software libre. Esto quiere decir que se puede hacer uso de esta aplicación de manera totalmente gratuita.

Empezando por el sistema operativo el elegido ha sido *Ubuntu 12.04 LST*, aunque el desarrollo del archivo de instalación ha sido probado con éxito en la versión *14.04* de *Ubuntu*. He elegido la plataforma *Ubuntu* por ser con la que mas he trabajado estos último años, ademas de las gran cantidad de opciones que ofrece para el desarrollo de aplicaciones.

El entorno de programación que he usado ha sido Spyder, este entorno da muchas facilidades a la hora de programar y es fácil de usar. El lenguaje de programación ha sido Python y la versión de este la 2.7.3.

Para el procesamiento del texto se ha hecho uso de la versión 3.0.2 de **NLTK**. La versión de **Summa** usada para realizar resúmenes de ha sido la 0.0.7, que en el momento de su uso era la última disponible.

Para usar los algoritmos de clasificación también se ha empleado la última versión de **Scikit-Learn**, esta versión es la 0.16.1. El uso de una versión posterior hará que la aplicación no funcione. Esta versión no esta disponible aún en los repositorios de *Ubuntu*, su instalación debe ser manual.

Para almacenar los modelos se ha usado una herramienta llamada *pickle-converter*. Esta herramienta nos permite crear modelos persistentes, con lo que es posible almacenar un modelo para su posterior uso.

La versión usada de **WxPython** ha sido la 2.8.11.0, y aunque no se ha probado, dado que sus actualizaciones son frecuentes no se puede garantizar que el uso de otra versión mantenga los mismos *widgets* que en la versión usada.

El resto de herramientas, tales como : pypi , setuptools,... no son mencionadas dado que su uso no varia de una a otra versión, realizandose sobre ellas sólo aplicaciones y no modificaciones.

Para evitar problemas a la hora de poner a punto para su uso esta aplicación, se aconseja hacer uso del archivo *install.sh*, en el que ya se han tenido en cuenta las versiones usadas para poder hacer uso de esta aplicación por aquellos que así lo deseen.

Para la realización de los diagramas se ha usado un software libre llamado *Umbrello*. Este programa permite generar diagramas de todo tipo de una manera sencilla. Facilita el uso el hecho de poder importar código lo que reduce el tiempo en diseñar las clases.

La documentación sobre la aplicación se ha desarrollado con una popular herramienta llamada *epydoc*. Es similar a *doxygen*, he usado *epydoc* porque es muy sencilla de usar y porque en la actualidad es la que utilizo para documentar las aplicaciones en las que trabajo. Esta herramienta permite crear documentación tanto en formato *.html*, como en formato *.pdf*. Para este trabajo la documentación se aporta únicamente en formato *.html*.

## C.2. Hardware usado

Respecto al hardware empleado para este trabajo se cita a continuación:

- Memoria:
  - description: ATA Disk
  - product: ST500DM002-1BD14

  - vendor: Seagate
  - physical id: 0.0.0
  - bus info: scsi@1:0.0.0
  - logical name: /dev/sda1
  - version: KC45
  - serial: Z3TQPLQ9
  - size: 465GiB (500GB)
  - capabilities: partitioned partitioned:dos
  - configuration: ansiversion=5 signature=50a6b66e
- CPU:
  - description: CPU
  - product: Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz
  - vendor: Intel Corp.
  - physical id: 4
  - bus info: cpu@0
  - version: Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz
  - serial: To Be Filled By O.E.M.
  - slot: LGA1155
  - size: 1900MHz
  - capacity: 3800MHz
  - width: 64 bits
  - clock: 100MHz
- RAM:
  - Array Handle: 0x0053
  - Error Information Handle: 0x0057
  - Total Width: 64 bits
  - Data Width: 64 bits
  - Size: 8192 MB
  - Form Factor: DIMM
  - Set: None
  - Locator: ChannelA-DIMM0
  - Bank Locator: BANK 0
  - Type: DDR3
  - Type Detail: Synchronous
  - Speed: 1333 MHz
  - Manufacturer: Kingston
  - Serial Number: 7F332142
  - Asset Tag: 9876543210

  - Part Number: 99U5471-038.A00LF
  - Rank: 2
  - Configured Clock Speed: 1333 MHz

- **A**
  - *Agrovoc*: Se forma por la unión de las palabras agricultura y vocabulario. Es un vocabulario multilingüe controlado, elaborado por la FAO.
- **B**
  - **Browsing**: Definido por *Xia Lin* como *ün proceso interactivo en el que uno puede visualizar grandes cantidades de información, percibir o encontrar estructuras o relaciones, y seleccionar ítems, centrando su atención visual en ellos"* https://es.wikipedia.org/wiki/Browsing
- **D**
  - **DOM**: Librería en lenguaje Python para la lectura, creación y modificación de archivos XML.
- **E**
  - **Eurovoc**: Tesauro multilingüe de la Unión Europea.
- **F**
  - **FAO**: siglas en inglés de *Food and Agriculture Organization*,es la Organización de las Naciones Unidas para la Alimentación y la Agricultura.
- **I**
  - **Idf**: Frecuencia inversa de los documentos.
- **M**
  - **MeSH**: Acrónimo de Medical Subject Headings, nombre que recibe el vocabulario terminológico controlado para publicaciones de artículos y libros de ciencia.
- **N**
  - **NLM**: Biblioteca Nacional de Medicina de Estados Unidos, acrónimo del inglés National of Medicine.
  - **NLTK**:
- **R**
  - **R**(Lenguaje): R es un lenguaje y entorno de programación para análisis estadístico y gráfico.
- **S**
  - **Stemming**: método para reducir una palabra a su raíz.

- **SVM**: Siglas del inglés *Support Vector Machine*, se refiere a las máquinas de soporte vectorial o máquinas de vectores de soporte.

- **T**
  - **Tf**: Frecuencia de un término dentro de un documento.
  - **Tf-idf**: Frecuencia de ocurrencia del término en la colección de documentos.
  - **Tkinter**: Adaptación de la biblioteca gráfica *Tcl/Tk* para el lenguaje de programación Python. Se considera un estándar para la interfaz gráfica de usuario para Python.
- **W**
  - **WxPython**: Adaptación de la librería gráfica wxWidgets para el lenguaje de programación Python.
- **X**
  - **XML**: Siglas del inglés *eXtensible Markup Language*,es un lenguaje de marcas desarrollado por el *World Wide Web Consortium*

# Bibliografía


[1] F. Bordignon and W. Panessi. Processing morphological variants in searches of spanish texts. *REVISTA INTERAMERICANA DE BIBLIOTECOLOGIA*, 24(1):69–88, 2001.

[2] R. Boulton. *Stemmer Snowall.* snowball.tartarus.org.

[3] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[4] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] L. Breiman. Manual on setting up, using, and undertanding random forest v4.0. *Statistics Deparment University of California Berkeley*, 2002.

[6] C. J. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[7] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods.* Cambridge university press, 2000.

[8] L. M. de Campos, J. M. Fernández-Luna, and J. F.Huete. Automatic indexing from a thesaurus using bayesian networks: Application to the classification of parliamentary initiatives. Technical report, Lecture Notes in Computer Science (LNCS), 2007.

[9] O. Dekel and O. Shamir. Multiclass-multilabel classification with more classes than examples, 2011.

[10] N. developers. *NLTK, Natural Language Toolkit.* www.nltk.org.

[11] C. Elkan. Evaluating classifiers. Technical report, University of California, San Diego, 2011.

[12] J. M. Fernández-Luna, J. F.Huete, and G. I. Osorio. Indexación de iniciativas parlamentarias mediante clasificación documental colaborativa. Technical report, Universidad de Granada, CITIC-UGR, 2012.

[13] M. Galar, A. Fernández, E. Barrenechea, and F. Herrera. Una debilidad de la estrategia uno-contra-uno en clasificación: Potenciando las clases dificiles.

[14] I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large vc-dimension classifiers. In *Advances in Neural Information Processing Systems*, pages 147–155. Morgan Kaufmann, 1993.

[15] T. K. Ho. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832–844, 1998.

[16] F. F. Luna, J. M.Huete, and J. Francisco. Recuperación de información. un enfoque práctico y multidisciplinar. In *Recuperación de información. Un enfoque práctico y multidisciplinar*, pages 147–155. RA-MA EDITORIAL, 2007.

[17] C. M. Luque. Clasificadores bayesianos. Technical report, Universidad Antonio Nebrija, 2003.

[18] V. T. M. Ikanomakis, S. Kotsiantis. Text classification using machine learning techniques. Technical report, University of Patras, Greece, 2011.

[19] A. J. O. Martos and M. Ángel García Cumbreras. Detección automática de spam utilizando regresión logística bayesiana. Technical report, Universidad de Jaén, 2005.

[20] T. Mitchell. Machine learning. *Ed. McGraw-Hill*, 1997.

[21] A. Moore. Cross-validation for detecting and preventing overfitting. Technical report, Carnegie Mellon University, 2005.

[22] R. A. Mur. Evaluación de técnicas de aprendizaje. Universidad Carlos III de Madrid.

[23] K. P. Murphy. Naive bayes classifiers. *University of British Columbia*, 2006.

[24] P. G. OTERO and M. G. GONZÁLEZ. Técnicas de procesamiento del lenguaje natural en la recuperación de información. Technical report, Centro de Investigación sobre Tecnoloxías da Lingua (CITIUS) Universidade de Santiago de Compostela, 2012.

[25] R. M. A. Pérez. *Búsqueda de respuestas en fuentes documentales multilingües*. PhD thesis, Instituto Nacional de Astrofísica, Óptica y Electrónica, México, 2008.

[26] Scikit-Learn. *Classification of text documents using sparse features*.

[27] C. Sutton. Nearest-neighbor methods. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(3):307–309, 2012.

[28] A. C. Vasquez, L. P. Concepción, O. R. Lazo, and R. CalmetAgnelli. Categorización de textos mediante máquinas de soporte vectorial. Technical report, Universidad Nacional Mayor de San Marcos, 2013.